Regis University
ePublications at Regis University

All Regis University Theses

Fall 2007

An Implementation of a Cross-Platform Wireless Router Operating System

David Hunt
Regis University

Follow this and additional works at: https://epublications.regis.edu/theses

⚛ Part of the Computer Sciences Commons

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

AN IMPLEMENTATION OF A CROSS-PLATFORM

WIRELESS ROUTER OPERATING SYSTEM

By

David Hunt

A professional project paper in partial fulfilment of

the requirements for the degree of

Masters of Science

(Software and Information Systems)

at

National University of Ireland (Galway)

and

Regis University, Colorado.

2007

Certification of Authorship

Student's Name	David Hunt
Email	daveh@climbing.ie
Telephone	+353 86 8109 704
Date of Submission	16th August 2007
Program	Masters of Science in Software and Information Systems
Title of Submission	An Implementation of a cross-platform Wireless Router Operating System
Advisor	Dr. Desmond Chambers, National University of Ireland, Galway.

Certification of Authorship: I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for the purpose of partial fulfilment of requirements for the Degree Program.

_____   14th August 2007

Student Signature   *Date*

Revision History

Version	Date	Description
0.1	Oct 22 2006	Initial Framework
0.2	July 1 2007	First Draft
0.3	August 6 2007	Final Draft 1
1.0	August 14 2007	Final Revision

Abstract

The aim of this project was to develop an open-source wireless router platform on several Single Board Computers (SBCs) of different architectures. It is a Linux distribution that is common to all supported boards, with Ethernet drivers, Wireless drivers, web-interface, persistent storage of settings, and remote upgrade capability.

Acknowledgements

Many thanks to my advisor, Dr. Desmond Chambers, who made the topic selection seem almost obvious.

Thanks to those from NUI who attended the (very helpful) thesis workshops, and special thanks to those who travelled all the way to Ireland from Regis, Colorado to attend.

Thanks, Dave, for having such a level head.

Thanks to my parents. I wouldn't have done it without them.

Most thanks goes to my wife, Therese, and my three daughters, Laura, Rachel and Jessie, for putting up with my endless hours locked away at my computer.

Definition of Terms

SBC	Single Board Computer
SNMP	Simple Network Management Protocol
DHCP	Dynamic Host Configuration Protocol
GUI	Graphical User Interface
SSH	Secure SHell
MADWIFI	Multimode Atheros Drivers for WIFI
HAL	Hardware Abstraction Layer
NVRAM	Non Volatile Random Access Memory
TFTP	Trivial File Transfer Protocol
MRTG	Multi-Router Traffic Grapher
UML	User Mode Linux

Table of Contents -

1 Introduction

1.1 Statement of Problem

Initial research has shown very few cross-platform implementations of wireless router operating systems. While there are open-source drivers and applications available for these boards, most of them have only proof-of-concept distributions available. For example, Mikrotik's routerboard's reference Debian image is missing basic functionality such as an editor, and without any persistent storage of configuration changes. Likewise, the Gateworks Avila board has a Snapgear Linux implementation, but again, no persistent storage of changes.

There are also sometimes difficulties with existing commercial implementations of wireless router operating systems, where issues are discovered, but due to the limitations of resources of the vendor, the problem might go unaddressed for some time. Having open source alternatives allows issues in the software to be addressed sooner, by those individuals with the expertise and time available.

Of course, this raises the interesting issue of support - there seems to be a 'critical mass' of a user base, without which the project will lose momentum and fade away, as is the case with many open-source projects. It is hoped that this project may gain enough momentum in the local wireless community to become more widely used, and further developed. There is already interest in the project proposal by some local wireless communities and local internet service providers.

1.2 Thesis Statement

There are many commercial wireless operating systems available. However, there are very few available as open-source, and fewer again that cover multiple architectures and allow easy upgradeability while conserving configuration. This causes problems where the systems are in inaccessible locations such as on top of large communications towers, as the system may need to be physically accessed in order to upgrade the software. Another issue with commercial solutions is that it is difficult to review any security issues, as the source is not generally available.

The proposed solution is to implement an open-source cross-platform Wireless Router Operating System using readily available software and drivers, and design and develop a web-interface and mechanism for examining the existing configuration of the system, and merging it into the configuration of the upgraded components. The focus will be on the connectivity configuration, where it is imperative that after an upgrade, the system is still contactable.

1.3 Project Need

What this project hopes to produce is a viable, usable Linux implementation, with a set of features that will allow installation of a working wireless operating system on any of the supported platforms with a minimum amount of difficulty. Basically, flash the CF or on-board flash, make a few configurations changes, and it should be up-and-running.

The Single Board Computers selected for inclusion in the case study implementation were:

1.   PC Engines WRAP2C (x86)   (www.pcengines.ch)

2.   Mikrotik Routerboard 532 (mips)   (www.mikrotik.com)

3.   Gateworks Avila 2348-4 (xscale)   (www.gateworks.com)

The existing commercial solutions for wireless routing operating systems have given the author several years of reliable service in the community broadband scheme in which he is involved, yet they all have some shortcomings (section 3.10 and section 4.4.2). It is hoped that the implementation of an open-source solution will allow the author to address any shortcomings, and adapt the solution for a closer fit to the environment in which it will be used.

Also, there have been some high-profile bugs with particular operating systems [16] which could have easily been addressed had the operating system been open source, rather than having to wait for the commercial developers to release a new update to the software.

1.4   Project Completion Criteria

In order to limit the scope of the case study, some completion criteria needed to be put in place. These criteria would define the point at which the project development would come to and end. This is not to say that that the operating system in question would be complete, but there needed to be an agreed point at which development stopped and the conclusions of the case study would be documented.

The scope of the project was limited to the following:

- Core Functionality
    - Common Linux Kernel (2.4.x)
    - Common file system layout
    - Common boot scripts
    - Common drivers (where applicable)
- Web Interface
    - IP address, netmasak and gateway for each interface.
    - Wireless settings, including distance and encryption settings for each wireless interface.
    - Routing settings, add/update/delete routes.
    - Basic firewall settings with iptables.
    - Feature enable/disable (e.g. switch on/off dhcp daemon, etc.)
    - Statistics/log viewer.
- Upgradability
    - Kernel and root file system remote upgrade

- o   Migrate IP and Wireless settings across upgrades

- Configuration Settings

   - o   Save in Persistent Storage

   - o   Settings recovery if corrupt.

- In-service Usage

   - o   Installation as part of live wireless environment.

The software packages available on this distribution will be at least as follows:

- Madwifi wireless drivers for Atheros Wireless Devices.

- Web server for configuration front-end.

- OpenSSH for secure logins.

- SNMP for statistics gathering by remote hosts.

- DHCP for easy configuration of clients.

- IpTraf and TCPDump for network debugging.

- IPTables firewalling.

- Quagga routing daemon for various routing protocols.

1.5 Project Task Plan

In order to help implementation, the practical portion of the case study was split into several stages, which are as follows:

1. The initial task of the project is to choose a tool-chain for building the operating system. Preliminary research has shown the Buildroot System to be an interesting choice, as it covers a wide variety of CPU architectures.

2. The choice of kernel is also important, as a stable, well-supported kernel is required. It is currently thought that a 2.4.x kernel will be used, as there is existing support for each of the chosen boards by each of the board manufacturers for this kernel.

3. Chose a wireless radio that will give most useful functionality. The Atheros based radio cards offer most of the functionality needed, and have driver support for Linux in the form of the Multimode Atheros Drivers (MADWIFI).

4. Build basic operating system on all three boards.

5. Start integrating packages into the buildroot system for easy compiling across platforms.

6. Design and develop web-front end.

7. Design and develop remote upgrade functionality

8. Design and develop configuration storage and migration software.

9. Test in a live environment, with at least one SBC placed in a waterproof box on a roof, connecting into an existing Wireless Wide Area Network in the locality. This will give us an idea of the stability of the new operating system.

10. Develop presentation in preparation of professional project delivery.

1.6 Capstone Project Paper Organization

This paper encompasses five chapters. Chapter 1 introduces the project by identifying the aim(s)of the paper, the project completion criteria and the project plan. Chapter 2 gives a comprehensive review of the research that was carried out into multi-platform, open-source operation systems and the selection of the various components of the case study. Chapter 3 will detail the case study that was implemented to investigate the thesis statement, i.e. the cross-platform wireless router operating system, the components developed as part of that project. Chapter 4 will detail the findings of the implementation of the project, along with any insight discovered and problems encountered. Chapter 5 will discuss any conclusions, and whether the thesis statement is valid, having implemented the case study project.

2 Review of Research

2.1 Research methods to be used

The author carried out initial research into the current state of development on cross platform single-board computers. This research helped the choice of tool-chain (the development environment) upon which the project was based. Since this is a particularly practical project, the research component will be small, yet important. The main research component will be into the best method of migrating configuration settings from one version of a component to another. It is also important to maintain connectivity in the case of a mis-configuration, so research will be carried out into the current methodologies of rolling back changes if particular conditions are met. Some commercial operating systems offer this feature, but research so far has no open-source methods available. Once the thesis statement was written, there were several important choices to be made with regards to putting together the case study. These are detailed in the following sections of this chapter.

2.2 Toolchain Selection

The author investigated the current state of development of embedded Linux. This was to see what was available in the open-source embedded Linux area, and also to help the author decide on the best development system toolchain on which to base the case study for the capstone project development.

Several development systems were looked at. PeeWeeLinux [21] was a similar project to the proposed case study, with its main aim being "an ongoing development effort to provide an environment that makes the configuration and installation of a Linux operating system on an embedded platform as easy and painless as possible". This aim is shared by the author in the development of the case study discussed below. Unfortunately, the PeeWeeLinux OS (Operating System) is built for x86 platforms only, a trend that would crop up again and again in existing open-source embedded Linux solutions. muLinux [22] was another distribution investigated. This was more aimed at old desktop machines, rather than embedded system, and again was limited to x86 machines. LEAF [23] (Linux Embedded Application Firewall) was another distribution investigated. As with previous implementations, this was also x86 based. The author had previous experience with this platform and thought that the development environment was quite limiting, in that it used a method of UML (User Mode Linux) in which the programmer developed modifications, making it quite difficult to set up. Also, the package delivery method of the LEAF environment was thought to be confusing to users, as was the complex method of storing configuration changes.

