AWLP: BUILDING A CUSTOM WIRELESS ACCESS POINT USING OPEN SOURCE SOFTWARE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ALPTEKİN ÇAKIRCALI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

SEPTEMBER 2005

Approval of the Graduate School of Informatics Institute

_____

Assoc. Prof. Dr. Nazife BAYKAL

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Dr. Onur DEMİRÖRS

Head of the Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Dr. Nazife BAYKAL

Supervisor

Examining Committee Members

Prof. Dr. Semih BİLGEN	METU / EE	_____
Assoc. Prof. Dr. Nazife BAYKAL	METU / IS	_____
Dr. Altan KOÇYİĞİT	METU / IS	_____
Assis. Prof. Dr. Y. Murat ERTEN	TOBB ETU / CENG	_____
Assoc. Prof. Dr. Erol SAYIN	METU / IE	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: **Alptekin Çakırcalı**

Signature: _____

ABSTRACT

AWLP: BUILDING A CUSTOM WIRELESS ACCESS POINT
USING OPEN SOURCE SOFTWARE

Çakırcalı, Alptekin
M.S., Department of Information Systems
Supervisor: Assoc. Prof. Dr. Nazife BAYKAL

September 2005, 74 pages

Almost all commercially available wireless access devices are special embedded systems with proprietary software that do not allow any modifications. Modifications to these systems are only possible by firmware upgrades released by manufacturers. However, release times of these firmware upgrades are unpredictable, and they are usually for bug-fix purposes rather than being feature and capability improvements. Thus, these devices fail to provide the needed flexibility. Ability to provide timely custom solution that is well integrated into existing network infrastructure is the key factor for a successful wireless access service implementation. In this thesis work, the open source software called AWLP is designed, coded, tested,

and released to public as a viable alternative for creating custom wireless access device.

Keywords: Wireless Access Point, Open Source Software, Perl, 802.11, HostAP

ÖZ

AWLP: AÇIK KAYNAK KODLU YAZILIM KULLANARAK İHTİYACA ÖZEL KABLOSUZ ERİŞİM CİHAZI YAPMAK

Çakırcalı, Alptekin
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Danışmanı: Doç. Dr. Nazife BAYKAL

Eylül 2005, 74 sayfa

Neredeyse tüm ticari kablosuz erişim cihazları, üzerinde değişiklik yapma imkanı olmayan kapalı kaynak kodlu, özel üretim, gömülü sistemlerdir. Bu tür cihazlar üzerinde değişiklik yapmak ancak üretici tarafından çıkarılan, bellenim tabir edilen kapalı kodlu aygıt yazılımları ile mümkündür. Fakat üreticilerin bu bellenimleri ne zaman çıkaracakları belirsizlik içermektedir. Bu bellenimler özellik ve kapasite iyileştirmeye yönelik olmaktan ziyade hata gidermeye yöneliktir. Dolayısıyla, bu tür cihazlar gerekli esnekliği sağlamada başarılı değillerdir. Başarılı bir kablosuz erişim servis uygulamasının anahtarı mevcut ağ altyapısıyla iyi bütünleşebilen ihtiyaca özel çözümleri vakitli bir şekilde geliştirebilmektir. Bu tez çalışmasında, özel kablosuz erişim cihazları oluşturmak isteyenler için geçerli bir seçenek olma iddiası taşıyan

AWLP isimli açık kaynak kodlu bir yazılım tasarlanmış, kodlanmış, test edilmiş ve yayınlanmıştır.

Anahtar Kelimeler: Kablosuz Erişim Noktası, Açık Kaynak Kodlu Yazılım, Perl, 802.11, HostAP

This thesis is dedicated to my family,

For their support,
For their love…

ACKNOWLEDGEMENT

This thesis is the result of three years of work on the subject. I would like to express my gratitude in general to all people who have supported me directly or indirectly, personally or technically during the process.

I express my sincere appreciation to Assoc. Prof. Dr. Nazife BAYKAL for her confidence in me. Without her guidance, encouragement, and support, this thesis would not be possible.

I have used computer and network equipments available in the Department of Industrial Engineering, Middle East Technical University for development and testing purposes throughout this study. I would like to thank all people in the department who made this possible for me.

I would also like to thank Halil Özbey for his timely help on formatting issues throughout the writing process.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AAA	Authentication Authorization Accounting
ACL	Access Control List
AP	Access Point
ASCII	American Standard for Character Information Interchange
AWLP	Alptekin's Wireless Linux Project
BASH	Bourne-Again Shell
BIND	Berkeley Internet Name Daemon
BSD	Berkeley Software Distribution
BSS	Basic Service Set
CD	Compact Disc
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DTIM	Delivery Traffic Indication Message
EAP	Extensible Authentication Protocol
FQDN	Fully Qualified Domain Name
HDD	Hard Disk Drive
HEX	Hexadecimal
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
GB	Giga Byte
GNU	GNU is Not Unix (a recursive abbreviation)
GPL	General Public License

Hz	Hertz
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISA	Industry Standard Architecture
IV	Initialization Vector
ISM	Industrial Scientific & Medical
LAN	Local Area Network
MAC	Medium Access Control
MB	Mega Byte
Mbps	Mega Bit Per Second
METU	Middle East Technical University
NAT	Network Address Translation
NIC	Network Interface Card
NTP	Network Time Protocol
OS	Operating System
OUI	Organizationally Unique Identifier
PCI	Peripheral Component Interconnect
PERL	Practical Extracting and Reporting Language
PHY	Physical
PCMCIA	Personal Computer Memory Card International Association
PRISM	Programmable Radio - Industrial Scientific & Medical
PTR	Pointer Record
RADIUS	Remote Authentication Dial-In User Service
RAM	Random Access Memory
RX	Receiver
SGID	Set Group ID
SNMP	Simple Network Management Protocol
SNR	Signal-to-Noise Ratio
SSH	Secure Shell
SSID	Service Set Identifier
SUID	Set User ID

TX	Transmitter
URL	Uniform Resource Locator
VA	Volt Amper
WAN	Wide Area Network
WDS	Wireless Distribution System
WEP	Wired Equivalent Privacy
WECA	Wireless Ethernet Compatibility Alliance
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access

CHAPTER 1

INTRODUCTION

IEEE (Institute of Electrical and Electronics Engineers) released 802.11 specifications [1], defining WLAN (Wireless Local Area Network) Medium Access Control (MAC) and Physical (PHY) Layer, in 1997. This was followed by IEEE 802.11b specifications [2] in 1999, which brought data transfer rate improvement from 2 Mbps to 11 Mbps. This was the year when several companies started to come up with WLAN products, compliant with IEEE 802.11b specifications. For the past couple of years, with dropping prices and wide range of products, WLAN technology has started seeing main stream use in home and corporate network settings.

WLAN clients can operate in ad-hoc mode without requiring an Access Point (AP), a special hardware and software combination that provides wireless network access to clients. Ad-hoc mode is often used when an AP does not exist and clients would like to share data over the air. If one or more of participating clients in ad-hoc network have some form of Internet connection, this connection can be shared among others if IP (Internet Protocol) settings are configured properly in clients. However, this type of service providing is nothing but dependable so at least one AP is used to provide stable and dependable wireless network access, and this type of setting is called "Infrastructure Mode". There are numerous AP alternatives in market today if one needs to setup a wireless network. Of course, price and functionality varies greatly among APs.

AP is basically a special embedded computer system designed to do one thing; provide wireless network access. The must-have features of an AP include an integrated web server for configuration and management. SNMP (Simple Network Management Protocol) is not unheard of, neither Telnet nor SSH (Secure Shell) but an integrated web server is the *de facto* standard when it comes to configuring and managing network-enabled embedded devices. Regardless of the configuration method, a common thing for off-the-shelf APs is that they are closed-boxes, meaning that one cannot alter the way they work. The question arises; why would anyone need to change the way an AP works? The answer lies in the word "customization".

Features provided by an AP do not always fulfill all needs, and requirements may change quite often. For example, if there are handfuls of wireless clients, it is easy to figure out which MAC address corresponds to which wireless client. However, when number of wireless clients increase, you would like to enter this information into AP configuration to keep track of them. Suppose, you are using ACL (Access Control List) for restricting access to designated clients based on MAC address, and AP configuration allows only MAC addresses to be entered. So, there is no chance to enter a note like client OS (Operating System), anti-virus program installed, contact information and other related information into the device configuration. Of course, new requirements might be more complex than this example like implementing 802.1x [3] client or implementing a detailed statistics gathering on wired and wireless interface of an AP. Most wireless device manufacturers release firmware upgrades in binary form but they are usually for bug-fix purposes so chances of finding all issues being addressed in a new firmware, if it ever comes out, are very slim. Custom building a modular AP, in which different components of the system and their interactions are configurable, is the solution to changing dynamic requirements.

1.1 Objective

Objective of the thesis is to build a modular, configurable, software-based AP using hardware and software components available in market. For the sake of completeness, and to prove that it is all possible, the open source software project

called AWLP (Alptekin's Wireless Linux Project) has been developed. AWLP [4] turns a PC (Personal Computer) with appropriate WLAN card into a full-featured, web-managed wireless AP with gateway/router functions. *http://awlp.sourceforge.net* is the URL (Uniform Resource Locator) that hosts AWLP software along with installation instructions and documentation. Version 1.0 of AWLP was released on December 25, 2004 under the GNU GPL (General Public License) version 2 [5].

In this thesis, detailed information about the design, implementation and inner workings of AWLP are provided to support the following claims:

- Custom building an AP is the solution to changing dynamic requirements of wireless networks;
- Building a custom AP using open source software is possible and practical.

1.2 Contributions

A wireless AP built with AWLP can be put to many uses just like an off-the-shelf wireless access product. AWLP has a distinct advantage over off-the-shelf wireless access products:

- It can be used to build wireless development, test and verification suites, and platforms for researchers. Performance testing and analysis of an off-the-shelf wireless device is limited to the capabilities and features of device itself. By utilizing AWLP, however, researchers can create custom-tailored development, test, and verification platforms that better suit their needs.

1.3 Structure

The remainder of this thesis will describe AWLP in details. Chapter 2 reviews related work about building a custom wireless access device. Chapter 3 has extensive coverage on software design, installation and implementation of AWLP. Usage details of AWLP along with a real implementation example are provided in Chapter 4.

Conclusions regarding advantages and limitations of building a custom wireless access device using open source software are presented in Chapter 5. The last chapter, Chapter 6, discusses possible further research in the subject. The typographic conventions used in this thesis are:

- *Italic*

  Italic is used for filenames, program and command names, directory names, and URLs except when they are part of headings or subheadings.

- `Constant width`

  `Courier New (10 pt)` is used for showing the output of commands.

CHAPTER 2

RELATED WORK

To increase possibility and range of implementations, efforts to build custom AP must consider storage size factor. So, choosing the right OS to build a custom AP on top is the most important decision that will ultimately influence the rest of the design. Linux and BSD (Berkeley Software Distribution) variants are the OS of choice on building custom AP because of two reasons; they are free of charge, and it is possible to strip-down unneeded programs and parts from OS as needed.

In order for a WLAN card to serve AP management functions, the wireless card must support a special operation mode called "BSS (Basic Service Set) Master" - a feature supported by only certain cards that have certain chipsets. List of drivers that work with WLAN cards supporting "BSS Master Mode", corresponding chipsets and URLs for these drivers are given below:

- HermesAP Chipset: Hermes
  http://www.hunz.org/hermesap.html
- HostAP Chipset: Prism 2/2.5/3
  http://hostap.epitest.fi
- MADWiFi Chipset: Atheros
  http://madwifi.sourceforge.net

- Prism54 Chipset: Prism Frisbee/GT/Duette/Indigo
  http://www.prism54.org

- wi(4) Chipset: WaveLAN/IEEE and Prism 2
  http://www.daemon-systems.org/man/wi.4.html

HostAP driver can take care of AP management functions in the host computer. This means that it will not require any special firmware for the wireless card. Special firmware complicates matters because of possible licensing issues involved and technical difficulties and risks associated with uploading them. HostAP driver is released under GNU GPL version 2. HostAP stands from the rest due to its well-maintained code, documentation and support. However, it has a major limitation; it can only support IEEE 802.11b compliant wireless cards, IEEE 802.11a [6] and IEEE 802.11g [7] are not supported. AWLP uses HostAP driver so it can support IEEE 802.11b only.

When design goals and features of AWLP are considered, there are two similar software; Pebble [8] distribution and m0n0wall [9] firewall package. Details about the two are given below along with a brief comparison with AWLP. Of course, these are not the only open source wireless AP projects but we will limit our discussion to these two because they have a certain level of acceptance. There are even open source AP projects that derived from Pebble and m0n0wall. For example, the project called pfSense, hosted at *http://www.pfsense.org*, is derived from m0n0wall. Pebble-Voyage project, hosted at *http://www.voyage.hk*, is derived from Pebble as the name implies.

2.1 Pebble

Pebble is a small read-only distribution based on GNU/Linux Debian [10] that turns a host machine into wireless AP. Pebble has been developed and maintained by Terry Schmidt, and has been hosted by NYC (New York City) Wireless, an advocacy group for wireless community networking, at *http://www.nycwireless.net/pebble*. Pebble distribution includes HostAP, Prims54 and MADWiFi drivers, thus it can support not only IEEE 802.11b compliant wireless cards but also newer IEEE 802.11g

compliant cards with certain chipsets. Pebble has its disadvantages and limitations as well:

- It is a read-only distribution, thus it requires another host with storage space to transfer and store any log file if needed.
- It does not come with a web server. Thus, it does not have a built-in web-based management. Configurations are to be done using SSH connection.
- As pointed out by Terry Schmidt in *README* file of Pebble distribution, it does not have a good documentation, and there is no future plan for it, neither.

2.2 m0n0wall

m0n0wall is an embedded firewall software package developed by Manuel Kasper based on FreeBSD [11]. It is an actively maintained and well-documented software package, hosted at *http://www.m0n0.ch/wall*. m0n0wall has concentrated on firewall features but added support later for wireless interface and wireless management functions. It uses wi(4) driver that comes with FreeBSD. Like HostAP, wi(4) driver works with IEEE 802.11b complaint cards only, thus cards supporting IEEE 802.11a or IEEE 802.11g cannot be used. Unlike Pebble, m0n0wall has web-based configuration interface. m0n0wall has been designed as a viable alternative to commercial firewall products, thus it contains various firewall related features like NAT (Network Address Translation), DNS (Domain Name System), DHCP (Dynamic Host Configuration Protocol), etc. Most of these features are also needed by an AP, thus it has a clear advantage in this perspective. AWLP has advantages over m0n0wall by providing configuration option for wider set of wireless parameters. AWLP has also advantages in statistic gathering, display and status display. However, I must accept that m0n0wall is more widely known and accepted than AWLP in many respects, this is especially true for features related to firewall functionality.

CHAPTER 3

AWLP: Alptekin's Wireless Linux Project

The idea behind AWLP was to develop an open source software that would turn a PC with appropriate WLAN card into a modular, customizable, web-managed AP, whose features are comparable to those of its commercial counterparts. Since almost all commercial alternatives are closed-boxes with proprietary software that do not allow any modifications, customizability and modularity are the major strength of AWLP. Installing AWLP on target platform requires considerable effort. Thus, documentation is provided. The documentation is crucial not just during the installation phase but also during operation and customization.

AWLP files and documentation are hosted at http*://awlp.sourceforge.net* by SourceForge.net [12], which provides free hosting services for open source code and applications. The initial release of AWLP - version 1.0 - was made public on December 25, 2004. Considerable amount of effort was put during design and test phases to make it a stable release. The project's website contains all necessary information and instructions to deploy an AP using AWLP. Documentation in the website focuses on getting AWLP to work. This chapter and this thesis in general, however, focus on explaining purpose and operation of each AWLP component.

3.1 Features of AWLP

- Written in Perl
- Option to trim-down non-critical OS parts for reduced storage
- Password protected web-based management
- 40-bit and 104-bit WEP (Wired Equivalent Privacy) Keys
- Firewall with editable outgoing port list
- Caching DNS server
- DHCP server
- IP masquerading with NAT
- NTP (Network Time Protocol) server for time synchronization
- Access Control List for authentication control
- Client SNR (Signal-to-Noise Ratio) and bandwidth usage radars
- Detailed client statistics and client management features

3.2 Screen Shots

Screen shots shown below were taken from AWLP sample setup (Refer to Figure 8). As can be seen from left part of these screen shots, there are 7 (seven) different menus. These menus give name to the following figures:

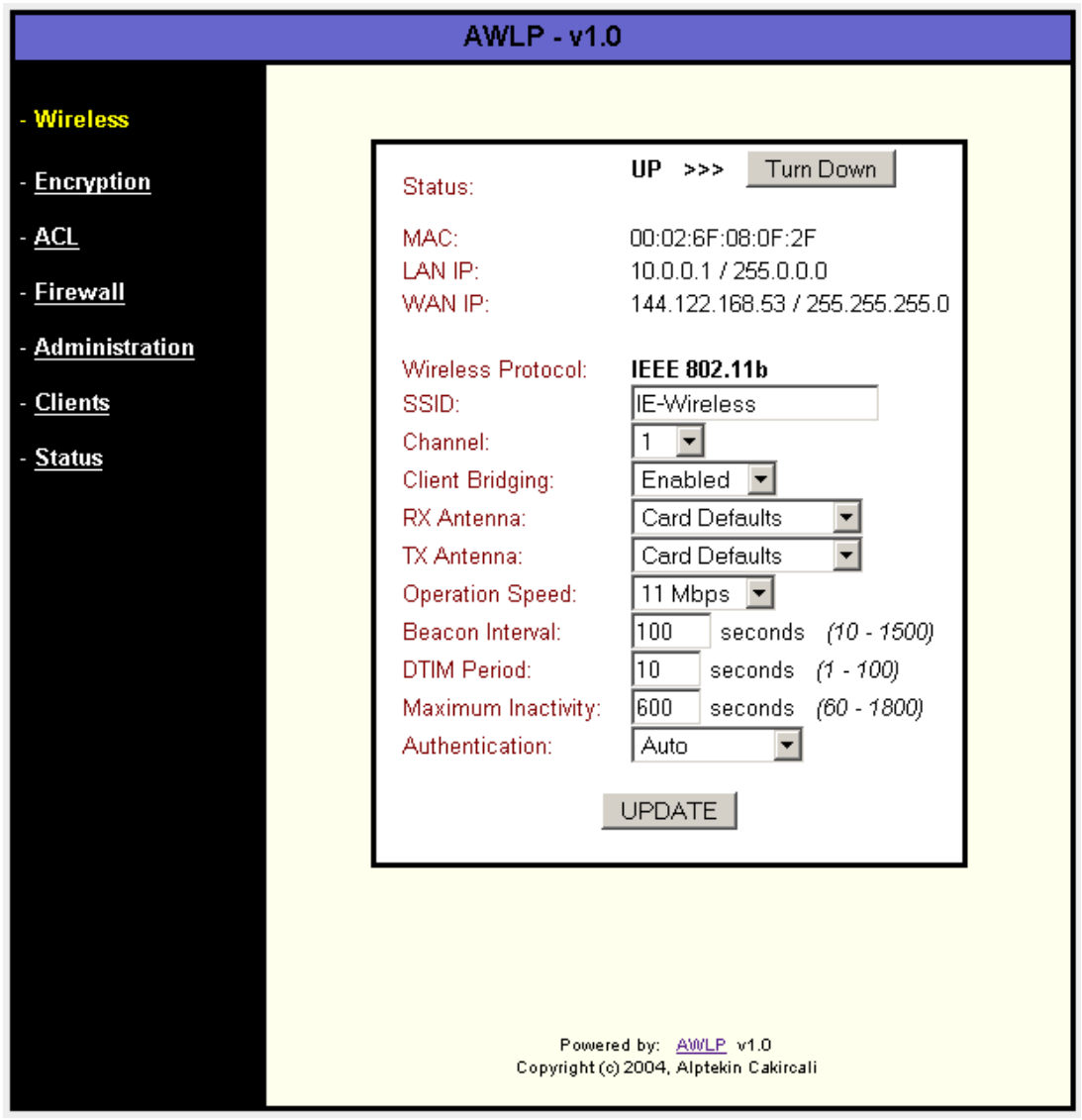

Figure 1 – AWLP, Wireless Menu

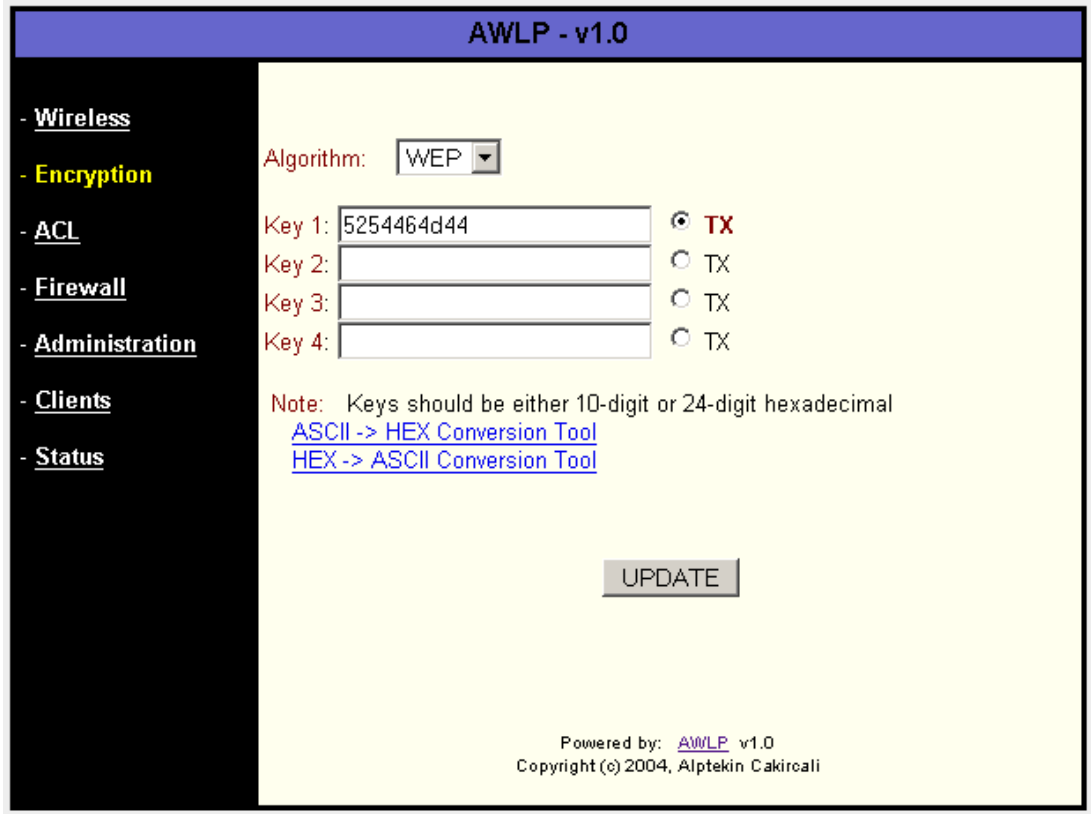

Figure 2 – AWLP, Encryption Menu

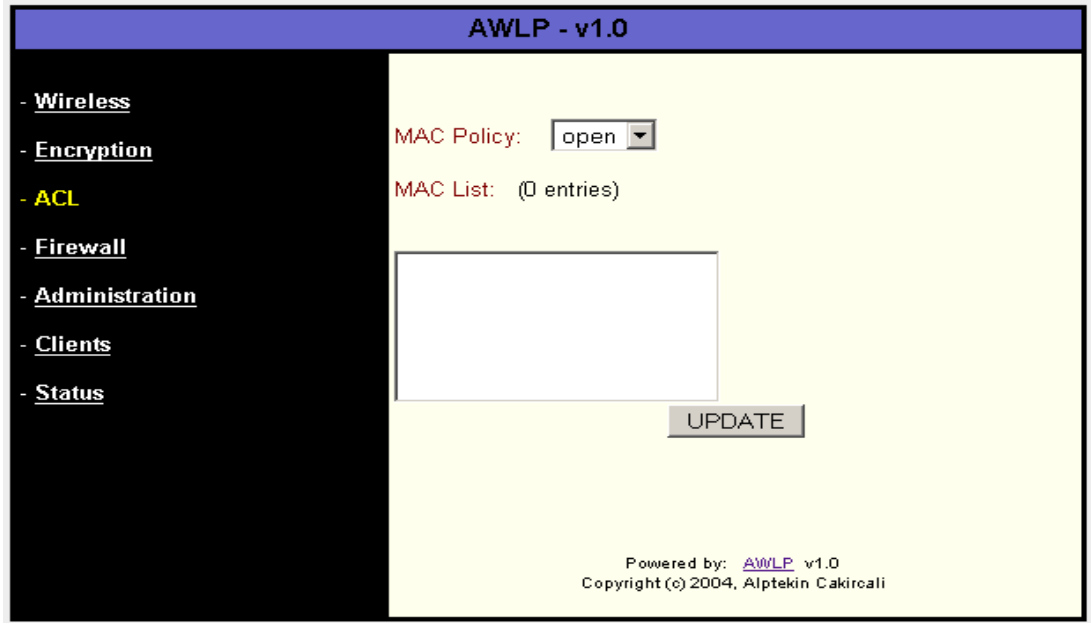

Figure 3 – AWLP, ACL Menu

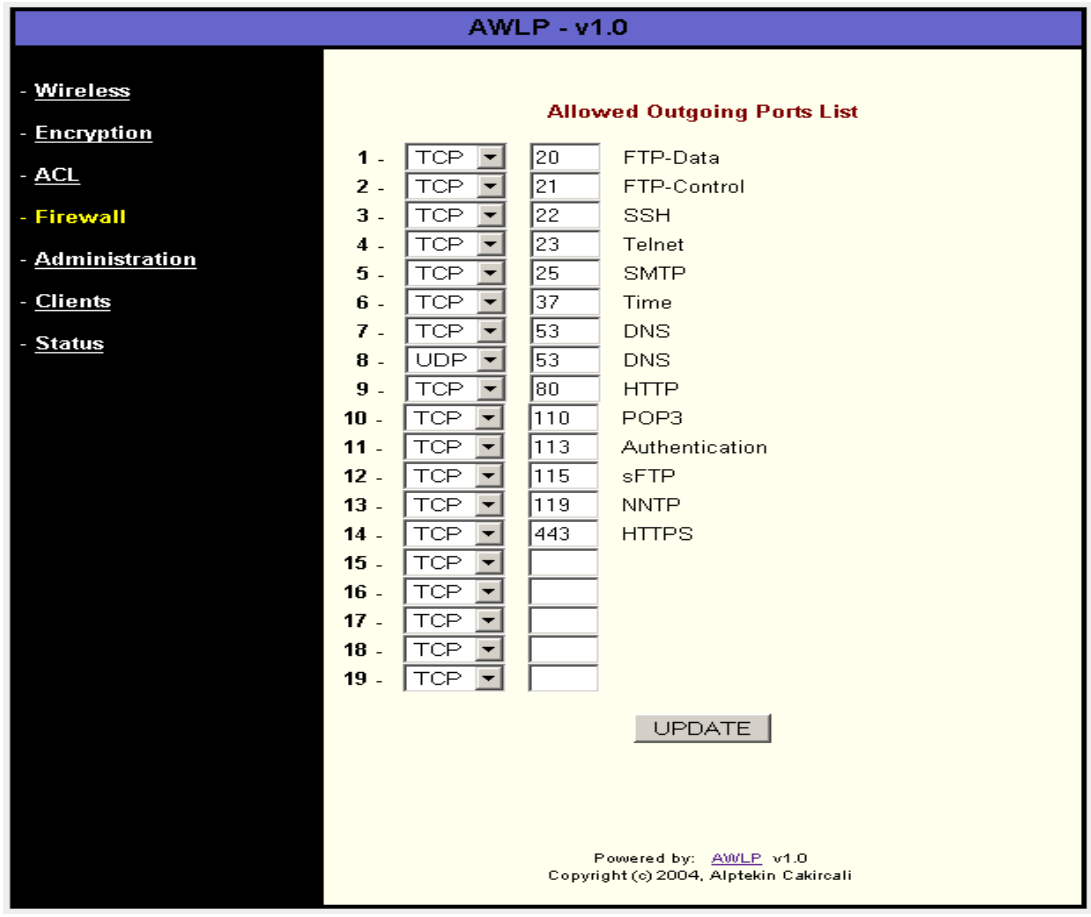

Figure 4 – AWLP, Firewall Menu

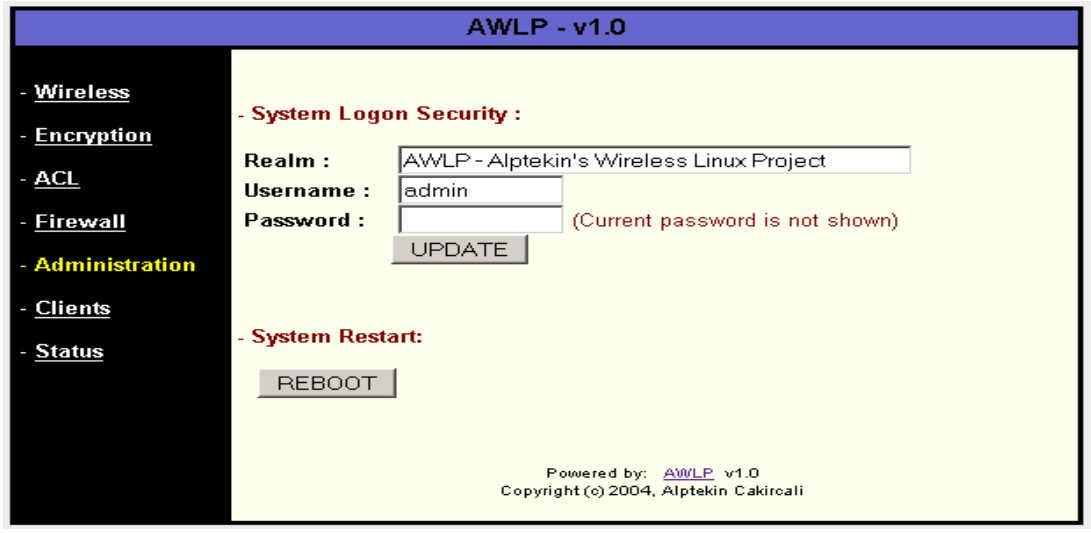

Figure 5 – AWLP, Administration Menu

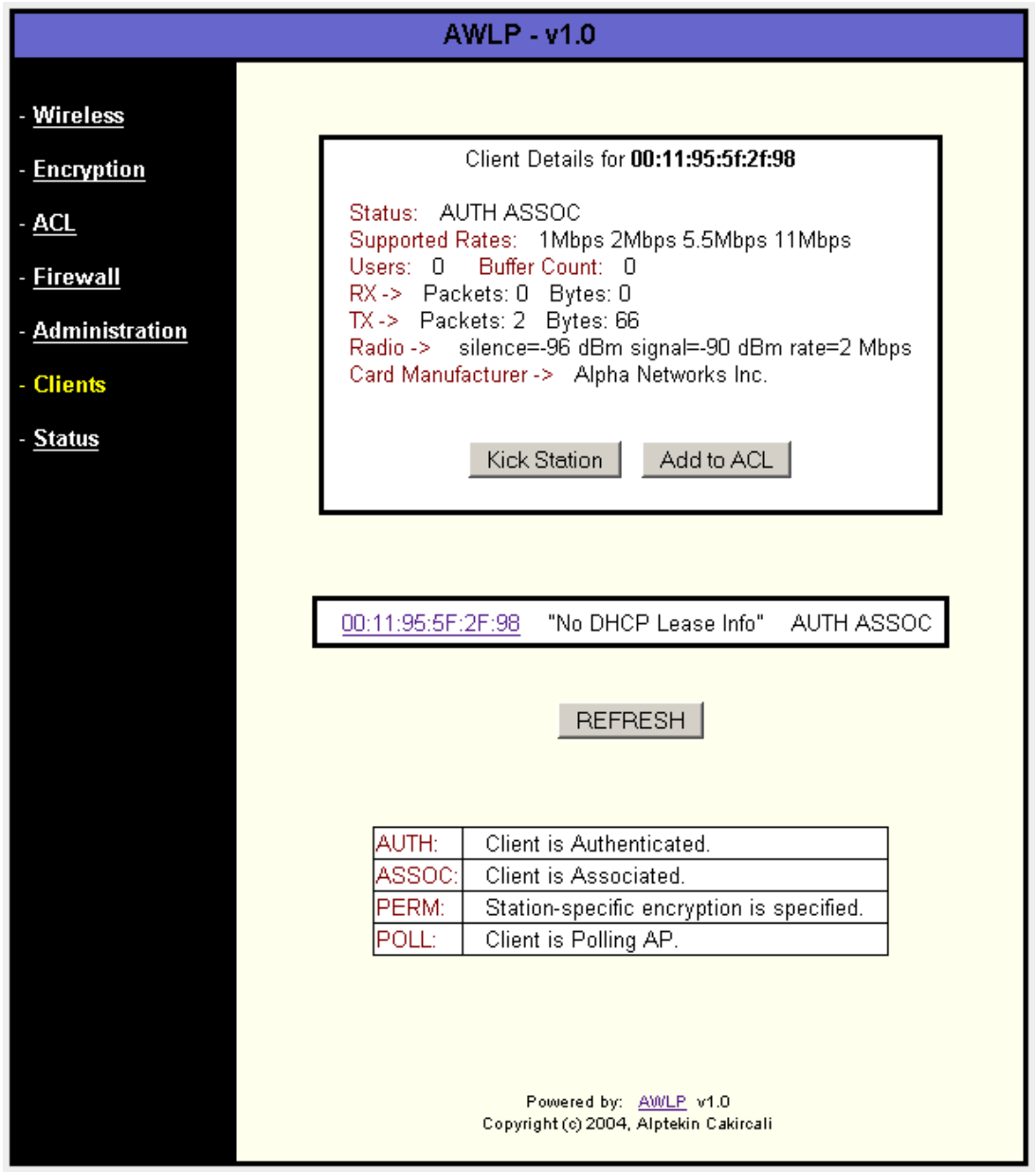

Figure 6 – AWLP, Clients Menu

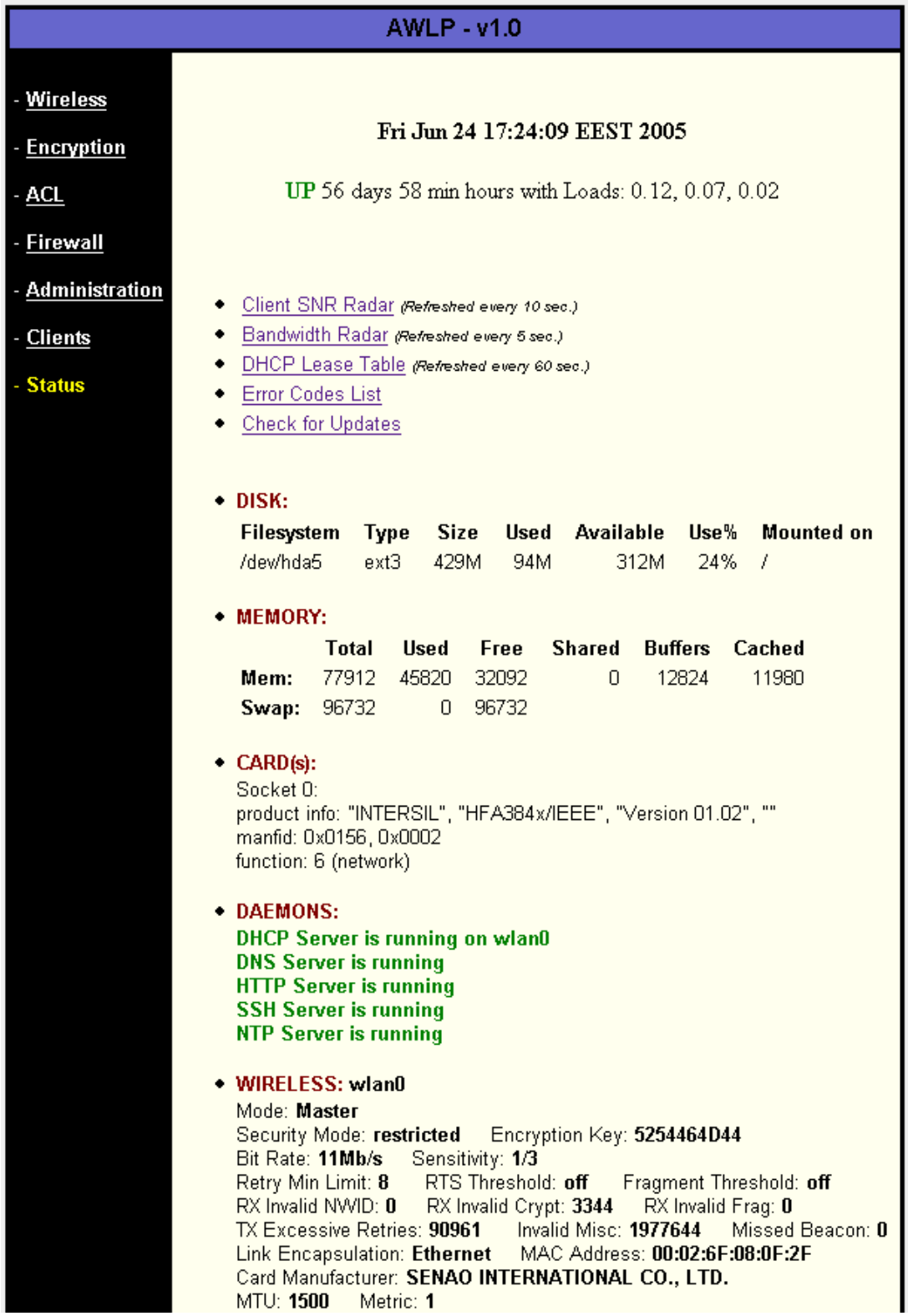

Figure 7 – AWLP, Status Menu

3.3 Operating System

The choice of OS for AWLP is GNU/Linux Slackware [13], which is known for its stability. Version 1.0 of AWLP is designed to be installed on version 10.0 of Slackware. Slackware has hundreds of packages that contain various binaries, configuration files, documentation files, etc. Each package does one or more thing, and contains one or more files. For example, if "cdrtools" package is chosen, necessary binaries and configuration files are installed to use CD (Compact Disc) writer device on the host system. While some of these packages are required, some of them are recommended, and others are optional. By choosing right packages during installation phase, you will have an OS that does what you need without any extra features and tools that might have consumed unnecessary space and memory otherwise. The packages are divided into software series for categorization. Appendix A.1, retrieved from *http://www.slackware.org*, contains a list of these series with their descriptions. AWLP has a directory named *tagfiles*, which contains text files that specify which package to install and which package to skip for each of the series. Appendix A.2 has a list of Slackware packages that will be installed if *tagfiles* directory of AWLP is used during Slackware installation.

3.4 AWLP Content

AWLP is written in Perl, a widely accepted, cross-platform, open source interpreted programming lanugage. The installation script of AWLP, *installer.sh,* is written in Bash (Bourne-Again Shell). AWLP software also contains various configuration files in ASCII (American Standard for Character Information Interchange) text.

Compressed tarball of AWLP version 1.0 can be found at "Download" section of AWLP's website. There are five subdirectories inside the archive, each containing various files. List of these files and their relative paths can be found in Appendix B. Descriptions of these files are given in subsequent sections.

3.4.1 installer.sh, COPYING, and README

installer.sh, *COPYING*, and *README* files reside in the root directory of AWLP.

3.4.1.1 ./installer.sh

This Bash script is responsible for setting up and configuring AWLP. Understanding how AWLP works is possible by understanding *installer.sh* first, so a detailed description of *installer.sh* is provided in Section 3.4.8.

3.4.1.2 ./COPYING

AWLP is released under GNU GPL version2. This file contains the related license information.

3.4.1.3 ./README

In addition to author and license information, *README* file has system requirements, hardware compatibility and installation sections. It should be read prior to AWLP installation.

3.4.2 Configuration Files

Files under *configfiles* directory are configuration files for various services such as DNS, HTTP (Hyper Text Transfer Protocol), DHCP, etc.

3.4.2.1 ./configfiles/dhcpd.conf

AWLP starts DHCP service on host machine to give IP leases to wireless clients. *dhcpd.conf* is the configuration file that gives directives to DHCP service. Default lease time is set to 1 day (86400 seconds). IP lease range is between 10.0.0.10 and 10.0.0.254. Router, DNS and NTP server addresses are set to 10.0.0.1. IP lease

range can be altered without any problem as long as it is in the same private "A" class. However, changing gateway, DNS and NTP server addresses will require various changes on several other configuration files.

3.4.2.2 ./configfiles/httpd.conf

AWLP has web-based management. Apache Web Server [15] is HTTP daemon being used by AWLP. *httpd.conf* is the configuration file containing various directives to customize the behavior of Apache Web Server.

3.4.2.3 ./configfiles/named.conf

AWLP starts DNS service on host system to answer domain name queries from connected clients. AWLP uses BIND (Berkeley Internet Name Daemon) [15] as the DNS daemon. *named.conf* is the configuration file for BIND. AWLP needs a caching-only DNS, so *named.conf* has directives accordingly. This caching-only DNS server is made to answer queries from IP addresses within private "A" block only. Thus, queries from outside of this block are discarded.

3.4.2.4 ./configfiles/named.ca

This is a required file for the operation of DNS server. *named.ca* contains IP addresses of 13 global root name servers.

3.4.2.5 ./configfiles/named.local

This file contains PTR (Pointer) records for reverse lookup operation of DNS. Since this is a caching-only DNS, it contains only one PTR record as default, which is "localhost".

3.4.2.6 ./configfiles/localhost.zone

localhost.zone contains records for forward lookup DNS operation. However, since this is only caching-only DNS, it does not contain any records. When clients make queries, DNS server resolves IP addresses of FQDNs (Fully Qualified Domain Names) with the help of root name servers (Refer to Section 3.4.2.4).

3.4.2.7 ./configfiles/ntp.conf

AWLP starts NTP service to synchronize time with global time servers and to provide time synchronization service to its connected clients. *ntp.conf* is the configuration file for NTP service. The global time servers contained in *ntp.conf* file are:

- ntp-1.cso.uiuc.edu
- a.ntp.alphazed.net
- ntp0.cornell.edu
- ntp3.theinternetone.net

3.4.2.8 ./configfiles/rc.ntpd

rc.ntpd is the file responsible for stopping, starting, and restarting NTP service with appropriate control switches and options.