With so many implementations limited to x86 architectures, the author looked to Busybox for a solution. One of the main requirements of the project was that it should be cross-platform, and this seemed to the author to be an ideal solution. The developers of Busybox [2] describe it as "The Swiss Army Knife of Embedded Linux". In looking at Busybox, the author found that there was a related website called Buildroot, which is "a set of Makefiles and patches that makes it easy generate a cross-compilation toolchain and root filesystem for your target Linux system using the uClibc C library". It included

support for building cross-platform targets (although not together in the same build tree). This meant that the binaries required for all three target architectures in the case study could be built using this build system.

2.3 Choice of Kernel

One of the primary attributes of the operating system coming out of the case study was stability. This helped focus the selection of kernels to those which were already tried-and-tested by the manufacturers of the individual SBCs, and also the wider Linux community. Upon investigating the reference Linux kernels available for each of the SBCs chosen, it turned out that each had a reference Linux kernel available, and that the version of each kernel was quite close. This fitted in with the author's goals; to use a tried-and-tested kernel to reduce the amount of work in getting three new kernels working with so many other unknowns (boot loader, init process, etc), and have a kernel close enough on the three boards as to make the inclusion of additional features and drivers into those kernels as straightforward as possible.

Table 1:   Kernel Version by Single Board Computer

Board	Architecture	Kernel Version
Routerboard 532	MIPS	2.4.31
Gateworks	XSCALE	2.4.27
Wrap 2C	X86	2.4.31

2.4 Wireless Driver Selection

Until 2006, there were two main sets of drivers for wireless cards, those based on the Hermes chipset, and those based on the Prism chipset. A third chipset emerged in 2003 from Atheros Communications. While the licensing of the Hardware Abstraction Layer (HAL) was available to commercial organisations for a fee, the open source community was left lagging behind in support for this chipset. The open-source community then started the MADWIFI (Multimode Atheros Drivers for WIFI) project. The author has been following the development of these drivers since 2004, and has attempted to use these drivers at several stages. Unfortunately these drivers turned out to be far from usable for application in environments where long distances were used, so they were never used in earnest by the author until early 2006. The timing was such that the first official releases of the MADWIFI drivers occurred around the same time as the proposal for this case study was being developed, at which time research indicated that several of the blocking issues had been resolved in the driver, finally making it a possible candidate for use in community wireless networks.

Initial research into the stability of these drivers was carried out, as well as investigation into the drivers operation on links greater than 50 metres. This investigation showed that the drivers, even though there were still issues, functioned on links of approximately 2km. These tests indicated that the MADWIFI drivers were suitable for inclusion in the case study. The inclusion of the drivers on all three architectures was also interesting in that the process of compiling the drivers for use on the three architectures only required minor tweaking into order for them to be built and used.

Investigation was also carried out into the possibility of using some of the older chipset cards, but while the drivers were available and compiled fine on the three platforms, the author did not have the hardware available to test these in operation. Also, since using the Hermes based chipsets at distance (greater than 12km) can be problematic, the author decided to focus exclusively on the use of the MADWIFI drivers for the case study, since the support for long-distance links was part of its built-in functionality.

2.5 Boot Loaders

Since all three boards were supplied with boot loaders by the manufacturers, the main portion of the work involved with this functionality was to investigate how to get the kernel loaded and booting on each of the SCBs. This is covered in more detail in Chapter 3.

2.6 Web Front-end

The main research that went into this area was to find out which web service and programming language to use. This was to satisfy a flexible development environment for the GUI (Graphical User Interface) developer, and a small memory footprint when running on each SBC. While the decision of the web server to use was straightforward, as a small, lightweight daemon (thttpd) was included in the buildroot system, the choice of language in which to write the scripts for the back end functionality was more difficult. There were many choices that could have been used in this case, each with their own

advantages and disadvantages. In the end, it came down to three choices, shell script, Perl or PHP. Shell script had the advantage of having a very small memory footprint, but limited functionality. This was ruled out pretty early, as the author did not want to be limited by the functionality and felt that the modern SBCs could bear the larger memory footprint of Perl or PHP. Perl and PHP used quite a large amount of memory, but had a large set of functionality available to the programmer. As part of the research, the author built a command-line version of PHP for use with thttpd, and also a microPerl binary. Each had enough functionality to satisfy the requirements for the web-front end, and since the microPerl binary (~1Mb) was one third the size of the PHP command-line binary (~3Mb), Perl was chosen as the language with which to develop the front end. On all other criteria of selection, Perl and PHP seemed very similar; functionality, availability of external libraries/functions, availability of expertise and ease of development. One small other reason was involved; the author had more experience in PHP, and wished to learn more about Perl. Apart from that small reason, memory footprint was the main deciding factor between Perl and PHP.

2.7 Resource requirements

A mixture of hardware and software requirements was needed. The hardware selected was based on the most commonly used hardware in the local community wireless network of the author. In this network, there is a varied selection of PCs, single board computers, and custom wireless networking hardware, so the author selected three of the most commonly used boards. The full list of requirements is as follows:

- At least one of each Single board computer of each type:

    o   PC Engines WRAP2C (x86)   (www.pcengines.ch)

    o   Mikrotik Routerboard 532 (mips)   (www.mikrotik.com)

    o   Gateworks Avila 2348-4 (xscale)   (www.gateworks.com)

- Development platform to build kernel for each board.

- Buildroot system to build common root filesystem.

- Source code control system.

- Capability for writing image onto Compact Flash cards.

- Several Atheros wireless cards (802.11a/b/g).

- Various Antennae.

Figure 1 The three target platform boards

The author acquired one of each of the three target single board computers, which

can be seen in the photograph above, with the WRAP2C on the left, the Routerboard 532

in the centre, and the Gateworks Avila on the right. The Compact flash card can be seen

on the centre board only, as the socket is on the underneath on the other two boards. Also

shown on the centre board is a CM9 miniPCI wireless radio card. The empty miniPCI

sockets can also be seen on the other boards.

Because this is an open-source project, all notes are kept in a Wiki website at

www.me2000.net to help others interested in the project to keep up with changes and

modifications. It is also an ideal way for these other users to report bugs, and request new

features. One disadvantage of Wiki's is the amount of spam attacks aimed at these types

of project. There were several dozen attacks during 2007, resulting in the author to be

forced to configure the Wiki to be read-only. Investigation will have to be done in the

future into how to make a more secure community portal for the project.

3 Case Study Implementation

3.1 Linux Kernel Selection

The choice of kernel is also important, as a stable, well-supported kernel is required. A 2.4.x Linux kernel was selected, as there is existing support for each of the chosen boards by each of the board manufacturers for this kernel. For some of the selected Single Board Computers, this selection would be dictated by the latest version of the kernel supported by the board manufacturers. In some cases, a kernel was available from the board manufacturer already patched with the board-specific hardware drivers. The Kernel version selection for each SBC is shown in Table 2 below.

Table 2:   Kernel Version by Single Board Computer

Board	Architecture	Kernel Version
Routerboard 532	MIPS	2.4.31
Gateworks	XSCALE	2.4.27
Wrap 2C	X86	2.4.31

The versions were kept as close together as possible, as this would make building common applications across all the platforms easier.

3.2 Tool Chain Selection

A tool-chain was needed for the project that allowed development of applications for all three architectures. While investigating Busybox, the multi-call binary that is used

in many embedded systems, it was shown to be able to emulate a basic init process, which could possibly be a good basis for an operating system. A sister website of the Busybox system was the Buildroot system which has a mechanism for compiling toolchains for various target platforms based on simple menu selections.
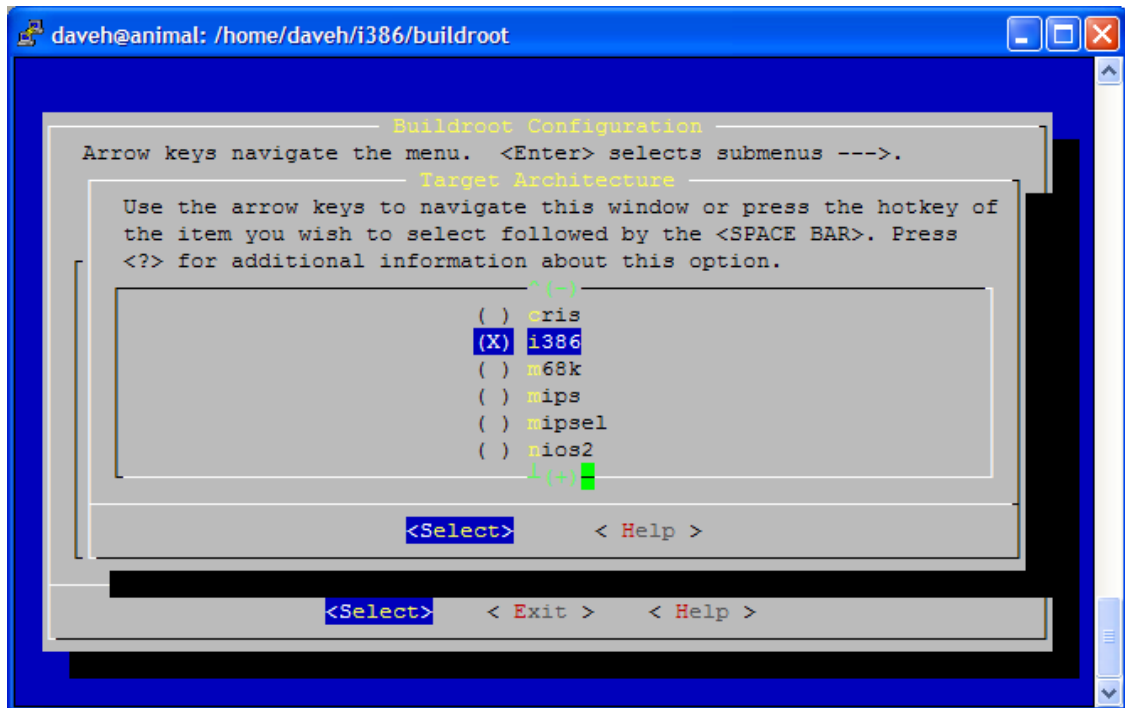
Figure 2 Selection of architecture using the Buildroot System

Investigation was carried out into building a rudimentary rootfs filesystem which was loaded by the kernel and the init process started. Because the busybox multi-call binary uses the buildroot system, which had been enhanced to provide an initial ramdisk and init process, it seemed the ideal candidate on which to base the project. Another major advantage of the Buildroot System it its ability to cover a wide variety of CPU architectures.