3.4.2.9 ./configfiles/oui_filtered.txt

IEEE controls MAC addresses of physical devices in organizational level. IEEE has a file called *oui.txt* [16], which contains first 24-bit portion of MAC address and the corresponding information such as name, address, phone for each organization. By looking at this file and first 24-bit of MAC address of device in question, it is possible to tell the manufacturer of a given network card. *oui_filtered.txt* is the trimmed-down version of *oui.txt* for AWLP. Parts that get trimmed include

additional information about organizations like address, phone, etc. AWLP searches within *oui_filtered.txt* file to find names of manufacturers for clients in question.

3.4.2.10 ./configfiles/rc.dhcpd

Slackware has various startup files under */etc/rc.d* directory. These startup files, if their permissions are set properly, start various services like DNS, HTTP, etc. However, not all services have startup files by default. For these services, appropriate startup files are created that are responsible for starting, stopping and restarting services. DHCP service is one of them. For AWLP to start DHCP service on host machine, *rc.dhcpd* is copied to */etc/rc.d* directory by *installer.sh* during installation.

3.4.2.11 ./configfiles/rc.firewall

Firewall that comes with AWLP is incoming-blocked and outgoing-configurable. *iptables* is the name of command for firewall software in Linux variants. *rc.firewall* is the configuration file containing directives to *iptables* firewall program. When list of allowed outgoing ports is updated through AWLP's web-based management interface, corresponding section in *rc.firewall* also gets updated in order to preserve changes through system restarts.

3.4.2.12 ./configfiles/rc.local

rc.local file usually comes empty with default Linux installations. It allows custom commands to be executed and custom services to be started at startups. *rc.local* file in AWLP does the following things in order:

- Starting cardmgr daemon if it is not already started
- Starting SSH server if it is not already started
- Pre-configuring WLAN interface
- Starting NTP daemon if it is not already started

- Starting DHCP server if it is not already started
- Starting DNS server if it is not started, and restarting it if already started
- Loading and activating firewall rules
- Configuring WLAN interface

3.4.2.13 ./configfiles/rc.wlan0

rc.wlan0 contains lines of commands to bring up the wireless interface. "AWLP, Wireless Menu" (refer to Figure 1) makes it possible to change SSID (Service Set Identifier), channel number, WEP keys, etc. These changes are stored in *rc.wlan0* so that they are preserved between system startups.

3.4.2.14 ./configfiles/rc.wlan0-pre

IP configuration of wireless interface needs to be set for other network services to function properly. However, clients should be blocked until all related services such as DHCP, DNS get started. This is achieved by blocking client authentication first, then configuring IP setting of wireless interface in *rc.wlan0-pre*. *rc.local* (refer to Section 3.4.2.12) calls *rc.wlan0-pre* as the third step, and it calls *rc.wlan0* as the very last step to start allowing wireless clients.

3.4.2.15 ./configfiles/.htaccess

AWLP's web-based management interface is password-protected to prevent unauthorized use. When an attempt is made to use this interface, a simple dialog box with realm of authentication shows up, asking for username and password. *.htaccess* file contains this realm along with directives to look for *.htpasswd* file, which contains valid username and password pairs for logon. HTTP daemon basically checks *.htaccess* file under requested directory for authentication directives each time a password-protected page is requested. AWLP does not contain *.htpasswd* file but this file is created on-the-fly by *installer.sh*.

3.4.3 Core Scripts

corescripts directory contains file and scripts that are essential to operation of AWLP.

3.4.3.1 ./corescripts/engines1.pl

This include file contains subroutines essential to operation of AWLP. The list of subroutines is given below. More information about these subroutines along with their input and output specifications are provided in Appendix D.

- proc_get_client_mac_list
- proc_get_client_mac_details
- proc_get_prism2_parameters
- set_prism2_parameter
- proc_get_AP_stats
- proc_get_AP_debug
- proc_get_debug
- add_mac_to_acl
- remove_mac_from_acl
- kick_station_from_AP
- ACL_status
- is_MAC_in_ACL
- ACL_policy_modify
- WDS_add_MAC
- WDS_remove_MAC
- Card_Identification

- Card_Port_reset
- Encryption_control
- check_daemon_status
- Logon_Security_control
- get_uptime_info
- get_card_manufacturer_info
- get_memory_usage_info
- get_disk_usage_info
- get_dhcpd_lease_info
- iwconfig_parser
- ifconfig_parser
- AP_Wireless_manage
- AP_LAN_manage
- System_reboot

3.4.3.2 ./corescripts/engines2.pl

This include file contains eight subroutines that are auxiliary in nature; they are not as critically important to operation of AWLP as those of *engines1.pl*. Below is the list of subroutines contained in this file.

- sort_ClientFlags_byvalue
- sort_OperationRates_bykey
- sort_OpenPorts_byportnumber
- find_ISM_channel_number
- convert_dec_to_binary

- convert_binary_to_dec
- find_broadcast_address
- show_date_time

3.4.3.3 ./corescripts/error_messages.pl

This include file contains error messages. Error codes are represented as array indices, error messages and their descriptions are represented as array members.

3.4.3.4 ./corescripts/extras.pl

Any functionality not covered by *index.pl* and *radar.pl* are handled by *extras.pl*. Conversion from ASCII to hexadecimal, hexadecimal to ASCII, showing DHCP lease table, checking AWLP updates, listing error messages are covered by *extras.pl.* It has "Action1" form variable that determines action to be taken. Values of "Action1" are listed below with description of things accomplished within each part:

- "**ASCIItoHEX**" - ASCII to Hexadecimal Conversion Tool
- "**HEXtoASCII**" - Hexadecimal to ASCII Conversion Tool
- "**DHCPLeaseTable**" - DHCP Lease Table
  This part is responsible for showing DHCP lease details. The details include lease duration, IP number, MAC address, and host name if specified by client. Subroutine calls made within this part is listed below:
  - get_dhcpd_lease_info
  - proc_get_client_mac_details
- "**CheckUpdates**" - Check for Updates
  In this part, system call is made to check the URL defined by configuration variable CHECK_FOR_UPDATES_URL (refer to Appendix C). This system call is:

- system("wget -q -t 1 -T 15 -O ${CHECK_FOR_UPDATES_PATH} ${CHECK_FOR_UPDATES_URL}")

- "**ErrorCodesList**" - Error Codes List
  Throughout AWLP management interface, error messages are shown with error codes. This part of *extras.pl* lists all error messages with their codes and descriptions.

3.4.3.5 ./corescripts/global_configuration.pl

This is just an include file containing various configuration variables. Understanding these configuration variables is essential to understanding how AWLP operates, thus description of variables along with their default values are given in Appendix C.

3.4.3.6 ./corescripts/index.html

This is the welcoming HTML (Hyper Text Markup Language) page for AWLP management interface. It has static content giving information about AWLP such as the copyright and author information.

3.4.3.7 ./corescripts/index.pl

This Perl script is the one that is being called when an "AWLP Management" link is clicked inside the welcoming page. It is the most important script of AWLP with over 1500 lines of code. It calls various subroutines that are defined in *engine1.pl* and *engine2.pl*. Refer to Section 3.4.7 for a detailed description of things accomplished within *index.pl*.

3.4.3.8 ./corescripts/radar.pl

In order to keep the size of *index.pl* reasonable, *radar.pl* is used. It is responsible for drawing SNR and bandwidth usage graphics. The links to *radar.pl*

were given inside “Status Menu” (refer to Figure 7) of AWLP management interface. It has a form variable called “Action1” that controls action to be taken by the script. “Action1” values are:

- “**SNRRadar**” - Client SNR Radar
  This part is responsible for drawing SNR graph of connected clients. SNR (Signal-to-Noise Ratio) tells us the quality of wireless link between host machine and client. SNR_RADAR_REFRESH_INTERVAL (refer to Appendix C) configuration variable in *global_configuration.pl* determines how many seconds will elapse before SNR graph gets refreshed. The default is 10 seconds. Subroutine calls made within this section are:
  - proc_get_client_mac_list
  - proc_get_client_mac_details
- “**BWRadar**” - Bandwidth Radar
  Monitoring bandwidth usage is very important to understand usage patterns and bandwidth requirements. This part is responsible for drawing bandwidth usage graph of connected clients. BW_RADAR_REFRESH_INTERVAL configuration variable defines the refresh period for the bandwidth usage graph. The default value is 5 seconds. Subroutine calls made within this section are:
  - proc_get_client_mac_list
  - proc_get_client_mac_details

3.4.4 Images

Images directory is here for organizing all image files. However, as of version 1.0 of AWLP, there exists only one image file.

3.4.4.1 ./images/LinuxPowered.gif

The welcoming page, *index.html* (refer to Section 3.4.3.6), contains a Linux logo. *LinuxPowered.gif* is this image file.

3.4.5 Tagfiles

tagfiles directory contains tag files for each Slackware software series (refer to Appendix A.1) defining which packages to install and which packages to skip. *tagfiles* directory and its content are copied to a floppy disk manually before starting installation, and is later used during Slackware setup.

3.4.6 Tarballs

During AWLP installation, hostap-driver and hostap-utils tarball packages, which are parts of HostAP, are extracted, compiled, and installed.

3.4.6.1 ./tarballs/hostap-driver-0.2.5.tar.gz

It is version 0.2.5 of hostap-driver; the most important part of HostAP software. It is the driver that makes WLAN card to work as AP.

3.4.6.2 ./tarballs/hostap-utils-0.2.4.tar.gz

It is version 0.2.4 of hostap-utils, which is part of HostAP software providing various utilities to manage wireless interface.

3.4.6.3 ./tarballs/bridge-utils-1.0.4.tar.gz

It is version 1.0.4 of bridge-utils software, which makes bridging between two or more interfaces in same host possible without requiring a special hardware. AWLP is designed to work as a wireless access gateway. One might need to make modifications to turn it into AP only device without any gateway functions. bridge-

utils software [17] will be needed in such a scenario, thus it gets also installed, although functionality it provides are not utilized by default.

3.4.7 index.pl

index.pl with its calls to subroutines defined in *engines1.pl* and *engines2.pl* is the core of AWLP management interface. To control its actions, *index.pl* is called with "Action1", "Action2" and "Action3" form variables. There is different "Action1" value for each menu in the management interface of AWLP (refer to Section 3.2). These different values are listed below with their corresponding menu names and their functionality:

- "**Wireless**" - Wireless
  This part is responsible for turning up and down wireless interface, listing IP configuration for wireless and wired interfaces, listing and changing wireless parameters such as SSID, channel number, operation speed, etc. Following subroutines are called within this part:
  - AP_Wireless_manage
  - set_prism2_parameter
  - ifconfig_parser
  - iwconfig_parser
  - find_ISM_channel_number
  - proc_get_prism2_parameters
- "**Firewall**" – Firewall
  Function of this part is to modify allowed outgoing port list. It has a simple protocol type selection box, port number text box, and submit button. System call made within this part is:
  - system("/etc/rc.d/rc.firewall")

- "**Administration**" - Administration

  This part has two main functions; changing system logon security and restarting the system. Managing AWLP through its interface requires a basic authentication to prevent unauthorized access. In this section, changing authentication realm, username and password are made possible. Two subroutines are called within this part.

  - Logon_Security_control
  - System_reboot

- "**ShowClients**" - Clients

  Several subroutine calls are made within this part, which are listed below, in order to provide information about clients connected. This information includes MAC addresses of clients, details about IP leases and statistics about the connection. Option to kick a station from AP and adding/removing MAC address from ACL are also provided by this part.

  - kick_station_from_AP
  - remove_mac_from_acl
  - add_mac_to_acl
  - proc_get_client_mac_list
  - proc_get_client_mac_details
  - get_dhcpd_lease_info
  - get_card_manufacturer_info

- "**ShowACL**" – Access Control List

  This part makes changing MAC policy and updating ACL possible. MAC policy can be either "open", "deny" or "allow", whereas ACL contains a list of MAC addresses. MAC policy is enforced on MAC addresses listed in ACL. Subroutine calls made within this section are listed below:

  - ACL_policy_modify

- add_mac_to_acl
- kick_station_from_AP
- proc_get_client_mac_list
- ACL_status

- "**ShowEncryption**" - Encryption

  This part allows encryption algorithm and WEP keys to be changed. Encryption algorithm can be either "WEP" or "none". WEP keys can be either 10-digit hexadecimal or 24-digit hexadecimal. With 24-bit IV (Initialization Vector), 10-digit hexadecimal corresponds to 64-bit, and 24-digit hexadecimal does correspond to 128-bit WEP key. List of subroutine calls made within this part is given below:

  - Encryption_control
  - proc_get_client_mac_list
  - kick_station_from_AP

- "**ShowStatus**" - Status

  This part is responsible for outputting status information. It includes current date and time, disk usage and memory usage information, wired and wireless interface information and statistics, and network daemons status. Subroutines calls within this part are given below:

  - show_date_time
  - get_uptime_info
  - get_disk_usage_info
  - get_memory_usage_info
  - iwconfig_parser
  - ifconfig_parser
  - Card_Identification

- get_card_manufacturer_info
- check_daemon_status

3.4.8 installer.sh

installer.sh script has been divided into commented sections, each doing different things to accomplish the final goal, setting up AWLP on host system. These sections in execution order are listed below:

- "Defining project name, project relative path, project version, default authentication username and password, hostap-driver version, hostap-utils version, bridge-utils version": These definitions will be later used in the script while compiling programs, and while copying necessary scripts and configuration files.
- "Defining required Slackware pages": Existence of these packages will be later checked by *installer.sh* based on this definition.
- "Defining required program paths": These programs are required to copy files, compile source files and configure network settings. Existence of these programs will be later checked by *installer.sh*. If the required Slackware packages are installed, then these programs will be in place, therefore this section provides an extra control mechanism.
- "Defining packages to be removed": In order to compile auxiliary programs (hostap-driver, hostap-utils, bridge-utils), some Slackware packages (kernel-source, automake, etc.) are required. However, after successfully compiling these auxiliary programs, these packages are no longer needed to run AWLP. They will be required, however, if you need to compile another source file later on. If you have enough disk space, that is around 1GB or more, then there is no need to remove these packages. Later in *installer.sh*, you will be given option to remove these defined packages from host machine.