One minor inconvenience is that the Buildroot system is designed so as to provide a build toolchain for a particular architecture, which meant creating a separate Buildroot environment for each of the three boards in the case study. This meant that there was a greater chance of a mismatch in the environment settings allowing an error of mismatched functionality on the three platforms. The kernel configuration for each platform was also separate, so care was needed to ensure that the same configuration settings were used for each kernel build. For example, there are several configuration changes required in order to have full functionality built into the kernel to support the IPTables firewall. These configuration changes to the kernel had to be made on all three kernels so that the functionality was consistent across all three boards.

3.3 Boot Loaders

Because the boot loader is different on each SBC (as shown in Table 3), research was needed to look into getting the kernels and root filesystems loaded into each board in a consistent manner. This was achieved by partitioning three compact flash cards in such a way as to facilitate easiest integration into the pre-existing default boot loaders on each SBC.

Table 3:   Boot Loader by Single Board Computer

SBC	Boot Loader
Mikrotik RouterBoard 532	RouterBoot
Gateworks Avila 2348-4	RedBoot
PCEngines WRAP2c	Grub

This gives an overview of the steps needed to get the commonly configured kernel and filesystem to boot on each platform. Each board has a Compact Flash socket, and it was possible on each board to configure it to boot a Compact Flash card when present in that socket.

Figure 3 Compact Flash Card

The intention was to get a common filesystem to all platforms once the system was booted. This consists of a root file system mounted as a Ram Disk, and partition on the Compact Flash card mounted as /boot (for persistent storage). In most cases, the kernel file is called zImage, and the compressed root file system is called rootfs.gz, a gnuzipped image of the rootfs. In all cases a 32Mb Compact Flash card was used.

Figure 4 Compact Flash Adapter

The Compact Flash was prepared using a Linux laptop, where the Compact Flash card was inserted into the PCMCIA slot using a PCMCIA to Compact Flash adapter. This allows us to partition the card make file systems on it, mount the file systems, and copy files onto it. We can also write data into specific sectors, as is required on the RouterBoard platform.

For the boot loader configurations outlined below, more detailed instructions are available for the commands used on each board, with the configuration of the different bootloaders. See www.me2000.net for more information.

3.3.1 *RedBoot (Gateworks Avila)*

RedBoot [8] is quite a sophisticated boot loader, in that there are many features to facilitate the developer in booting from a variety of sources, be it a TFTP server, the internal flash memory, or a Compact Flash card. For most of the duration of this project, the TFTP server was chosen, as it was convenient to build the kernel or root file system, and re-boot the target board, which would then pick up the latest kernel image or root file

system image. In the case of the Compact Flash card, a simple RedBoot script was saved

in the SBC's NVRAM, which tells RedBoot to load two files into memory and jump to

the start point of one of them (the kernel). The following paragraphs describe how the

kernel and root file system were placed on the Compact Flash card to facilitate loading by

the RedBoot loader.

The Compact Flash card was initially partitioned with one partition, using a laptop

running Linux, and the Compact Flash card inserted in a CompactFlash to PCMCIA

adapter, and formatted with an e2fs filesystem. Once the filesystem is created, we can

now mount the filesystem and copy on the kernel and the compressed root filesystem.

As supplied, the Gateworks Avila 2348-4 does not contain an operating system. We

need to specify a configuration for RedBoot so that it will load the correct boot image

into memory and start it. Redboot is aware of ext2 filesystems, so we can simply tell

redboot to load the zImage file into one location in memory, load the rootfs.gz file into

another section of memory, then jump to the start address of the kernel. This is achieved

by storing a 3-line boot script in the boards non-volatile storage, which tells Redboot

where to load the kernel and root filesystem from, and then where to jump to start

executing the kernel.

3.3.2 RouterBoot (Routerboard)

The router board comes with the RouterOS operating system installed, so the settings on the board need to be changed to force booting from the new operating system on the Compact Flash.

Preparation of the compact flash card is slightly different to the Avila, in that the RouterBoot boot loader loads the first partition it finds into memory, and jumps to its start address. This bootloader also requires that the compressed root filesystem is embedded into the kernel image as an extended elf module. This is achieved by using the "objcopy" tool as part of the script that builds the zImage file for copying onto the first partition of the compact flash card.

The default configuration of the RouterBoard is to boot the on-board operating system, RouterOS. We need to change this behavior so as to boot the operating system on the Compact Flash card. This is achieved by interrupting the boot process with a serial cable and terminal program, and changing the default boot device of the board.

3.3.3 Grub (WRAP)

The Wrap board is probably the most straightforward of all three boards in the case study, as it is x86 based, and boots by default from the compact flash card, so no boot parameters needed to be changed in the bios of the SBC. The WRAP board contains a very simple bootstrap loader, which allows us to install a slightly more sophisticated bootloader to load the Linux kernel and root filesystem. There are many boot loaders

available for x86 based systems, but one of the most popular is the GRUB bootloader. This was chosen simply because of the author's experience with it in the past as a simple, no-nonsense bootloader for use with PCs and x86 based SBCs.

In this case, a single ext2 filesystem was created on the compact flash card, and then the partition had the grub bootloader installed using the 'grub-install' tool. This created the MBR (master boot record) on the compact flash, and installed the default scripts for booting the system. After some slight modifications of these scripts to default the console to the serial port (there is no graphics output on a WRAP board), the root filesystem and Linux kernel were copied on, and the system booted.

3.4 Operating System Framework

Once all three boards were booted up to the stage of successfully running the init process, the operating system framework could now be developed. This included the basic scripts to mount the filesystems and start up the critical daemons that are necessary for the basic operation of the system. Since all three systems were now at a common stage, with an init process starting, the scripts that were to be developed could be used across all three platforms without modification.

One of the more critical scripts is the rcS script, which loops through the contents of the /etc/init.d directory and executes each script with a 'start' parameter. All the scripts in this directory are written to start a service or daemon, or to load some operating system drivers, such as the Madwifi drivers for the 802.11b radio cards. The order of the script

execution is based on the 3 characters at the start of the filename with S00 scripts being executed first, and S99 scripts being executed last, and all scripts in between being executed in numerical order. This is common with many Linux distributions. The initial rcS scripts is specified in the inittab file, which gives the init process a list of tasks to carry out on initial boot. These tasks include mounting the filesystems, remounting the root filesystem as read/write, eventually executing the rcS script which starts all the daemons, and finally starting a 'getty' process on the serial port to allow console logins. These scripts were common across all three boards.

3.5 Wireless Radio Selection

The Atheros based radio cards offer most of the functionality needed, and have driver support for Linux in the form of the Multimode Atheros Drivers (MADWIFI). As mentioned earlier, the first official release of the MADWIFI drivers happened about the time that the case study was started, but during the testing phase, there were still some bugs to be addressed. It was necessary for the author to investigate a particular bug so that the drivers would allow the radio card to associate with the existing access points in the locality. This bug caused the MADWIFI drivers to be unable to associate with any access points based on the Hermes chipset (originally manufactured by Lucent Technologies). The author discovered an open issue on the MADWIFI issue tracker database [25], and after several weeks, one of the driver developers submitted a patch that resolved the issue. The patch was applied to the case study version of the drivers, and testing could progress. This patch was included in the next official release of the drivers.

3.6 Package Integration

For the purpose of the case study, the following packages were compiled and added to the root filesystem, along with scripts to start any daemons necessary.

- MicroPerl – A cut down version of the Perl package. Includes only the perl binary, no Perl modules were included to save space.

- Thttpd – a small http daemon.

- IPTraf, TCPdump and iftop – Network utilities for sniffing and debugging network traffic.

- OpenSSH for secure logins.

- Net-SNMP for statistics gathering by remote hosts.

- DHCP for easy configuration of clients.

- IpTraf and TCPDump for network debugging.

- IPTables firewalling.

- Quagga routing daemon for various routing protocols.

Integration of most of the above packages required modifications to the build scripts in order for them to be built under the three environments for the various target architectures. The MADWIFI drivers, for example, needed to be modified to use the buildroot binaries in places, as the makefiles were hardcoded to use the operating systems

binaries (on the path) rather then the cross-compiling versions for the particular CPU architecture. Once a build was successful, the author produced a patch by comparing the original source with the modified source. By then placing this patch in the relevant directory, the package could then be downloaded in its original form from the internet, patched, and built without errors under the Buildroot system. These patches were short-lived, as the next version of the MADWIFI source had already had these patches applied in the source, negating the need to patch during build.

3.7 Configuration Front-end

The web front-end was developed to implement the basic web-based user interface, allowing the user to set the basic settings needed to get the system on a wireless or wired network, and functioning as a router. The areas covered by the front end are:

- TCP/IP settings,

- Wireless settings (for Atheros cards)