- "Outputting project name and version along with author and copyright information": In this phase of execution, *installer.sh* displays the project name, version, author and copyright information to visually convey the message that the installation has just been started.
- "Marking the time for elapsed installation time": Time has been marked to calculate elapsed time for the installation.
- "Fetching installation directory": The directory where *installer.sh* script is run gets saved into a scalar variable. The path of this directory will be later used by *installer.sh*.
- "Checking the user running *installer.sh*": The installation script, *installer.sh*, is designed to do several things like compilation, package removals, permission and owner settings of various files, and other tasks that require administrative privileges. In order for *installer.sh* to accomplish these tasks, user running *installer.sh* must be the user "root". If not, *insaller.sh* aborts execution with an error message.
- "Checking */var/log/packages*": Information about the Slackware packages installed are stored under */var/log/packages* directory. In order to check the existence of these packages, it is imperative that */var/log/packages* directory exists with read and execute permissions for the user "root". If there is a problem, execution gets aborted.
- "Checking if the required Slackware packages are installed": In this section, the packages are checked to make sure they are installed. If there are any missing packages, execution is aborted with an error message containing the list of missing packages.
- "Checking the required program paths": Required programs that were defined previously are checked in this section to detect any missing one. If one or more paths are found missing, execution is aborted with an error message containing the list of missing programs paths.

- "Gathering installation options": The installation script, *installer.sh*, goes to an interactive mode in this section to gather installation options. There are three installation options:
    - Removing non-critical packages to save space
    - Trimming down documents and manuals to save space
    - Stripping down the Perl package to bare minimum to save space

  For each of these three questions, you can answer "Y" or "N". If you type anything other than "y" or "Y", it will be treated as "N". These installation options are here to save disk space. Therefore, if you have around 1 GB of or more disk space, you are advised to answer "N" to all the questions.
- "Confirming installation options": Selections to the three aforementioned installation options are confirmed in this section to prevent any mistakes. If answered "Y" to the confirmation question, the actual compilation, copying, configuring phases will get started.
- "Installing hostap-driver, hostap-utils, bridge-utils": hostap-driver and hostap-utils are crucial to the operation of AWLP but bridge-utils is installed just in case bridging function rather than routing function is needed between wireless and wired interfaces.
- "Adding */bin/false* and */dev/null* to */etc/shells*": For the security of the system, it is required that home directory and default shell of some users are set to */dev/null* and */bin/false* respectively. In order to do this, */bin/false* and */dev/null* shall be listed in */etc/shells* file.
- "Creating the user and group apache": For AWLP's web-based management interface, HTTP daemon is needed. For AWLP, it is Apache Web Server. The user and group "apache" is created in this section for this daemon to be running as.
- "Copying and setting permissions for configuration files": The configuration files (refer to Section 3.4.2) are needed to be copied over. They require

changes in ownership and permission for HTTP daemon to modify the content of these files. These configuration files are listed below along with permissions and owner information in Unix/Linux style.

```
-rwxrwx---  root apache /etc/dhcpd.conf
-rwxrwx---  root apache /etc/apache/httpd.conf
-rw-r--r--  root root
            /var/named/caching-example/localhost.zone
-rw-r--r--  root root   /var/named/caching-example/named.ca
-rw-r--r--  root root   /etc/named.conf
-rw-r--r--  root root   /var/named/caching-example/named.local
-rw-r--r--  root root   /etc/ntp.conf
-rwxrwx---  root apache /etc/awlp/oui_filtered.txt
-rwx--x---  root root   /etc/rc.d/rc.dhcpd
-rwxrwx---  root apache /etc/rc.d/rc.firewall
-rwxr-x---  root apache /etc/rc.d/rc.local
-rwxrwx---  root apache /etc/rc.d/rc.ntpd
-rwxrwx---  root apache /etc/rc.d/rc.wlan0
-rwxrwx---  root apache /etc/rc.d/rc.wlan0-pre
```

- "Copying *.htaccess*, and creating *.htpasswd* files": These two files along with a couple of directives in *httpd.conf* (refer to Section 3.4.2.2) are the core of authentication mechanism of AWLP's web-based management interface.
- "Setting up permissions on various commands": For AWLP's management interface, HTTP daemon needs certain permission on some commands. Some of the permission changes involve "suid" and "guid" bits to be set as well. Some of these commands are standard Linux commands, and some of them come with hostap-driver and hostap-utils. These commands along with their permissions and owner information in Unix/Linux style are listed below:

```
-rwsr-x---  root apache /usr/local/bin/prism2_param
-rwsr-x---  root apache /usr/local/bin/hostap_crypt_conf
-rwsr-x---  root apache /usr/sbin/dhcpd
-rwsr-x---  root apache /sbin/iwpriv
-rwsr-x---  root apache /sbin/iwconfig
-rwsr-x---  root apache /sbin/ifconfig
```

```
-rwsr-x---  root apache /bin/ps
-rwsr-x---  root apache /sbin/halt
-rwsr-x---  root apache /usr/bin/htpasswd
-rwsr-x---  root apache /sbin/cardctl
-rwsr-x---  root apache /usr/sbin/iptables
-rwsr-x---  root apache /sbin/modprobe
-rwx--x---  root root   /etc/rc.d/rc.bind
-rwx--x---  root root   /etc/rc.d/rc.dhcpd
-rwxr-xr-x  root root   /etc/rc.d/rc.hotplug
-rwx--x---  root root   /etc/rc.d/rc.httpd
-rwxr-xr-x  root root   /etc/rc.d/rc.inetd
-rwxr-xr-x  root root   /etc/rc.d/rc.pcmcia
-rwxr-xr-x  root root   /etc/rc.d/rc.sshd
-rwxr-xr-x  root root   /etc/rc.d/rc.wireless
```

- "Removing optional Slackware packages": While defining the installation options, one of the questions was about removing non-critical Slackware packages. If "Y" was chosen, the Slacware packages that were defined previously get removed in this section.
- "Trimming down documents and manuals": If "Y" was chosen for the second installation option, some documents and manual pages are removed to save disk space. The commands to remove these documents and manuals are given below:

```
rm -rf /usr/doc/*
rm -rf /usr/man/*
rm -rf /usr/share/locale/*
rm -rf /var/www/icons/*
rm -rf /var/www/htdocs/*
```

- "Stripping down the Perl package": If "Y" was given as the answer to the third installation question, the Perl package, which is around 35 MB, is stripped down to bare minimum (approx. 1 MB) to save disk space.

- "Copying core scripts and setting their permissions": These scripts are what really make AWLP. These scripts along with their permission and ownership information are listed below in Unix/Linux style:

```
-rwxr-xr-x  root apache /var/www/cgi-bin/awlp/engines1.pl
-rwxr-xr-x  root apache /var/www/cgi-bin/awlp/engines2.pl
-rwxr-xr-x  root apache
            /var/www/cgi-bin/awlp/error_messages.pl
-rwxr-xr-x  root apache
            /var/www/cgi-bin/awlp/global_configuration.pl
-rwxr-xr-x  root apache /var/www/htdocs/index.html
-rwxr-xr-x  root apache /var/www/cgi-bin/awlp/index.pl
-rwxr-xr-x  root apache /var/www/cgi-bin/awlp/radar.pl
-rwxr-xr-x  root apache /var/www/cgi-bin/awlp/extras.pl
```

- "Copying Linux logo": The management interface of AWLP has a welcoming screen with a Linux logo. The image file of this logo (refer to Section 3.4.4.1) gets copied over in this section.
- "Doing elapsed time calculation": Once reached to this point, it means that the installation has actually finished. Elapsed time calculation gets carried out in this phase of the execution.
- "End of successful installation": A message is shown along with project name, version and the time it took to complete the installation. After 5 seconds, the host machine gets rebooted. Once it is rebooted, it starts running AWLP.

CHAPTER 4

AWLP IN ACTION

In order to run AWLP on a host, you do not have to know all the details of AWLP as it was discussed in previous chapters. You can simply install and start using it on a host system as long as the system meets the requirements for AWLP. Hardware compatibility is another issue since HostAP driver can only work with certain WLAN cards. Moreover, AWLP runs on GNU/Linux Slackware so choosing other peripherals such as NIC (Network Interface Card), graphic display card will require compatibility with Linux. Installing AWLP is straight-forward and easy but preparing a host system for AWLP installation requires considerable effort. Once AWLP is successfully installed, it starts running with pre-configured parameters. However, these parameters can be changed later on through the management interface. It is imperative to field test software after its release to observe its reliability and performance. All these issues will be discussed in subsequent sections of this chapter.

4.1 System Requirements

AWLP can run even on Intel [18] 486 processors but systems with these processors usually lack PCI (Peripheral Component Interconnect) slots. This, in turn, makes it extremely difficult to find compatible WLAN cards and 10/100 Mbps ethernet cards for ISA (Industry Standard Architecture) slots. For this reason, systems with Intel Pentium series (> 75 Mhz) CPU (Central Processing Unit) or above such as Pentium II, III, IV series are recommended. Anything below than 64 MB of RAM

(Random Access Memory) has tendency to become a bottleneck. Therefore, the minimum recommended RAM figure is set to 64 MB but it is possible to run it with 32 MB albeit with degraded performance. In order to successfully install all needed Slackware packages (refer to Appendix A.2), 786 MB of storage space is needed. While installing Slackware, swap space also needs to be setup for virtual memory. In Linux systems, swap space is configured to be twice the size of installed RAM. Thus, an HDD (Hard Disk Drive) with at least 1 GB of storage would be ideal.

4.2 Hardware Compatibility

AWLP is designed to act as a wireless access gateway. So, it needs a WAN (Wide Area Network) interface connected directly to Internet or intranet. For this purpose, it requires an ethernet card compatible with Linux. The most crucial component, however, is WLAN card. For compatibility with HostAP driver, it must have Prism 2/2.5/3 chipset. Wireless cards with these chipsets are listed under "Hardware Compatibility" section of AWLP website - *http://awlp.sourceforge.net*. This online section also contains PCI to PCMCIA (Personal Computer Memory Card International Association) converters that are known to work with Linux. Depending on whether the choice of WLAN card is PCMCIA type and availability of PCMCIA slot on host system, a PCI to PCMCIA converter might also be needed.

4.3 Installation

Installing AWLP consists of three steps that should be followed in order. The first step is to custom install Slackware version 10.0 with tag files provided in AWLP. This will make sure that only needed packages will be installed. The second step is to upgrade Slackware packages. Slackware team releases package upgrades mostly for bug-fix purposes. Upgrading Slackware packages ensures that known problems that might affect the operation of AWLP get fixed. The third and the final step is to install AWLP. To accomplish this, AWLP package should be extracted from its compressed tarball, and *installer.sh* script should be run. It takes about couple of minutes for *installer.sh* to finish execution depending on how fast the system is. After *installer.sh*

completes, the system gets rebooted automatically, and AWLP starts running. Detailed step by step installation instructions are provided under "Installation" section of AWLP website – *http://awlp.sourceforge.net*.

4.4 Default Settings

After the system reboot, followed by end of AWLP installation, the system starts serving wireless clients with pre-configured settings. Table 1 contains these default settings. These settings can be changed later on using AWLP's web-based management interface.

Table 1 – AWLP Default Settings

SSID	AWLP
Channel	6
Beacon Interval	100 ms
DTIM (Dynamic Traffic Indication Message) **Period**	10
Maximum Inactivity	600 seconds
Authentication	Auto
Encryption Algorithm	WEP
Encryption Key	AWLP1

4.5 Sample Setup

As the System Administrator of Department of Industrial Engineering at METU (Middle East Technical University), one of my duties includes maintaining WLAN service in the department. A custom-made wood enclosure (refer to Figure 9) with 300 VA power supply and two 8 dbi omni-directional antennas were in place. D-

Link [19] DWL-6000AP access point was originally in use. I took off this AP, and put a system running AWLP. Hardware configuration of this system is detailed in Table 2, and Figure 8 contains a still image of the system. There are more than 30 WLAN equipped portable computers registered and configured to use WLAN service of the department. This AWLP setup (refer to Figure 10) had been running for 27 consecutive days without any problems as of this writing. No support calls were made about the issue. This showed us that the change was transparent to users.

Table 2 – AWLP Sample Setup Hardware Configuration

CPU	Pentium MMX 200 MHz
RAM	80 MB
HDD	Fujitsu M1636TAU (1.28 GB)
Ethernet Card	Realtek 8139D
WLAN Card	Senao 2511 CDPLUS EXT2
PCI-PCMCIA	Ricoh Co Ltd RL5c475

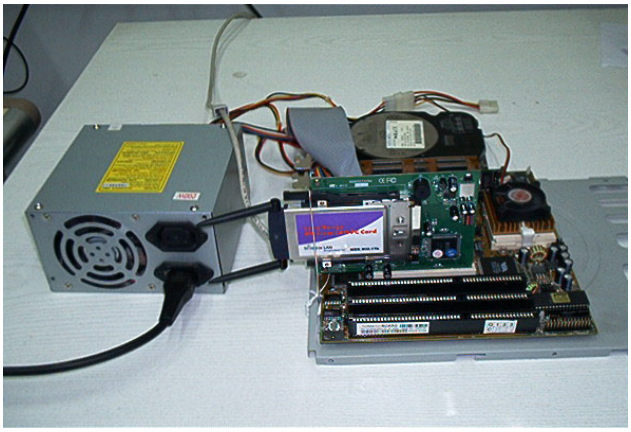

Figure 8 – System Running AWLP

Figure 9 – Wood Enclosure for WLAN Equipment

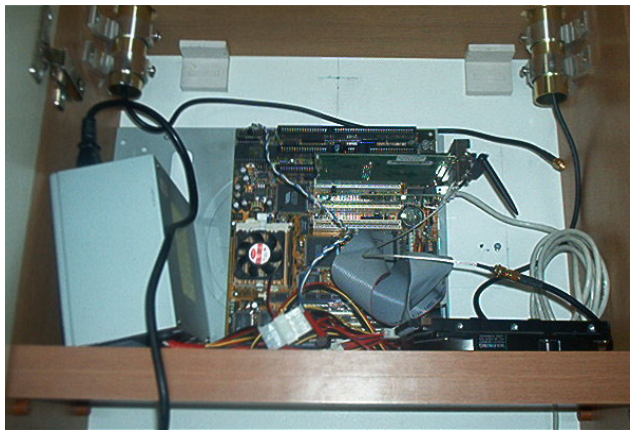

Figure 10 – AWLP Setup in Place

4.6 Performance Tests

In 2004, IEEE 802.11T Task Group was formed to standardize wireless test methods and metrics. However, as of this writing, there is no deliverable by the group to be used as a guideline. Therefore, we turned our attention to benchmarking efforts by 802.11 chipset manufacturers. Our test setup, depicted in Figure 11, was based on the throughput test setup used in the white paper "802.11 Wireless LAN Performance" [20] by Atheros Communications [21], a reputable developer of semiconductor systems for wireless communications products.