- Static routing table modification

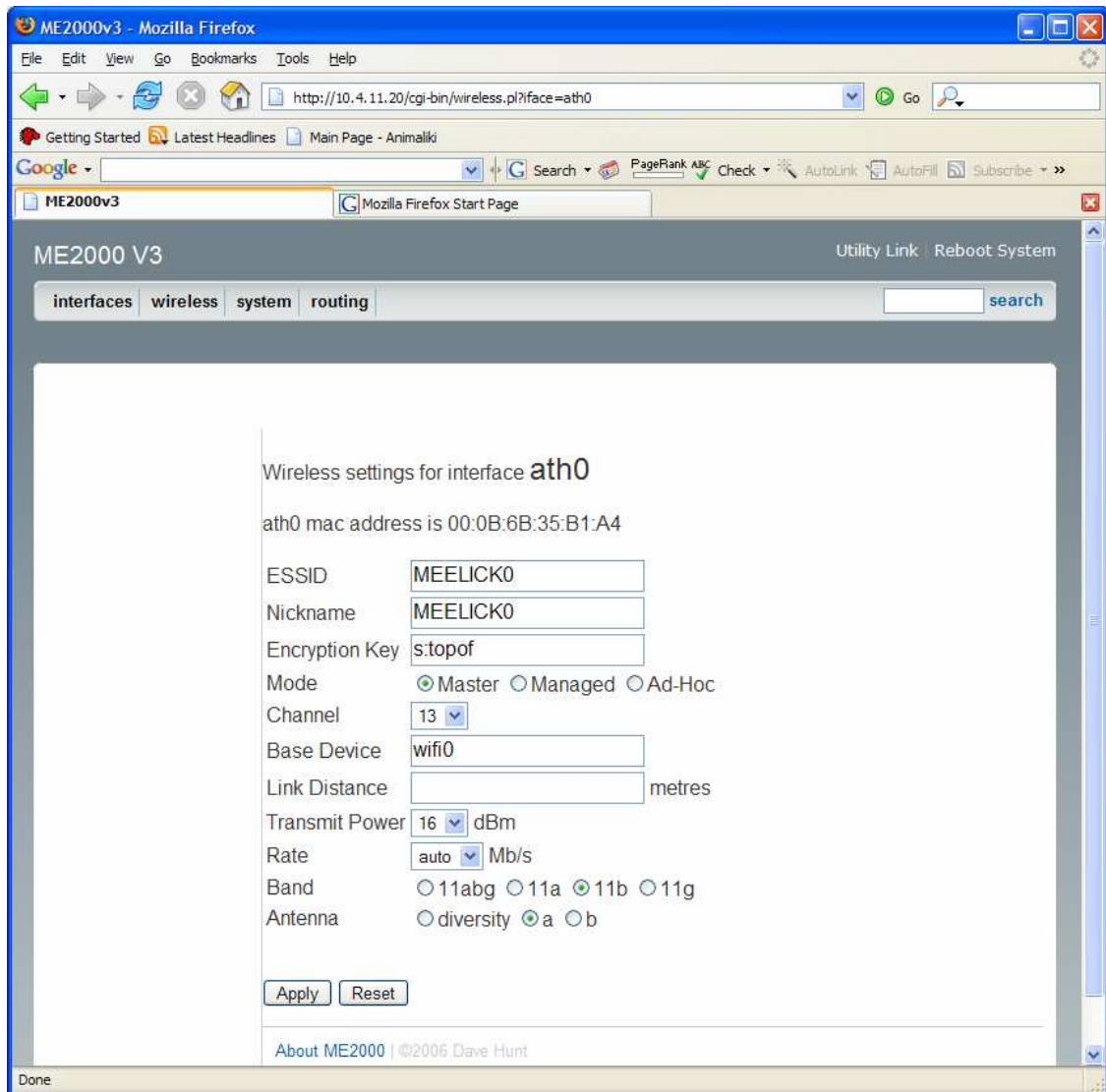- System information, such as log files.

Figure 5 Web front-end screenshot

The screenshot above shows the basic layout of the type of user interface used.

Upon the user selection the relevant section from the menu, they are then presented with

a series of input gadgets. The fields are pre-filled with the current settings, allowing easy

viewing of existing settings before a change is made. The interface makes use of some of

the more advanced features available in the MADWIFI drivers for the wireless settings,

such as transmit power and distance settings, allowing links to be tuned to be optimal for

a particular link distance between routers. This feature had been unstable but this was addressed in early 2006, making the MADWIFI drivers finally usable on long distance outdoor links. Appendix A contains screen shots and descriptions of the front-end functionality developed as part of the case-study.

The Web interface proved to require the most intensive development, and also proved to be very portable across all three platforms. This is due to the fact that the same version of the web server and perl binary was used across all platforms in the case study. Functionality developed on one platform could be seamlessly copied across all three boards.

3.8 Remote Upgrade Functionality

Upgrading the Kernel and root file system is straightforward for all platforms, as the common file system was established; allowing common scripts to copy the images onto the local file system of the router, comparing md5 checksums to ensure image integrity. Common settings are maintained across upgrades, due to method of storage of particular files onto persistent storage, before reboot, and restoring of those settings over the defaults at a particular stage in the boot process. This feature was used several times during the in-service usage period to upgrade the software, and worked well. At no time was it required to physically access the SBC. As the amount of upgrades goes up in extended roll-out, then the chance of an interruption in the process increases, thereby increasing the chance of a corruption on the operating system.

One of the potential problem areas is just after the md5 checksum is performed on the new image, as it is being written to the compact flash card in preparation for the reboot. If there is a problem in writing the image to compact flash, in some instances this can be trapped and some problems circumvented, but in the case of a power loss during the image write, physical access to the card will be needed to restore it to a functioning condition. This has been seen by the author on a number of occasions over the last 5 years where interruptions in upgrading software or saving settings has caused the operating system to stop responding after a reboot, necessitating a time-consuming visit to a remote site to recover the system.

It is imperative that effort is focused on this particular area to take the project into the future. Every function call, every write to flash, every file transfer and every configuration change must have its return code checked to ensure that no corrupt data is being written that might cause the system to become unresponsive after a re-boot.

The remote upgrade facility was developed based on the *wget* utility, which allows the scripted downloading of files from a remote web server. This remote web server is a repository of operating system images and is similar for all three platforms. Along with each image, there is an associated file containing the md5 checksum of the image. This is set up as part of the release process of each image. When the script to update the firmware on the SBC is called, two files are downloaded from the server - the image file, and the md5 checksum file. When the download of these files are complete, the md5 checksum is re-calculated, and compared against the md5 checksum in the downloaded checksum file. If these two checksums compare favourably, then there is an extremely

high probability that the downloaded image is the same as the one on the remote server. It is only after these checks are complete that the operating system image is written to the flash of system. The settings file is left unaffected so that the same settings are used after rebooting, and connectivity will be restored, eliminating the need for local access to the system after reboot.

One area for improvement in the future is to introduce a patching mechanism that would involve extracting the existing settings where necessary, patch the relevant files with the extra configuration settings required for any applications that need new settings, and re-packaging those settings suitable for loading by the operating system when a reboot is requested. The current scheme will suffice for most upgrades, and has served the author well for the duration of the development of the case study. The script for updating the flash memory on the Routerboard is shown in Appendix B.

3.9 Persistent Configuration Storage

The persistent storage of the configuration settings was straightforward once the common partition (/boot) was available on all three platforms. The persistent storage currently consists of creating a tar file of the contents of the /etc directory, along with the password file for the thttpd daemon (which is stored elsewhere). The tar file is created in /tmp, and once created, is copied onto the /boot partition on the compact flash storage.

At boot time, the default /etc directory is extracted from the rootfs.gz file, the init process is started, and one of the very first scripts to be executed in the /etc/init.d

directory is the script to extract the /boot/settings.tar file on top of the default /etc, overwriting the default files with the files modified by the user interface or manually edited by the user. The script that creates this settings.tar file is called savesettings.sh, and may be called directly on the command line, or by clicking the relevant button on the web front end.

3.10   In-service Usage

For the purposes of testing the system in a live environment, a full wireless node was constructed. This required the following hardware (photographs below):

- WRAP2C single board computer

- Two Atheros wireless cards

- Waterproof box (IP56 rating)

- Mounting brackets

- Two pigtails for connecting antennae to wireless cards

- 2.4 GHz directional antenna

- 2.4 GHz omni-directional antenna

- 3metre steel pole

- Chimney lashing kit

- Cat 5 cable from SBC to inside residence.

- Power-Over-Ethernet (POE) kit

- 18v power supply

This equipment was assembled into a functioning wireless node, and installed on the authors' residence in April 2007. The equipment was configured to connect into the existing wireless network in the locality, a community broadband network which has been in existence since 2001. The case-study operating system was loaded onto the SBC and put into commission for testing. Some live network traffic was re-directed to go through this node for long-term testing purposes.

Figure 6 Live installation example

This has turned out to be rather successful, and has had very little downtime since installation in April 2007. The only downtime (in the order of minutes, rather than hours) was due to configuration changes and rebooting. Statistics are being gathered by the central server and graphed using MRTG [20]. These statistics include both throughput graphs and wireless SNR graphs. SNR graphs via SNMP are commonly problematic with commercial wireless routing operating systems such as Staros [15] and RouterOS [4], as

the wireless signal levels are often not exposed by the SNMP daemon. The open-source advantage is obvious here, as the author was easily able to extent the existing SNMP functionality to include these statistics.

The author also looked into porting the operating system to a standard PC, and with very little modification was able to boot the system on an old Pentium II Dell Optiplex desktop system. This PC was capable of running a CM9 radio card, and has been installed in the attic of a neighbour's residence, providing them with broadband since August 2006. Its current uptime is 79 days, with the last reboot being caused by an electricity supply outage in the area.

4   Case Study Evaluation

4.1 *Summary of the project*

The author succeeded in developing an operating system based on the Busybox/Buildroot systems, across three CPU architectures. These three operating systems had practically all of the packages described in the initial expectations, and could be easily configured via the web interface, and have those changes saved in persistent storage. Several SBCs in the authors section of a community wireless network were upgraded to use this operating system, and have given very good service since they were installed.

While the author believes that the case study was successful in meeting the initial project expectations of developing an open-source, cross platform operating system for three different CPU architectures, the future of the project could go several directions.

1. Take the existing scripts and current snapshot of the makefiles and build scripts and store them in a repository, allowing anyone to check them out and build a particular snapshot of the operating system.

2. Approach the Buildroot team and suggest enhancements to their existing structure to include extensions to fill in the gaps to becoming a fully bootable ram file system, using hardware specific scripts dependent on the menu selections made by the user at build time.

Of the above two options, the author feels that the first option is the easier, because it is under the control of the few developers who have the most need to develop a cross-platform wireless routing operating system. However, at the same time, the author feels that the second option is the better one, as there is a wide community of developers already involved in the Buildroot system, and it seems to be so close to the full bootable file system that the development team may be open to suggestions in that direction.

4.2 *In-service Usage*

The in-service usage has proved very successful, with very little downtime since the system was installed in April 2007. The software has proved to be very stable, providing service as long as power was applied. In fact, power was probably the one place where service suffered, in that there were several days where the electricity supplier brought down the power supply for essential maintenance in the area. This could easily be circumvented by using some low power uninterruptible power supplies on the ADSL modem and the SBC on the author's residence, as these pieces of equipment draw very lop power (under 15 watts each), and would last for several hours on a standard 20 minute 200 watt UPS.

As far as the software is concerned, no reboots were needed due to processes hanging, or due to lost connectivity.

4.3 Included Software Packages

There were slight changes in the initial package list as planned at the start of implementation. It was planned to use the OpenSSH package, but it's resulting size was considered to be too much compared to the dropbear implementation of what turns out to be very similar functionality, an ssh client and server. Dropbear is roughly 145K, where ssh is roughly 700K (including the OpenSSL library). Another package that was not implemented on all three platforms was IPTraf, which is a curses-based IP traffic monitoring tool. While it compiled without problems under the buildroot toolchain, it failed to function as expected on the Gateworks Avila platform and the Routerboard platform. Initial (brief) investigation seems to point at some non-standard keyboard handling routines that only function correctly on the X86 platform.

Apart from the issues with the packages mentioned above, very few problems were encountered when building the following packages for all three boards.

- Madwifi wireless drivers for Atheros Wireless Devices.

- Web server for configuration front-end.

- SNMP for statistics gathering by remote hosts.

- DHCP for easy configuration of clients.

- IPTables firewalling.

- Quagga routing daemon for various routing protocols.

4.4 Advantages of open-source over commercial

4.4.1 Adaptability

In recent years, when implementing wireless networks as part of community broadband schemes, the author has encountered problems when trying to monitor statistics of some commercially available operating systems, particularly when it comes to signal levels of the radio cards. For example, StarOS will show the Signal Levels of associated clients on the text-based UI, but does not make that data available via SNMP. Using an open-source solution allows developers to extend the existing functionality of the SNMP daemon to provide whatever pieces of information they want to monitor. In the implementation of the case study for this capstone paper, the SNMP daemon was extended to provide wireless signal levels and signal-to-noise ratio to an authorized SNMP client. The resulting graphs of a 24-hour period can be seen in Figure 7 below.
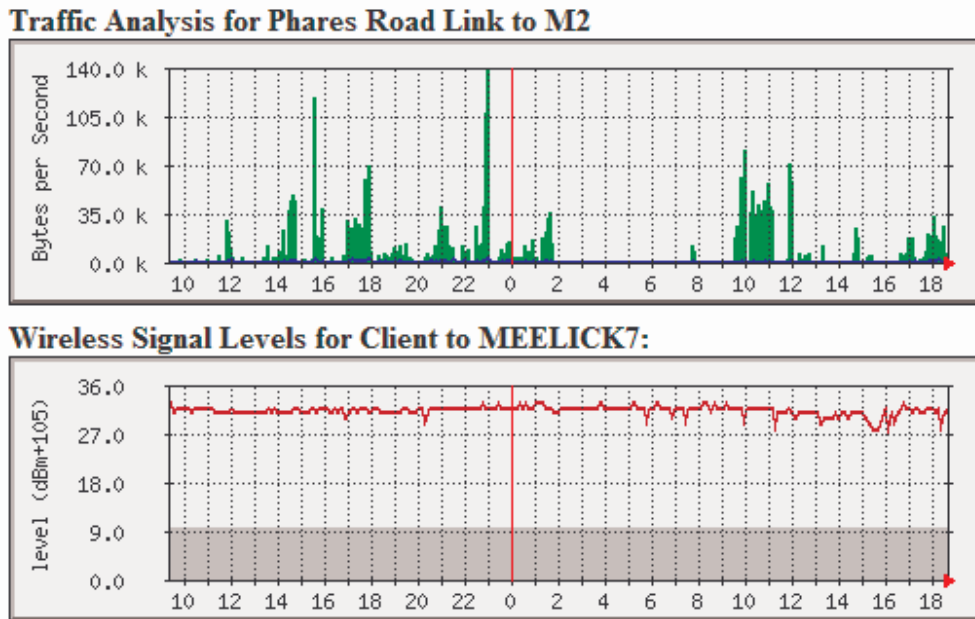
Figure 7 Graphs of Traffic and Wireless Signal Levels

The advantage of being able to modify an operating system to provide important statistics is critical in areas where these statistics can be used to predict failure or downtime, or in areas where the data will help analyse bottlenecks or problem areas. At the time of writing, signal levels were not available via SNMP from two of the most popular commercial wireless router operating systems.

4.4.2 Security

The Verisign website contains details [16] of a vulnerability which was discovered in StarOS [15] in 2003. This vulnerability is believed to be in the StarOS Operating system since its initial development. This vulnerability would probably have been discovered much sooner (possibly years sooner) had the source been open to peer review.

With open source software, the code can be peer reviewed, and any vulnerabilities discovered can be addressed. This, of course, is provided someone goes to the trouble of peer reviewing or looking for vulnerabilities, and issue which is covered in the following section entitled "Support".

Hopeman et. al. [14] concludes: "In the long run, openness of the source will increase its security". While there may be an initial period of increased vulnerability due to the immaturity of a given system, this will be replaced by increased security as the system matures. They go on to say: "Open source allows users to make a more informed choice about the security of a system, based on their own or on independent judgment". This is a very relevant area to the reasons behind implementing this case study. A wireless routing platform needs to be secure, to prevent unauthorized access to the data passing through it. It is my conviction that an open-source solution will offer increased security over a closed-source solution, or at the very least leave it open for improvement by whoever sees the need to improve it.

4.4.3 Suitability for target board

Some commercial operating systems [15][4] are designed with the same image to be run on several different types of hardware platform, from a small SBC with limited memory to a more powerful board with plenty of resources. This may mean that the less powerful board is not performing as well as it should because of unnecessary software running on the board using valuable resources. An open source solution gives the flexibility to completely disable un-needed daemons, or even add functionality into the

user interface to disable them. If the target board is very limited in resources, entire

packages can be left off the Compact Flash image, reducing the footprint further.

4.5 Advantages of commercial over open-source

One of the main advantages of commercial solutions over their open-source

equivalent is the issue of support. In the 5 years in which the author has been involved in

open source software for wireless community projects, there is very little in the way of

submitted enhancements or bug fixes for the software that has been produced by the

community. This is probably down to several factors.

1. The narrow scope of the software. Because the community of network builders is
   small, there are not a great number of developers available that are capable of
   contributing to the project.

2. Many communities feel the need to develop their own hand-rolled solution, or
   even simply accept the limitations of existing commercial solutions. It is quite
   straightforward to build a small x86 version of Linux that is portable across many
   x86-based PC platforms.

3. The infrastructure of the project. Work still needs to be done on making the build
   system of this particular project easy to check out and get started with. Currently,
   the scripts that were developed as part of the case study are stored in a separate
   repository to the central Buildroot system, and there is no easy way of
   synchronising them. This may discourage new developers from partaking in the

project. It needs to be very simple to check out all the code, and start building the system, ready to make enhancements or fixes.

4.6 Case Study Testing Phase

In order to prove the viability of this new operating system, the author undertook several tests to ensure that the functionality was acceptable from various points of view. Probably the most important aspect was the ability of the OS to route traffic wirelessly at a speed comparable to the commercial operating systems. Upon setting up the initial test rig, the bug discussed in section 3.8 was discovered, which did not allow association with Hermes chipset access points. This was a blocking issue, as most of the access points in the locality used this type of chipset. Thankfully, a solution was found by the MADWIFI developers, and testing could proceed. Once associations could take place, the author found that the speed was comparable to the commercial operating systems at 2.4 GHz (802.11b), giving approximately 700KB/sec transfer rates. The drivers also compared favourably at 5.8GHz (802.11a), giving approximately 2.5MB/sec transfer rates. Unfortunately, the drivers could not compete with the proprietary modes available in some commercial operating systems such as the NSTREME mode available in RouterOS, which uses two antennas at each end to give a full duplex connection, dramatically increasing the throughput. The author believes that it is unlikely that the open-source drivers will see these accelerated modes in the near future, as these protocols are closed-source.

The author also used the case study operating system in a live environment (discussed in section 3.10), which made an SBC available for broadband access on a

segment of the local community broadband scheme. Due to time constraints in the case study, only limited testing was carried out on this installation, but the uptime was very good, and did not give any problems, even with several remote users logging in to view the graphical user interface from time to time. There are still many features that were added onto the installation such as automatic routing and firewall functionality, and while these features are enabled, lack of time limited the amount of testing that could be performed on them.

There was also an earlier version of the case study operating system which had a more limited set of packages, but this was in a heavy traffic environment routing traffic between two residences on the local network backbone. This installation has given no problems since it was installed in late 2006, and shows that the basic operating system and wireless drivers are certainly stable enough for use in a community broadband wireless environment.

5 Conclusions

The primary goal of the case study was to show that it was possible to develop a cross-platform open-source wireless router operating system. The author believes that this has been successfully achieved, and in the following sections will discuss the conclusions drawn from the case study implementation.

5.1 Case Study Benefits

One of the most interesting conclusions drawn from the development of the case study is how close the open-source community is to having a menu driven, cross-platform operating system generator. The current Buildroot system goes as far as building the applications with the relevant toolchain and placing them in a root file system directory structure, ready to be built into a file system image used to boot into the operating system. What that Buildroot system does not contain is an agreed set of scripts with which to boot the operating system, or a set of device nodes in the /dev directory. The Buildroot system could be extended to fill these gaps, and the users would end up with a ready made operating system for any of the supported hardware platforms.

This Buildroot "operating system generator" would seem to be of great benefit to the academic community, where students choose to implement some functionality on embedded systems. The Buildroot system is very close to give the student a ready-to-go development system in which the basic operating system can be built with relatively little effort, allowing them to focus on the task at hand, such as adding external sensors,

monitoring software, etc., or whatever the student has selected as their embedded system project.

5.2 Limitations of the Case Study

The author feels that the case study was a good choice for the investigation into the viability of developing an open-source cross-platform wireless router operating system. Having a live installation in a real-world environment was very satisfying. However, it may have been advantageous to get more installations live by getting more people involved in the testing.

Focussing on one particular wireless card is a cause for concern. This limits the operating system to systems that are capable of hosting that particular radio card. Many older PCs cannot host the particular radio card used (CM9), so this rules out the use of many old PCs that people may wish to recycle by using as a router in their attic. Using a Wiki for the notes is a potential risk, as the author has lost count of the number of times that it has been defaced. Luckily, no information was deleted, just links to external sites added. This is a problem in that if a page was changed slightly, it is possible that the administrators may not notice. The Wiki may be replaced in the future with something more secure. More research will be carried out in that area.

5.3 Further developments

The author plans to use the case study as the basis of the community network in a small part of rural Ireland where traditional ADSL is not available. The first step has been taken, with one node live (at the author's home), but this will be extended rapidly from September 2007 onwards. Many of the 30 nodes in the network in question use a mix of operating systems, including an old open-source implementation that the author was involved in. The nodes with the older open-source versions will be replaced first, as it has been seen from the case study that the new software has many advantages, especially the web front-end for configuration.