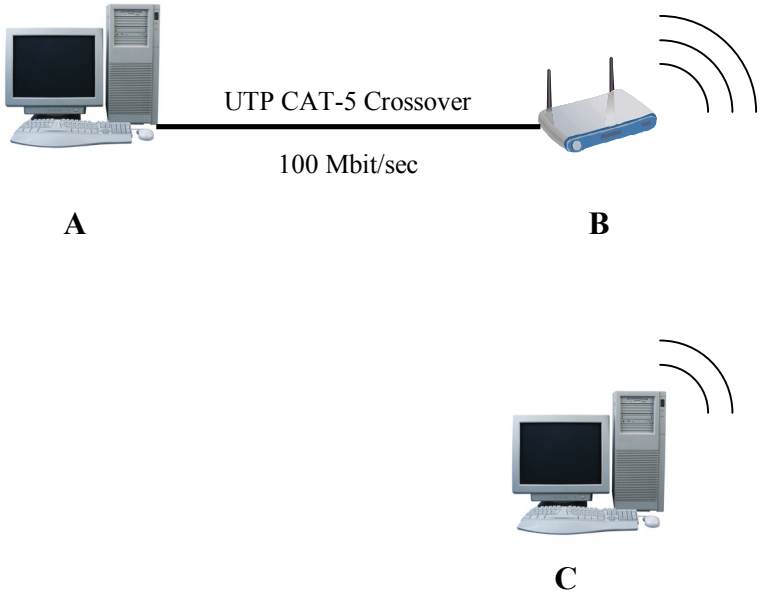

Figure 11 – Throughput Test Setup

In the setup, version 2.1 of the network performance benchmark software called Netperf [22] is used. Computer A runs server component of Netperf and wireless client C runs client component of Netperf. System C is placed within 2 meters of system B with line of sight to avoid any degraded performance. Details about computer systems A, B, and C are given in Table 3.

Although, theoretical maximum application-level throughput for Transmission Control Protocol (TCP) for IEEE 802.11b is 5.9 Mbps as pointed out by the white paper, various outside factors reduces this number; Humidity, temperature, Radio Frequency (RF) complications such as reflection, diffraction, scattering result in lower throughput. Differences in RF circuitry design and 802.11b implementation techniques by different vendors result in varying throughput rates.

Table 3 – Computer Configurations Used in Throughput Test

A	PIV 1.8GHz, 512MB RAM Windows 2000 Pro
B	1)- D-Link DWL-6000AP 2)- HP Compaq Evo N1015v (AMD 1.66GHz, 256MB RAM, Senao 2511 CDPLUS EXT2 – 802.11b PCMCIA)
C	HP d330ut (PIV 2.66GHz, 512MB RAM) Windows XP Pro D-Link DWL-650+ (802.11b - PCMCIA)

Results of our tests are close to each other, thus they are not shown in a graph. With 128-bit encryption (WEP) in effect, D-Link DWL-6000AP has 4.39 Mbps. With the same key-length (128-bit), AWLP, m0n0wall and Pebble have 4.49 Mbps, 4.48 Mbps, 4.49 Mbps throughputs respectively. The same WLAN card being used for testing AWLP, m0n0wall, and Pebble is the most important contributor factor for these close results. D-Link DWL-6000AP's slightly less throughput can be explained by different WLAN card used in this embedded system, and relatively scare resources of RAM in this system when compared to 256MB of RAM in computer system B-2.

4.7 Features Comparison

The most distinguishing features and components of Pebble, m0n0wall and AWLP are compared as shown in Table 4.

Table 4 – Pebble, m0n0wall and AWLP Feature Comparison

	Pebble	m0n0wall	AWLP
OS	GNU/Linux Debian 3.0r1	FreeBSD 4.10	GNU/Linux Slackware 10.0
File System Mounting	· read-only	· read-only · read-write	· read-write
WLAN Driver	· MadWifi · HostAP · Prism54	· wi [4]	· HostAP
IEEE 802.11	802.11b/g	802.11b	802.11b
WPA	No	No	No
Captive Portal	Yes	Yes	No
Firewall	iptables 1.2.6a	ipfw	iptables 1.2.10
DNS	Djbdns	Dnsmasq 1.18	bind 9.2.3
Perl	Perl 5.6.1	N/A	Perl 5.8.4
Web Server	N/A	mini_httpd 1.19	apache 1.3.31
SSH Server	openSSH 3.4	N/A	openSSH 3.8.1
Documentation	Limited	Adequate	Adequate (Comprehensive with this thesis)
Modification Capability	Read-only mounting complicates modification efforts.	Purposefully limited by developer.	Not limited.

As can be seen from Table 4, Pebble is designed for embedded systems with read-only mounting. It is based on old version (3.0r1 – July, 2002) of GNU/Linux Debian. It has

rich WLAN driver support, and the only one to support IEEE 802.11g standard therefore. Although with slightly outdated versions, it has pretty much all necessary components except web server, thus configuration has to be done manually instead of through a web-based interface. Developer Terry Schmidt declares he will not work on documentation, so there is only README file accompanying the software. Lack of documentation is possibly the greatest setback for researchers trying to build WLAN test and development platforms. Moreover, read-only mounting complicates modification and data acquisitions efforts since Pebble cannot write to any local storage device.

m0n0wall is designed to be a viable alternative for commercial firewall products. Thus, its firewall features are stronger. It is based on FreeBSD 4.10. It is the latest version (May, 2004) along the 4-STABLE branch. This branch does not have as wide hardware support as the 5-STABLE branch. Thus, m0n0wall suffers from hardware compatibility when it comes to supporting different network cards and other relevant peripheral components. It has both read-only and read-write mounting options, which makes it possible to install on an embedded platform or on a regular PC with local storage. It does have a web-based management, made possible by integrated web server – mini_httpd. The biggest drawback of m0n0wall for researchers is that system modification capability is purposefully limited by the developer. Custom console is in effect to prevent shell access and to force users to web-based management interface instead.

AWLP is based on relatively new version and latest stable branch (10.0 – June, 2004) of GNU/Linux Slackware. This results in definite advantage over the rest for hardware and peripheral compatibility. Versions of software components it has are also relatively new. Although, its shortcomings along with possible solutions to those are mentioned in detail in Chapter 6, it is important to point out that AWLP lacks captive portal and WPA support. Moreover, it can support only HostAP driver, thus supporting WLAN cards operating with IEEE 802.11g standards is not possible at this point. Comprehensive documentation by taking the thesis into account, modification capability with standard Slackware software package management tools accompanied

with shell access and combined with file system read-write mounting option are AWLP's greatest strength over Pebble and m0n0wall.

CHAPTER 5

CONCLUSION

This thesis has shown a new way of building and deploying a wireless access device using open source software. All source code and related documentation to build a wireless access device were made public at *http://awlp.sourceforge.net* as an open source project. Features, content, implementation, details, tests, advantages and limitations of AWLP were discussed in this document along with related background information. Comparisons with other similar open source software were also included.

5.1 Fast Pace of Requirements Change

Almost all commercially available wireless access devices are closed-boxes, meaning that you will not be able to alter the way they work in any other way than it is allowed by the system. Most manufacturers are releasing firmware upgrades time to time to fix known bugs, and sometimes to improve product features. However, increasing number of wireless clients and increasing need of more versatile management, authentication and security functions are causing change in requirements at a much faster pace than can be answered by improvements of new firmware. The thesis supports the argument that building a custom AP is the solution to these dynamic changes in requirements.

5.2 Building Custom AP with Open Source Software

This thesis is the material proof that building a custom AP requires a great deal of control on configuring OS and various interacting programs and tools. Microsoft [23] offers "Platform Builder" for developers to create customized OS based on "Windows CE" for embedded devices. However, flexibility provided with Linux is superior to that of Microsoft. Licensing and cost of licensing is another issue when using proprietary software, whereas it is free with an open source OS such as Linux. Open source software can be modified and customized. Proprietary software, on the other hand, is basically a black-box not allowing necessary level of customization primarily due to commercial constraints and motives. This thesis demonstrates that building a custom AP is possible and practical using open source software.

On May 19th of 2005, an online introductory article with the title "Build a Wireless Gateway with Perl" is published about AWLP by O'Reilly Media, Inc., a reputable publisher among computer professionals. This article can be reached online at the following URL: *http://www.perl.com/pub/a/2005/05/19/wireless_gw.html*.

CHAPTER 6

FURTHER WORK

It has been shown that AWLP is a viable alternative for setting up wireless access service using a dedicated PC. Result of throughput tests (refer to section 4.6) shows us that AWLP has the same level of performance as competing solutions; m0n0wall and Pebble. It is well documented. It comes pre-configured; it immediately starts serving wireless clients after installation. However, it lacks some advanced features. These lacking features and further work that can be carried out are detailed below:

- MadWifi and Prism54 support
  Most WLAN cards coming out nowadays are IEEE 802.11b/g compliant. These cards cannot be used in AWLP to turn PC into a wireless access device since AWLP has only HostAP driver, which works with wireless LAN cards that have Prism 2/2.5/3 chipset. These cards support IEEE 802.11b only. MadWiFi and Prism54 provides "BSS Master" mode feature that allows WLAN cards with certain chipsets, which are IEEE 802.11g compliant, to act as AP. MadWiFi and Prism54 drivers can be incorporated into AWLP to make it 802.11g compliant.
- 802.1x client support
  Most modern off-the-shelf wireless access devices have 802.1x client support, by which clients can be authenticated against a RADIUS (Remote Authentication Dial-In User Service) server. RADIUS takes care of AAA

(Authentication Authorization Accounting) tasks. AWLP lacks 802.1x client support. AWLP can do authentication by means of ACL and by sharing WEP keys. Jouni Malinen has developed, and has been maintaining hostapd; a daemon that implements IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator. hostapd can be downloaded from HostAP site – *http://hostap.epitest.fi*. It can be incorporated into AWLP to support client authentication using 802.1x.

- Lack of WPA/WPA2 support
  Weakness in protection provided by WEP has forced researchers to find an alternative in wireless data security and integrity. Wi-Fi alliance, previously named WECA (Wireless Ethernet Compatibility Alliance), is a non-profit industry association working to promote WLANs and compatibility between IEEE 802.11 compliant wireless products. Wi-Fi alliance has an alternative to WEP; it is called WPA (Wi-Fi Protected Access). WPA2 is the second version of WPA. Although more secure than WEP, WEP is still being widely used but it will eventually be replaced by WPA. Necessary modifications can be made in AWLP to support WPA/WPA2 capable clients.

- Lack of captive portal
  Number of wireless hotspots, where wireless service is provided to mobile clients for a fee, has been in increase since the last couple of years. With the help of captive portal feature, all outgoing traffic of a client is blocked and all HTTP requests are redirected to a special web page until clients authenticate. Payment methods are incorporated into this authentication step to collect any applicable fees. AWLP does not have an integrated captive portal. Open source community has a possible solution; NoCatAuth [24] is a popular authenticating captive portal written in Perl. Open source captive portal alternatives can be evaluated and incorporated into AWLP.

It is important to point out that AWLP incorporates and uses several programs, tools, commands and features to do its tasks. These programs, tools and commands are orchestrated by AWLP code and configuration files. If new programs and features are

to be added to AWLP, necessary effort needs to be spent to preserve the harmony among these components. The fact that AWLP is open source software provides definite advantages for researchers and developers as they can see and modify 100% of AWLP source code. We hope that documentation, manuals, and the thesis will be great help to them.

REFERENCES

[1] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-1997. (1997). LAN/MAN Standards Committee of the IEEE Computer Society.

[2] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band, IEEE Std 802.11b-1999. (1999). LAN/MAN Standards Committee of the IEEE Computer Society.

[3] Port-Based Network Access Control, IEEE Std 802.1x-2001. (2001). LAN/MAN Standards Committee of the IEEE Computer Society.

[4] Çakırcalı, Alptekin. (2004). AWLP: Alptekin's Wireless Linux Project. *http://awlp.sourceforge.net*

[5] GNU General Public License version 2. (1991). Free Software Foundation, Inc. *http://www.gnu.org/licenses/gpl.html*

[6] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High Speed Physical Layer in the 5 GHz Band, IEEE Std 802.11a-1999. (1999). LAN/MAN Standards Committee of the IEEE Computer Society.

[7] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Further Higher Data Rate Extension in the 2.4 GHz Band, IEEE Std 802.11g-2003. (2003). LAN/MAN Standards Committee of the IEEE Computer Society.