It is debatable that existing sites with commercial operating systems will have their software replaced. Most of these sites are functioning adequately, and the author feels that it may cause problems to significantly change something that currently does not have any reliability issues. Sites that are more easily accessible are more likely candidates to be upgraded, and based on the performance of these nodes, a decision will then be taken as to whether to upgrade the more inaccessible nodes. This period will probably be of the order of 6 months.

In the near future, support for a wider variety of hardware will be added, such as Ethernet cards, wireless radio cards, etc. Many older PCs cannot boot with CM9 radio cards due to hardware incompatibilities. Extending the driver support will allow these PCs to use alternative radio cards, allowing them to be recycled more easily by using them as routers in people's attics.

5.4 Validity of Thesis Statement

The author stated in the thesis statement of this paper that it should be possible to develop an open-source cross-platform wireless router operating system with specific criteria. The author feels that those criteria have been met, in that a fully functioning operating system has been produced for the three target boards, and is currently installed in live installations and is of benefit to the wireless community in which the author is involved.

There were some problems encountered, such as the inability of the first release of the wireless drivers to associate with most of the access points in the locality, but this was eventually addressed by the driver developers. This raises an interesting question: had these drivers been commercially developed, would a solution have been released any quicker? The author believes that this is very difficult to quantify, as it depends a lot on the ability of the developers to react and produce quality solutions regardless on whether they are commercially driven or open-source advocates.

In order to gain most benefit out of the case study, it seems to the author that the best way to proceed is to feed back the knowledge gained back into the Buildroot community, and help get that system to a stage where an operating system can be build for a target platform with the minimum of effort. This would probably be far more advantageous than attempting to maintain a branch of that system specifically for the

local wireless community, where the knowledge gained may well be lost. The wider the range of developers involved in the project, the more likely it is to survive into the future.

To conclude, the author feels that the case study was a success, the thesis statement holds up, and also raises some interesting questions. What to do next? Is open-source 'better' than closed-source? How do we quantify 'better'? Some of these questions will undoubtedly keep the software community busy for many years to come.

References

[1] Buildroot Embedded Linux Website - http://buildroot.uclibc.org/ - Accessed 25th September 25, 2006. No published date.

[2] BusyBox: The Swiss Army Knife of Embedded Linux http://www.busybox.net- Accessed 25th September 25, 2006. No published date.

[3] Gateworks Corporation Website - http://www.gateworks.com/- Accessed 25th September 25, 2006. No published date.

[4] Mikrotik Website - http://www.mikrotik.com/- Accessed 25th September 25, 2006. No published date.

[5] PC Engines - Embedded PC Design - http://www.pcengines.ch/- Accessed 25th September 25, 2006. No published date.

[6] ME2000 Wireless Router Node System - http://www.me2000.net- Accessed 25th September 25, 2006. No published date.

[7] Vonk. C.J.S. (2005) - Secure Internet Appliance for Small Office / Home Office HOWTO – Retrieved on 19th April 2006 from http://users.gotsky.com/cvonk/linux/siso/

[8] Redboot Website - RedBoot Debug and Bootstrap Firmware – Accessed 18th March 2007 from http://www.ecoscentric.com/ecos/redboot.shtml

[9] thttpd website - tiny/turbo/throttling HTTP server - Accessed 18th March 2007 from http://www.acme.com/software/thttpd/

[10] IrishWAN community website – Accessed 18th March 2007 from http://www.irishwan.ie

[11] GNU GRUB – GRand Unified Bootooader website - Accessed 7th May 2007 from http://www.gnu.org/software/grub/

[12] Hoepman, J. and Jacobs, B. 2007. Increased security through open source. Commun. ACM 50, 1 (Jan. 2007), 79-83.

[13] XORP project website – Open Source IP Router - Accessed 7th May 2007 at http://www.xorp.org

[14] Quagga project website – Quagga Routing Software Suite, GPL licensed IPv4/IPv6 routing software. - Accessed 7th May 2007 at http://www.quagga.net/

[15] Valemount Website – StarOS Operating System – Accessed on 10th May 2007 at http://www.staros.com/

[16] VeriSign Website – *StarOS Static Provate Key Usage Vulnerability* – Accessed on 10th May 2007 at http://www.verisign.com/security-intelligence-service/current-intelligence/vulnerability-advisories/2003/86.html

[17] Lehrbaum (2000, July). Using Linux in Embedded and Real-Time Systems. Linux Journal Volume 2000 , Issue 75

[18] Herlien (1998). Linux in an Embedded Communications Gateway. Linux Journal Volume 1998 , Issue 54

[19] Mockus, A., Fielding, R. T., and Herbsleb, J. 2000. A case study of open source software development: the Apache server. In *Proceedings of the 22nd international Conference on Software Engineering* (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM Press, New York, NY, 263-272.

[20] Oetiker. T.- MRTG – The Multi Router Traffic Grapher – Retrieved on 9th July 2007 from http://oss.oetiker.ch/mrtg/

[21] PeeWeeLinux – A Small Linux Distribution for Embedded Application – Retrieved on 11th July 2007 from http://www.peeweelinux.com/

[22] muLinux - Minimalistic Linux distribution - Retrieved on 11th July 2007 from http://mulinux.dotsrc.org/

[23] LEAF – Linux Embedded Appliance Firewall - Retrieved on 11th July 2007 from http://leaf.sourceforge.net/

[24] MadWIFI Drivers website - Retrieved on 11th July 2007 from http://www.madwifi.org

[25] MadWIFI |Issue Tracker Database - Retrieved on 6th August 2007 from http://madwifi.org/ticket/698

Appendix A - User Interface

It was felt that it is very important to make the basic configuration of the router as simple as possible, so a web-based interface was developed. This interface allows configuration of the basic network and wireless settings, including static IP addressing, wireless radio configuration and static routing, as well as showing information about the current status of the system.

The interface is implemented using thttpd [9], a turbo/tiny/throttling HTTP server and a series of Perl and shell scripts. Limited use is made of CSS style sheets within the HTML. The Perl implementation is Microperl, a cut-down version of Perl specifically designed for embedded systems with a small memory footprint.
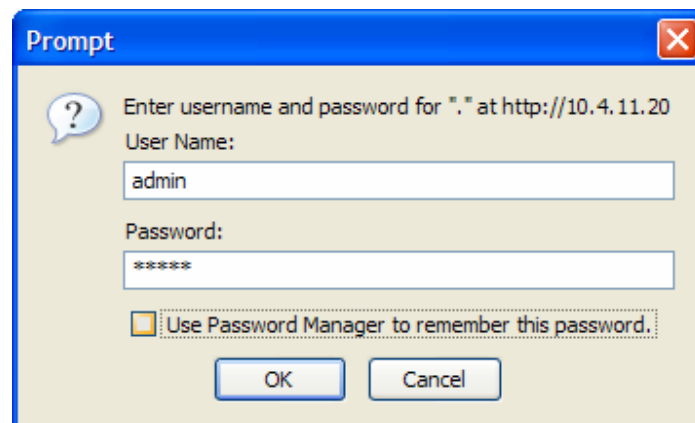
A.1 Network Interface Configuration

The current network interface information is available by selecting the **interfaces** option on the main menu.
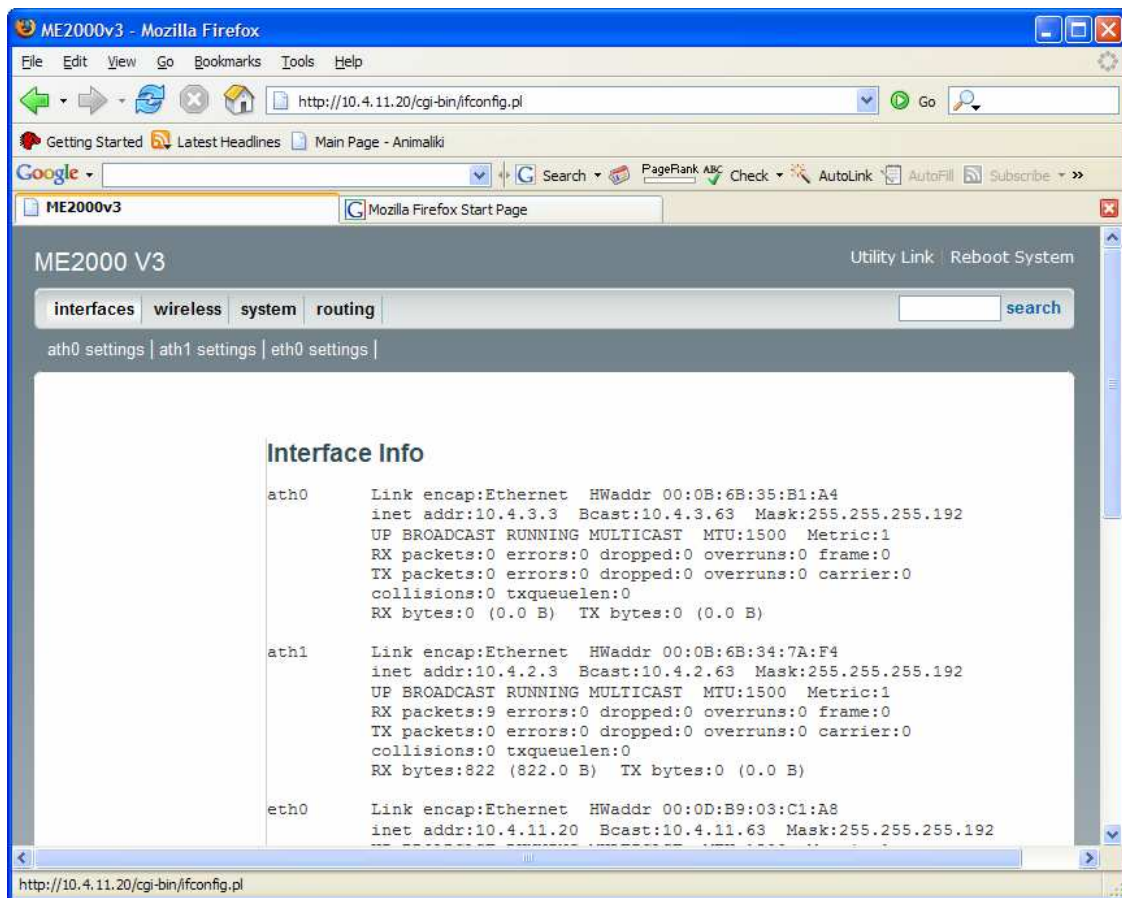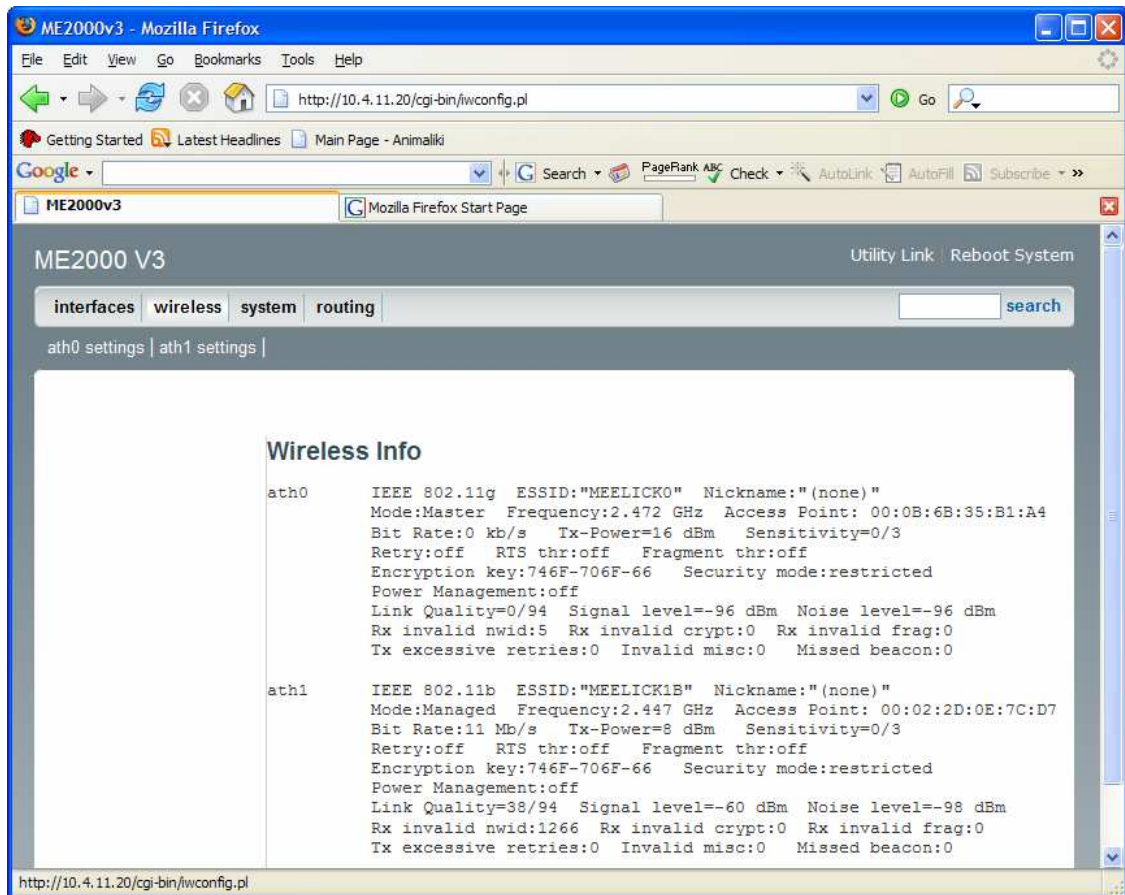

Network Interface Information

Selecting a particular interface from the **interfaces** sub-menu allows the user to configure the settings.
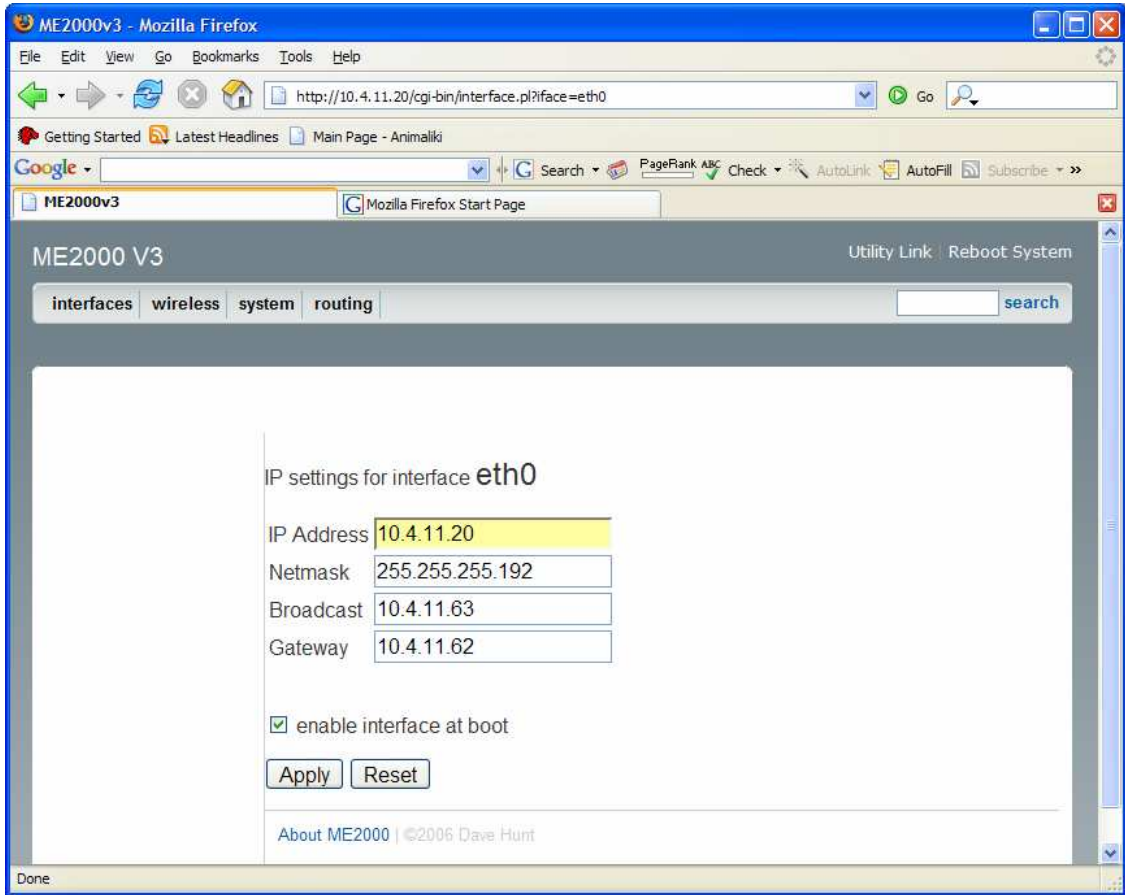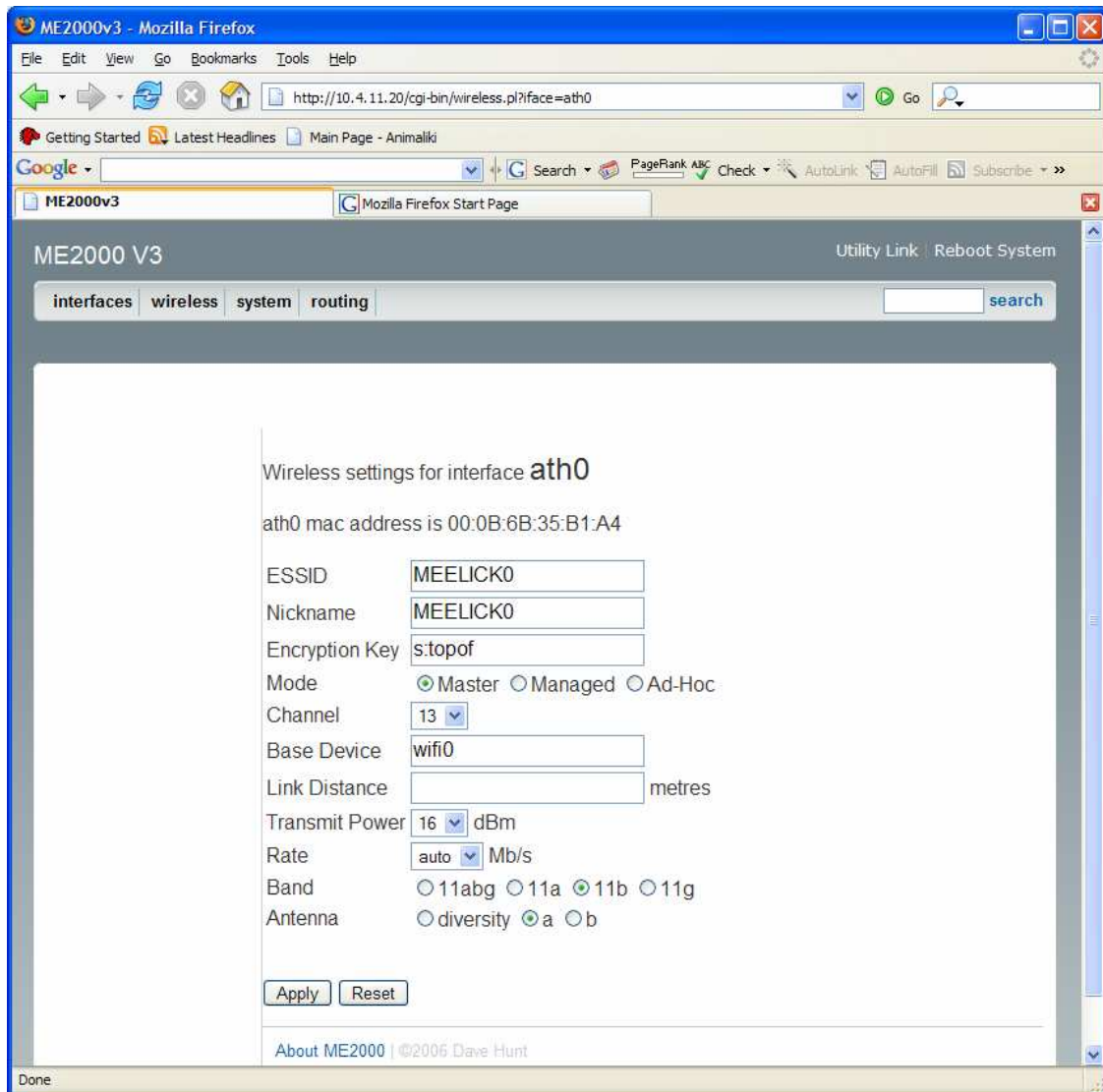
Network Interface Configuration

A.2 Wireless Configuration

The current wireless configuration information is available by selecting the **wireless**

option on the main menu.