[8] Schmidt, Terry. (2004). Pebble. Retrieved June 10, 2005, from *http://www.nycwireless.net/pebble*

[9] Kasper, Manuel. (2003). m0n0wall. Retrieved June 10, 2005, from *http://www.m0n0.ch/wall*

[10] Murdock, Ian. (1993). Debian: The Universal Operating System. *http://www.debian.org*

[11] The FreeBSD Project. (1995). FreeBSD. *http://www.freebsd.org*

[12] Open Source Technology Group, Inc. (1996). SourceForge.net. *http://www.sourceforge.net*

[13] Slackware Linux, Inc. (1993). The Slackware Linux Project. *http://www.slackware.org*

[14] The Apache Software Foundation. (1995). Apache: HTTP Server Project. *http://httpd.apache.org*

[15] Internet Systems Consortium, Inc. (1996). BIND: Berkeley Internet Name Daemon. *http://www.isc.org/sw/bind*

[16] IEEE, Inc. OUI: Organizationally Unique Identifier. Retrieved December 2004, from *http://standards.ieee.org/regauth/oui/oui.txt*

[17] Hemminger, Stephen. (2004). Linux Ethernet Bridging. Retrieved December 2004, from *http://bridge.sourceforge.net*

[18] Intel Corporation. *http://www.intel.com*

[19] D-Link Corporation. *http://www.dlink.com*

[20] Atheros Communications, Inc. (2003). 802.11 Wireless LAN Performance. Retrieved August 2005, from http://www.atheros.com/pt/whitepapers/atheros_range_whitepaper.pdf

[21] Atheros Communications, Inc. *http://www.atheros.com*

[22] Rick Jones. Netperf. *http://www.netperf.org*

[23] Microsoft Corporation. *http://www.microsoft.com*

[24] Erle, Schuyler. (2001). NoCatAuth. Retrieved June 12, 2005, from *http://nocat.net*

APPENDICES

APPENDIX A. SLACKWARE PACKAGES

A.1. Slackware Software Series

A	The base system. Contains enough software to get up and running and have a text editor and basic communication program.
AP	Various applications that do not require the X Window System.
D	Program development tools. Compilers, debuggers, interpreters, and man pages are all here.
E	GNU emacs.
F	FAQs, HOWTOs, and other miscellaneous documentation.
GNOME	The GNOME desktop environment.
K	The source code for the Linux kernel.
KDE	The K Desktop Environment. An X environment which shares a lot of look-and-feel features with the MacOS and Windows. The Qt library, which KDE requires, is also in this series.
KDEI	The KDE internationalization package series.
L	Libraries.
N	Networking programs. Daemons, mail programs, telnet, news readers, and so on.
T	teTeX document formatting system.
TCL	The Tool Command Language. Tk, TclX, and TkDesk.
X	The base X Window System.
XAP	X Applications that are not part of a major desktop environment (for example, Ghostscript and Netscape).
Y	Games

A.2. Slackware Packages Selection for AWLP

A	sysklogd	KDE	tcpdump
aaa_base	syslinux	–	tcpip
aaa_elflibs	sysvinit		wget
bash	tar	KDEI	wireless-tools
bin	util-linux	–	
bzip2			T
coreutils	AP	L	–
dcron	sudo	glib	
devs		glib2	TCL
e2fsprogs	D	glibc	–
elvis	autoconf	glibc-i18n	
etc	automake	gnet	X
findutils	bin86	libtermcap	–
floppy	binutils	mhash	
gawk	ccache	pcre	XAP
getty-ps	gcc	popt	–
glibc-solibs	kernel-headers	readline	
glibc-zoneinfo	libtool	startup-notification	Y
grep	m4	zlib	–
gzip	make		
hdparm	nasm	N	
hotplug	perl	apache	
kernel-ide	pkgconfig	autofs	
kernel-modules		bind	
lilo	E	dhcp	
minicom	–	dhcpcd	
mkinitrd		dnsmasq	
module-init-tools	F	gnupg	
openssl-solibs	–	inetd	
pciutils		iproute2	
pcmcia-cs	GNOME	iptables	
pkgtools	–	mod_ssl	
procps		ntp	
sed	K	openssh	
shadow	kernel-source	openssl	
slocate		pine	

APPENDIX B. AWLP FILE STRUCTURE

./	**./corescripts/**	**./tagfiles/**
installer.sh	engines1.pl	a/tagfile
COPYING	engines2.pl	ap/tagfile
README	error_messages.pl	d/tagfile
	extras.pl	e/tagfile
./configfiles/	global_configuration.pl	f/tagfile
dhcpd.conf	index.html	k/tagfile
httpd.conf	index.pl	l/tagfile
localhost.zone	radar.pl	n/tagfile
named.ca		t/tagfile
named.conf	**./images/**	tcl/tagfile
named.local	LinuxPowered.gif	x/tagfile
ntp.conf		xap/tagfile
oui_filtered.txt		y/tagfile
rc.dhcpd		
rc.firewall		**./tarballs/**
rc.local		hostap-driver-0.2.5.tar.gz
rc.ntpd		hostap-utils-0.2.4.tar.gz
rc.wlan0		bridge-utils-1.0.4.tar.gz
rc.wlan0-pre		
.htaccess		

APPENDIX C. AWLP CONFIGURATION VARIABLES

$PROJECT_NAME = "AWLP"
Name of the project.

$PROJECT_RELATIVEPATH = "awlp"
This variable helps to construct full paths of AWLP specific folders (i.e., */var/www/cgi-bin/awlp*).

$PROJECT_VERSION = "1.0"
Project version.

$WirelessInterfaceName = "wlan0"
When *hostap-driver* is installed with the help of *installer.sh* , Slackware identifies wireless interface as *wlan0* by default. Just in case things might change in future, name of wireless interface is made configurable with this variable.

$WiredInterfaceName = "eth0"
When Slackware is installed, and it detects a wired interface, it names it as *eth0* . Just in case there might be more than one wired interface, name of wired interface is made configurable with this variable.

$ProcWLANDir = "/proc/net/hostap/${WirelessInterfaceName}/"
Full path of directory under /proc that contains various dynamic information about status of wireless interface.

$PRISM2_PARAM_PATH = "/usr/local/bin/prism2_param"
Full path of *prism2_param* command that comes with *hostap-utils* .

$HOSTAP_CRYPT_CONF_PATH = "/usr/local/bin/hostap_crypt_conf"
Full path of *hostap_crypt_conf* command that comes with *hostap-utils* .

$IWPRIV_PATH = "/sbin/iwpriv"
Full path of *iwpriv* command that comes with hostap-driver.

$IWCONFIG_PATH = "/sbin/ifconfig"
Full path of *ifconfig* command that comes standard with Linux.

$UPTIME_PATH = "/usr/bin/uptime"
Full path of *uptime* command that comes standard with Linux.

$FREE_COMMAND_PATH = "/bin/free"
Full path of *free* command that comes standard with Linux.

$FREE_COMMAND_SWITCHES = "-o"
The switch to *free* command. *get_memory_usage_info* subroutine uses free command with this switch to obtain memory usage information.

$DF_COMMAND_PATH = "/bin/df"
Full path of *df* command that comes standard with Linux.

$DF_COMMAND_SWITCHES = "-T -h -l --no-sync"
The switch to *df* command. *get_disk_usage_info* subroutine uses *df* command with this switch to obtain disk usage information.

$CARDCTL_PATH = "/sbin/cardctl"
Full path of *cardctl* command that comes with pcmcia card services (cardmgr) for Linux.

$STANDARD_SHELL = "/bin/bash"
Full path of default shell on the host machine.

$IEEE_OUI_FILTERED_PATH = "/etc/${PROJECT_RELATIVEPATH}/ oui_filtered.txt"
Full path to the filtered version of IEEE OUI (Organizationally Unique Identifier) file.

$CHECK_FOR_UPDATES_PATH = "/tmp/CheckForUpdates.txt"
Full path to the file that will temporarily contain result while checking AWLP updates site.

$CHECK_FOR_UPDATES_URL = "http://www.cakircali.com/cgi-bin/AWLP_CheckUpdates.pl"
URL for checking AWLP updates.

$HTACCESS_CONFIG_PATH = "/var/www/cgi-bin/${PROJECT_RELATIVEPATH}/.htaccess"
Full path to *.htaccess* file that controls HTTP authentication.

$HTPASSWD_CONFIG_PATH = "/var/www/cgi-bin/${PROJECT_RELATIVEPATH}/.htpasswd"
Full path to *.htpasswd* file that contains valid username and password pairs for HTTP authentication.

$RCWLAN_PATH = "/etc/rc.d/rc.${WirelessInterfaceName}"
Full path of the file that initializes wireless interface.

$FIREWALL_CONFIG_PATH = "/etc/rc.d/rc.firewall"
AWLP comes with a firewall feature. This variable defines the full path for the firewall configuration file.

$DHCPD_LEASES_PATH = "/var/state/dhcp/dhcpd.leases"
/var/state/dhcp/dhcpd.leases file contains details on IP leases given by DHCP service.

@PRISM2_PARAM_LIST = ('beacon_int', 'dtim_period', 'antsel_rx', 'antsel_tx', 'ap_max_inactivity', 'ap_bridge_packets', 'max_wds', 'autom_ap_wds', 'ap_auth_algs', 'host_encrypt', 'host_decrypt', 'ieee_802_1x', 'wds_type', 'basic_rates', 'oper_rates', 'hostapd')
This array contains the list of valid PRISM2 parameters.

@ISM_FREQ_MIDPOINTS	= (2.412, 2.417, 2.422, 2.427, 2.432, 2.437, 2.442, 2.447, 2.452, 2.457, 2.462, 2.467, 2.472)
This array contains midpoint frequencies for the first 13 channels of IEEE 802.11b.	

%WELL_KNOWN_PORTS	= (7 => 'Echo', 13 => 'Daytime', 20 => 'FTP-Data', 21 => 'FTP-Control', 22 => 'SSH', 23 => 'Telnet', 25 => 'SMTP', 37 => 'Time', 42 => 'WINS', 43 => 'Whois', 53 => 'DNS', 67 => 'BOOTP Server', 68 => 'BOOTP Client', 69 => 'TFTP', 79 => 'Finger', 80 => 'HTTP', 110 => 'POP3', 113 => 'Authentication', 115 => 'sFTP', 119 => 'NNTP', 123 => 'NTP', 137 => 'NETBIOS Name', 138 => 'NETBIOS Datagram', 139 => 'NETBIOS Session', 143 => 'IMAP', 161 => 'SNMP', 443 => 'HTTPS', 445 => 'SMB', 531 => 'IRC', 554 => 'RTSP', 1521 => 'Oracle SQL', 1645 => 'RADIUS Authentication', 1646 => 'RADIUS Accounting', 2049 => 'NFS', 3306 => 'MySQL', 8080 => 'HTTP-Alternate')
AWLP's firewall has port-based outgoing filter. This hash contains port number-name pairs for well known ports such as FTP, HTTP, POP3, etc.	

$PRISM2_MAX_WEP_KEYS	= 4
Most wireless cards allow up to 4 (four) WEP keys to be defined. This configuration variable defines this value.	

$PRISM2_PARAM_MAX_VALUE	= 100000
Maximum value for any PRISM parameter.	

$MAX_CHANNEL_NUMBER	= 13
Laws and regulations allow 11 channels to be used in the US for 2.4 GHz ISM band. Europe allows 12nd and 13th channel to be used in the same band, 14th channel is allowed in Japan. This variable defines the maximum channel number, which is to be set based on location.	

$BEACON_INT_MIN_VALUE = 10
Minimum value for beacon interval. It defines the interval in milliseconds for each consecutive beacon frame.

$BEACON_INT_MAX_VALUE = 1500
Maximum value in milliseconds for beacon interval.

$DTIM_PERIOD_MIN_VALUE = 1
Minimum value for DTIM period. It defines how many beacon frames will elapse before sending beacon frame that contains DTIM.

$DTIM_PERIOD_MAX_VALUE = 100
Maximum value for DTIM period.

$AP_MAX_INACTIVITY_MIN_VALUE = 60
Minimum value in seconds before a client is kicked out from AP due to inactivity.

$AP_MAX_INACTIVITY_MAX_VALUE = 1800
Maximum value for the inactivity timeout.

$SNR_RADAR_MAXIMUM_WIDTH = 200
SNR radar is displayed in a separate window. This configuration variable defines maximum width for this window in pixels.

$SNR_RADAR_REFRESH_INTERVAL = 10
This configuration variable defines the refresh interval of SNR radar in seconds.

$BW_RADAR_MAXIMUM_WIDTH = 200
Bandwidth radar is displayed in a separate window. This configuration variable defines maximum width for this window in pixels.

$BW_RADAR_REFRESH_INTERVAL = 5
This variable defines the refresh interval of bandwidth radar in seconds.

$DHCP_LEASE_TABLE_REFRESH_INTERVAL = 60
This variable defines the refresh interval for DHCP lease table window.

$ALLOWED_OUTGOING_PORTS_MAX_NUMBER = 100
Maximum number of outgoing ports that can be opened in the firewall.

$ALLOWED_OUTGOING_PORTS_INCREMENT_VALUE = 5
Number of empty text boxes for new ports in the firewall configuration page.

$LOGON_SECURITY_REALM_MIN_LENGTH = 3
AWLP management page has password-protected authentication. While changing realm of this authentication, this variable sets the minimum length for it.

$LOGON_SECURITY_REALM_MAX_LENGTH = 50
Maximum number of characters for the authentication realm.

$LOGON_SECURITY_USERNAME_MIN_LENGTH = 3
This variable sets minimum character limit for the authentication username.

$LOGON_SECURITY_USERNAME_MAX_LENGTH = 12
Maximum number of characters for the authentication username.

$LOGON_SECURITY_PASSWORD_MIN_LENGTH = 3
This variable sets minimum character limit for the authentication password.

$LOGON_SECURITY_PASSWORD_MAX_LENGTH = 12
Maximum number of characters for the authentication password.

$DF_CAPACITY_ALARM_LEVEL = 90
AWLP management shows status information of the system including free disk space. If used disk space percentage is greater than or equal to this value for a partition, its detail will be shown in color red instead of black.

$MAIN_SCRIPT_NAME = "index.pl"
Name of the script responsible for outputing main window.

$RADAR_SCRIPT_NAME = "radar.pl"
Name of the script responsible for outputing radar pages; SNR radar, bandwidth radar.

$EXTRAS_SCRIPT_NAME = "extras.pl"
Name of the script responsible for outputting pages that contains functionalities such as ASCII to hexadecimal conversion, error codes list.

$CHARACTER_SET = "UTF-8"
Character set to be used while outputing dynamically generated HTML pages. The default value is UTF-8 for easily displaying most of international characters.

$MAIN_TITLE = "${PROJECT_NAME} - Alptekin's Wireless Linux Project"
Title of the main window of AWLP's web-based management interface.

APPENDIX D. AWLP SUBROUTINES

Subroutines that were defined in engines1.pl and engines2.pl are listed below with input, output specifications and descriptions. In Perl, a scalar variable has "$" prefix, an array has "@", and a hash has "%". These prefixes are used in subroutine definitions in this section to describe types of input and output arguments. For example, "@ = &subroutine_name($, $)" implies us that this subroutine accepts two scalar variables, and return an array. If an input argument is conditionally required, it is suffixed with "?". If there is more than one alternative for output type of subroutine, they are separated by "|". "$ | % = &subroutine_name($, $?)" implies that first input argument is required, and second one is optional. Subroutine returns either a scalar or a hash value.

@ = &proc_get_client_mac_list($)	
Input	- Full path of directory where client information is held.
Output	- Array of MAC addresses.
Desc	This subroutine gets list of wireless clients by using *ls* command on the directory specified as the input argument. This directory resides under */proc* directory, and contains information about connected clients. The subroutine returnes MAC addresses of connected clients in an array.

% = &proc_get_client_mac_details($, $)	
Input	- MAC address of client in question. - Full path of directory where client information is held.
Output	- Hash containing details and statistics for given MAC address.
Desc	Under */proc* directory, there exists a file for each MAC address associated with AP. These files have the same name as their associated MAC addresses. This subroutine parses out the corresponding file to get detailed information and statistics about the client in question. The subroutine returns the result in a hash.

% = &proc_get_prism2_parameters($, $)	
Input	- Full path of *prism2_param* command that comes with *hostap-utils* . - Name of wireless interface (i.e., *wlan0*).
Output	- Hash containing wireless card configuration parameters.
Desc	This subroutine uses *prism2_param* command that comes with *hostap-utils* to get values of parameters listed in *PRISM2_PARAM_LIST* configuration array, that is defined in *global_configuration.pl* . The parameters in this array are specific to wireless nature of AP such as beacon interval, DTIM period, etc.

$ = &set_prism2_parameter($, $, $, $)	
Input	- Full path of *prism2_param* command. - Name of wireless interface (i.e., *wlan0*). - Parameter name whose value is to be set. - New value of the parameter to be set.
Output	- Scalar variable containing output of *prism2_param* command, which is 0 (zero) if parameter setting operation is succeeded.
Desc	The subroutine makes sure that parameter to be set is greater than or equal to 0 (zero), and it is less than or equal to *PRISM2_PARAM_MAX_VALUE* configuration variable.

% = &proc_get_AP_stats($)	
Input	- Full path of directory in which the file with AP statistics resides.
Output	- Hash containing statistics about AP.
Desc	This subroutine gathers statistics about AP by simply parsing the content of *stats* file that resides in the directory specified by the input argument.

% = &proc_get_AP_debug($)	
Input	- Full path of directory where the file with AP debug information resides.
Output	- Hash containing information about the operation of AP.
Desc	The subroutine reads and parses out the content of *ap_debug* file, which resides in the directory specified by the input argument. It contains couple of configuration variables such as authentication method in use, maximum inactivity timeout, and other information regarding the state of AP rather than detailed debug information as the name of the subroutine might imply.

% = &proc_get_debug($)	
Input	- Full path of directory where the file about debug information resides.
Output	- Hash containing debug information about the operation of the device.
Desc	The subroutine reads and parses out the content of *debug* file just like *proc_get_AP_debug* subroutine does. The content of *debug* file is slightly more comprehensive than *ap_debug* file but basically contains parameter-value pairs about the state of the device.

$ = &add_mac_to_acl($, $, $)	
Input	- Full path of *iwpriv* command that comes with *hostap-driver*. - Name of wireless interface. - MAC address to be added to ACL.
Output	- Scalar variable containing output of *iwpriv* command, which is 0 (zero) if the operation succeeds.
Desc	The subroutine uses *iwpriv* command with *addmac* switch to add given MAC address to ACL.

$ = &remove_mac_from_acl($, $, $)	
Input	- Full path of *iwpriv* command that comes with *hostap-driver*. - Name of wireless interface. - MAC address to be removed from ACL.
Output	- Scalar variable contaning output of *iwpriv* command, which is 0 (zero) if the operation succeeds.
Desc	The subroutine uses *iwpriv* command just like *add_mac_to_acl* subroutine with only one difference; it uses *delmac* switch instead to remove given MAC address from ACL.

$ = &kick_station_from_AP($, $, $)	
Input	- Full path of *iwpriv* command that comes with *hostap-driver* . - Name of wireless interface. - MAC address to be kicked from AP.
Output	- Scalar variable contaning output of *iwpriv* command, which is 0 (zero) if the operation succeeds.
Desc	The subroutine uses *iwpriv* command with "kickmac" switch to kick given station from AP. In most cases, station that was kicked will attempt to re-associate and re-authenticate immediately, and it will succeed if it is not blocked through ACL.

% = &ACL_status($)	
Input	- Full path of directory where ap_control file resides.
Output	- Hash containing *MACPolicy* , *MACEntries* , *MACList* keys.
Desc	The subroutine parses out content of *ap_control file* , which resides under */proc* . This file contains details about the status of ACL.

$ = &is_MAC_in_ACL($, $)	
Input	- Full path of directory where *ap_control* file resides.
Output	- Scalar variable with values, 1 (one) or 0 (zero).
Desc	The subroutine uses a regular expression to extract given MAC address from *ap_control* file. If MAC address is found in *ap_control* file, 1 (one) is returned as the result otherwise, 0 (zero) is returned.

$ = &ACL_policy_modify($, $, $)	
Input	- Full path of *iwpriv* command. - Name of wireless interface. - Variable to control modification to be made to ACL policy.
Output	- Scalar variable, which is 0 (zero) if the operation succeeds. If there is a problem, it will be a non-zero value.
Desc	The subroutine uses *iwpriv* command with *maccmd* switch to change ACL policy. The third input argument, which is a scalar variable, defines the modification to ACL policy: 0 - Change the ACL policy to "open". 1 - Change the ACL policy to "allow". 2 - Change the ACL policy to "deny". 3 - Flush MAC address control list 4 - Kick all authenticated clients

$ = &WDS_add_MAC($, $, $)	
Input	- Full path of *iwpriv* command. - Name of wireless interface. - MAC address to be added to WDS.
Output	- Scalar variable containing output of *iwpriv* command, which is 0 (zero) if the operation succeeds.
Desc	The subroutine uses *iwpriv* command with *wds_add* switch to add given MAC address to WDS (Wireless Distribution System).

$ = &WDS_remove_MAC($, $, $)	
Input	- Full path of *iwpriv* command. - Name of wireless interface. - MAC address to be removed from WDS.
Output	- Scalar variable containing output of *iwpriv* command, which is 0 (zero) if the operation succeeds.
Desc	The subroutine uses *iwpriv* command with *wds_del* switch to remove given MAC address from WDS.

$ = &Card_Identification($)	
Input	- Full path of *cardctl* command that is part of cardmgr daemon.
Output	- Scalar variable containing output of *cardctl* command with *ident* switch.
Desc	This subroutine captures output of *cardctl* command with *ident* switch. This output, if command succeeds, contains information about PCMCIA card(s) in the system, either connected directly or through a PCI-PCMCIA converter.

$ = &Card_Port_reset($, $, $)	
Input	- Full path of *cardctl* command. - Name of wireless interface. - An integer to control the state of wireless card.
Output	- Scalar variable containing output of *cardctl* command with *reset* switch.
Desc	This subroutine uses *cardctl* command with *reset* switch to change the state of wireless card in the system. Just like *Card_Identification* subroutine, this subrotine can reset only PCMCIA cards either connected directly or through a PCI/PCMCIA converter. The third input argument defines the state: 0: perform soft reset of the card 1: perform COR reset 2: perform port reset 3: disable port 0 4: enable port 0

$ = &Encryption_control($, $, $, $?, $?, $?, $?)	
Input	- Full path of *hostap_crypt_conf* command that comes with *hostap-utils* . - Name of wireless interface. - Scalar variable defining action to be taken: SetDefaultKey: Setting default WEP keys ListDefaultKeys: Listing WEP keys (Arguments below are required if the third one is set to "SetDefaultKey") - Name of encryption method (i.e., *WEP*). - Number index of WEP key to be set. - WEP key to be set. - Scalar variable defining if WEP key to be set is the default TX key.
Output	- Depending on the third input argument, it is either a scalar variable containing output of *hostap_crypt_conf* command or a hash containing WEP keys.
Desc	This is the subroutine for configuring WEP keys. The subrotine uses *hostap_crypt_conf* command that comes with *hostap-utils* . Depending on value of the third input argument, this subroutine either lists WEP keys or sets WEP keys defined through four additional input arguments.

% = &check_daemon_status($, $)	
Input	- Daemon name to be checked: *dhcpd* : DHCP Server *dhcpcd* : DHCP Client *ntpd* : Network Time Server *httpd* : HTTP Server *named* : DNS Server *sshd* : SSH Server *syslogd* : Syslog Server *cardmgr* : PCMCIA Card Manager Server *crond* : Cron Daemon - Name of wireless interface.
Output	- Hash containing status of daemon in question.
Desc	The subroutine uses *ps* command with "-ealf" switch to capture list of running processes. Based on the first input argument, the subroutine searches this process list. Regardless of the daemon being sought is found or not, a hash with *Code* and *Info* keys are returned. If daemon is found to be working, value of *Code* key is set to 1 (one) otherwise, it is set to 0 (zero).

% | $ = &Logon_Security_control($, $, $, $?, $?, $?)	
Input	- Full path of *.htaccess* file. - Full path of *.htpasswd* file. - Scalar variable defining action to be taken: 0: Lists logon security info except for password. 1: Sets logon security info as specified by additional arguments. (Arguments below are required if the third one is set to "1") - Authentication realm to be set. - Authentication username to be set. - Authentication password to be set.
Output	- It is a scalar variable containing output or a hash containing authentication realm and username depending upon the third input argument.
Desc	This subroutine modifies content of *.htaccess* and *.htpasswd* files, which control the authentication process. Depending on the third input argument, this subroutine either lists authentication settings or sets them with values specified by additional input arguments.

% = &get_uptime_info($)	
Input	- Full path of *uptime* command that comes with Linux.
Output	- Hash containing uptime and load average info of the host machine.
Desc	This subroutine uses Linux standard *uptime* command to get elapsed time since the host machine has been started. The subroutine also parses out average load information from output of *uptime* command.

$ = &get_card_manufacturer_info($, $)	
Input	- Full path of *oui_filtered.txt* file. - MAC address in question.
Output	- Scalar variable containing manufacturer information for given MAC address.
Desc	The subroutine searches *oui_filtered.txt* file for given MAC address, and returns organization information corresponding to this MAC address.

@ = &get_memory_usage_info($, $)	
Input	- Full path of *free* command that comes standard with Linux. - Switch to be used with *free* command.
Output	- Array containing lines of memory and swap space usage information.
Desc	The subroutine uses Linux standard *free* command to get memory and swap space usage information. For calling script to easily format the output, the subroutine returns the result in an array, each array member containing a line.

@ = &get_disk_usage_info($, $)	
Input	- Full path of *df* command that comes standard with Linux. - Switch to be used with *df* command.
Output	- Array containing lines of disk space usage information
Desc	A call to Linux standard *df* command is made within this subroutine to get disk space usage information. For calling script to easily format the output, this subroutine returns the result in an array, each array member containing a line.

% = &get_dhcpd_lease_info($, $, $)	
Input	- Full path of the file that contains DHCP lease info. - String to be sought in the lease file. - Variable to specify search and list methods to be used: 0: Search by MAC 1: Search by IP 2: List all DHCP leases
Output	- Hash containing relevant DHCP lease information.
Desc	Based on the third input argument, this subroutine either searches for given string or lists all DHCP lease information. While searching, the third input argument defines whether string to be sought is a MAC address or an IP address. The subroutine uses content of *dhcpd.leases* file, which contains all DHCP lease information.

% = &iwconfig_parser($, $)	
Input	- Full path of *iwconfig* command. - Name of wireless interface.
Output	- Hash containing several key-value pairs about wireless card configuration.
Desc	The subroutine uses *iwconfig* command to get detailed information about wireless interface on the host machine.

% = &ifconfig_parser($, $)	
Input	- Full path of *ifconfig* command that comes standard with Linux. - Name of wired or wireless interface (i.e., *eth0* , *wlan0*)
Output	- Hash containing several key-value pairs about the interface.
Desc	The subroutine uses *ifconfig* command to get detailed information about the interface in question.

$ = &AP_Wireless_manage($, $, $, $?, $?)	
Input	- Full path of *iwconfig* command. - Name of wireless interface. - Variable to specify action to be taken by the subroutine: MasterMode: Changes the wireless card to "Master" mode MonitorMode: Changes the wireless card to "Monitor" mode SetSSID: Sets SSID of the wireless card SetChannel: Sets channel number of the wireless card (One of the arguments below is required depending on the third input argument) - New SSID of the wireless card. - New channel number of the wireless card.
Output	- Scalar variable containing result of *iwconfig* command.
Desc	This subroutine manages wireless card on the system with the help of *iwconfig* command. Setting the mode of a wireless network card helps to put the card into AP mode or put into passive monitor mode. Switching between these two modes helps to activate and de-activate AP function.

$ = &AP_LAN_manage($, $, $, $?, $?, $?)	
Input	- Full path of *ifconfig* command. - Name of either wired or wireless interface (i.e., *eth0* , *wlan0*). - Variable to specify action to be taken by the subroutine: TurnUp: Turns the interface up TurnDown: Turns the interface down Restart: Turns the interface down, then turn it back up SetIPConfig: Sets IP configuration of the interface (Arguments below are required if the third input argument is set to "SetIPConfig") - IP number of the interface. - Netmask of the interface. - Broadcast address of the interface.
Output	- Scalar variable containing output of *ifconfig* command.
Desc	The subroutine uses *ifconfig* command that comes standard with Linux to turn up and down the wired or wireless interface. If the third input argument is set to "SetIPConfig", the subroutine modifies IP configuration for the interface using the additional input arguments provided.

$ = &System_reboot()	
Input	- None. (There is no input argument for this subroutine)
Output	- Scalar variable containing output of *reboot* command.
Desc	This subroutine simply calls *reboot* command that comes standard with Linux. *reboot* initiates an immediate restart on the host machine.

$ = &sort_ClientFlags_byvalue()	
Desc	This is a special type of subroutine that helps to sort string values.

$ = &sort_OperationRates_bykey()	
Desc	This is a special type of subroutine that sorts operation rates.

$ = &sort_OpenPorts_byportnumber()	
Desc	This is a special type of subroutine that sorts port numbers, which are integers ranging from 0 to 65535.

$ = &find_ISM_channel_number($)	
Input	- Mid-point frequency (in GHz) of channel in question.
Output	- Scalar variable containing channel number.
Desc	*iwconfig* command lists wireless card operating frequency but it does not explicitly list channel number associated. The subroutine finds channel number given mid-point frequency.

$ = &convert_dec_to_binary($, $)	
Input	- Decimal number to be converted into binary. - Length of binary digits (i.e., *8*).
Output	- Scalar variable containing resultant binary number.
Desc	This subroutine converts given decimal number into binary number. The result is returned as a string in the length specified as the second input argument.

$ = &convert_binary_to_dec($)	
Input	- Binary number to be converted into decimal.
Output	- Scalar variable containing resultant decimal number.
Desc	This subroutine simply converts given binary number into decimal.

$ = &find_broadcast_address($, $)	
Input	- IP number in dotted format (i.e., *192.168.0.1*). - Netmask value in dotted format (i.e., *255.255.255.0*).
Output	- Scalar variable containing broadcast address in dotted format (i.e., *192.168.0.255*).
Desc	This subroutine finds broadcast address given IP number and netmask.

$ = &show_date_time($)	
Input	- Integer controlling output format of date and time to be shown.
Output	- Scalar variable containing date and time information.
Desc	This subroutine uses *date* command that comes standard with Linux. *date* command is run with various switches in order to attain the desired output format.