Selecting a particular interface from the **wireless** sub-menu allows the user to configure

the settings for a particular radio card.

Wireless configuration

The configuration options available are:

- ESSID

- Nickname

- Encryption Key (64-bit or 128-bit WEP in ASCII or HEX)

- Selection of Master (access point), Managed (client), or ad-hoc modes of operation.

- Channel selection (depends on band active on radio card)

- Base device (Madwifi driver attribute of each radio, generally best left alone)

- Link distance (important for long distance links)

- Transmit Power (important for staying within the allowed power limits)

- Bit-rate adjust

- Band (802.11a/b/g, 802.11a, 802.11b or 802.11g)

- Antenna Mode (diversity, or selection of one of the two antenna ports, important for systems with one large directional antenna per radio)

A.3 System Information

System Information

A.4 Static Routing

A.5 Miscellaneous Features

The web-front end includes a copy of the putty.exe binary which is a secure shell terminal program for windows. This allows the user to log in via ssh in order to carry out advanced configuration that may not be available via the web front end. Also available is the ability to save the chanced settings to persistent storage, and reboot the system.

A.5.1 Save Settings

When any configuration changes are made via the web interface, the "Save Settings" menu option will appear in the utilities menu. This gives the user to save the changed settings to persistent storage. This option simply uses the "tar" utility to create an archive of critical configuration files on the system and copies it to the filesystem on the Compact Flash card. Early in the boot sequence, the system looks for this archive and extracts it, where the settings will then override the "factory" defaults. These settings will also survive a kernel upgrade and a root filesystem upgrade.

A.5.2 Reboot System

The user may wish to apply new configuration settings, or reboot after a kernel or root filesystem upgrade. The "Reboot System" menu option will allow the user to reboot the system, booting into the new operating system or configuration. A confirmation message is displayed to ensure that a reboot is not executed accidentally.
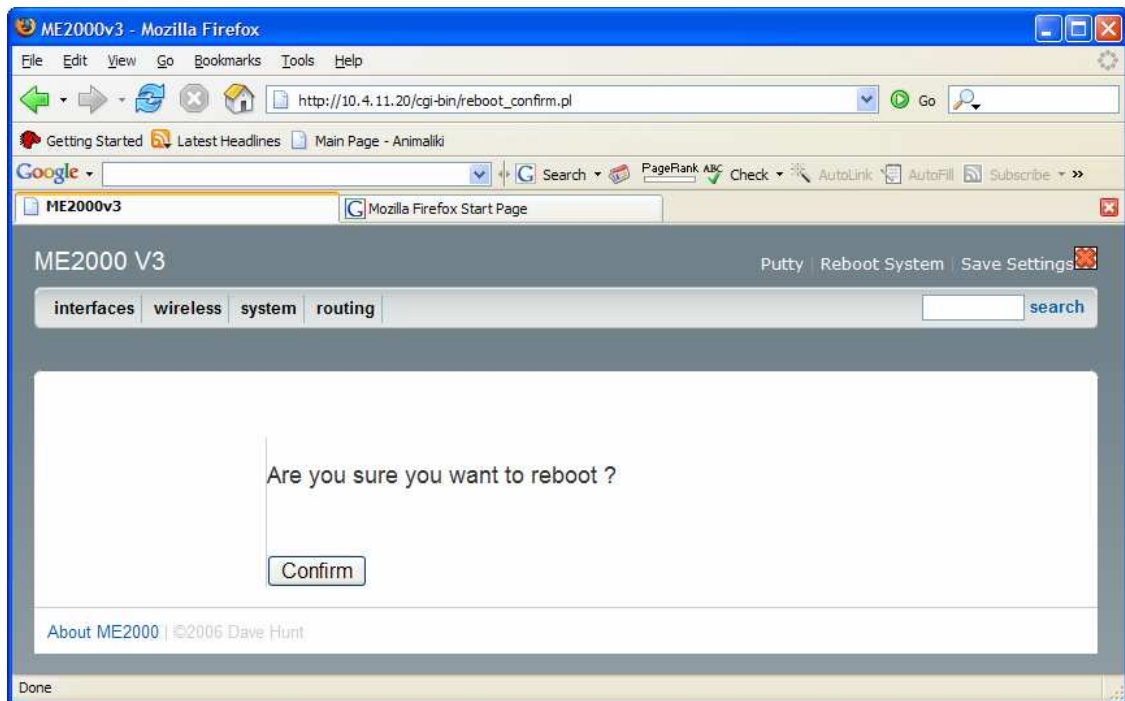
Reboot Confirmation

Appendix B – Source Code

B.1 Source Code Introduction

Only sample source code segments are shown to demonstrate concepts. The author did

not feel it was necessary to include all source code developed as part of the case study. Of

course, should the full source code be required, it is available upon request.

B21 Routerboard Image Update Script

The following script is used for upgrading the kernel on the routerboard. A brief

summary is as follows:

- Clean up any previous upgrades

- Get kernel image

- Get kernel image checksum

- Ensure checksup ok against downloaded kernel image

- Write kernel image into relevant place on CF card

```
#!/bin/sh
#
# This script gets the remote kernel/rootfs file along with it's md5sum,
generates a local md5sum.
# If the two checkums compare fafourably, the image is written onto the
first partition of the CF.
#
# Dave Hunt, April 2007.
#

FD=/boot
URL=${1}

if [ -f ${FD}/vmlinux ] ; then
        /bin/rm ${FD}/vmlinux
fi
if [ -f ${FD}/local.md5 ] ; then
        /bin/rm ${FD}/local.md5
```

```
fi
if [ -f ${FD}/remote.md5 ] ; then
        /bin/rm ${FD}/remote.md5
fi

# wget http://10.4.2.9/pub/me2000/vmlinux.rb532 -O${FD}/vmlinux
/usr/bin/wget ${URL} -O${FD}/vmlinux
if [ -f ${FD}/vmlinux ] ; then
        echo "Received Image File."
else
        echo "*** ERROR: Could not get image file, aborting."
        exit 1
fi

# wget http://10.4.2.9/pub/me2000/vmlinux.rb532.md5 -O${FD}/remote.md5
/usr/bin/wget ${URL}.md5 -O${FD}/remote.md5
if [ -f ${FD}/remote.md5 ] ; then
        echo "Received Image File checksum."
else
        echo "*** ERROR: Could not get image file checksum, aborting."
        exit 1
fi

/usr/bin/md5sum ${FD}/vmlinux >>${FD}/local.md5

/usr/bin/diff -q ${FD}/remote.md5 ${FD}/local.md5

if [ $? == 0 ] ; then
        echo "Image file md5 checksum OK."
else
        echo "*** ERROR: New image file corrupt. Aborting update."
        exit 1
fi

if [ -f ${FD}/vmlinux ] ; then
        echo "Writing image to Compact Flash..."
        /bin/dd if=${FD}/vmlinux of=/dev/cfa1
        echo "Cleaning up."
        /bin/rm ${FD}/vmlinux
        echo "Update Successful."
else
echo "Error getting image file. Exiting."
fi
```

B.3 Sample web-front end Perl script

The following script shows the common structure of the web-front end files. Most of the web front-end scripts follow this structure. Initially, they call the header() function to draw the header, menus, etc. Then a function to do the work specific to that screen, then finish off the script with a call to the footer function.

```perl
#!/usr/bin/perl
#
use commonfuncs;
use hfuncs;
use assoc;

header();

print "<h1>Wireless Info</h1>\n";

print "<pre>\n";

iwconfig();

print "</pre>\n";

footer();
```

B.4 Sample web-front end Header/Footer code

The following code is taken from hfuncs.pm which is a perl module containing functions related to drawing the header and the footer of all web pages output as part of the web front-end of the case study. This file contains two main functions,- header() and footer(). header() draws the menus, which are context sensitive depending on which screen is currently displayed. Also, extra menu options appear if the settings have been changed, drawing the users attention to the fact that the settings need to be saved to persistent storage.

```perl
#!/usr/bin/perl
#

sub ltrim($)
{
        my $string = shift;
        $string =~ s/^\s+//;
        return $string;
}
1;



sub header {
print <<EMBEDDED_HTML;




<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>ME2000v3</title>
<link rel="stylesheet" href="/emx_nav_left.css" type="text/css">
<script type="text/javascript">
<!--
var time = 3000;
var numofitems = 7;

function menu(allitems,thisitem,startstate){
  callname= "gl"+thisitem;
```

```
divname="subglobal"+thisitem;
  this.numberofmenuitems = 4;
this.caller = document.getElementById(callname);
  this.thediv = document.getElementById(divname);
this.thediv.style.visibility = startstate;
}

function ehandler(event,theobj){
  for (var i=1; i<= theobj.numberofmenuitems; i++){
  var shutdiv =eval( "menuitem"+i+".thediv");
  shutdiv.style.visibility="hidden";
}
  theobj.thediv.style.visibility="visible";
  }

  function closesubnav(event){
if ((event.clientY <48)||(event.clientY > 107)){
for (var i=1; i<= numofitems; i++){
  var shutdiv =eval('menuitem'+i+'.thediv');
//shutdiv.style.visibility='hidden';
}
  }
  }
function netmode()
{
        if(document.form1.mode[1].checked){

//window.open('wep1.html','','toolbar=no,location=no,scrollbars=yes,widt
h=760,height=600');
                document.form1.channel.disabled = true;
                document.form1.all("chn").disabled = true;
                document.form1.nmode.value = "Master1";
        } else {

//window.open('wep2.html','','toolbar=no,location=no,scrollbars=yes,widt
h=760,height=600');
                document.form1.channel.disabled = false;
                document.form1.all("chn").disabled = false;
                document.form1.nmode.value = "Master2";
        }
        if(document.form1.mode[0].checked){
                document.form1.nmode.value = "Master";
        }
        if(document.form1.mode[1].checked){
                document.form1.nmode.value = "Managed";
        }
        if(document.form1.mode[2].checked){
                document.form1.nmode.value = "Ad-hoc";
        }
}
function do_band()
{
        if(document.form1.band[0].checked){
                document.form1.nband.value = "0";
        }
        if(document.form1.band[1].checked){
                document.form1.nband.value = "1";
```

```
        }
        if(document.form1.band[2].checked){
                document.form1.nband.value = "2";
        }
        if(document.form1.band[3].checked){
                document.form1.nband.value = "3";
        }
}
function do_antenna()
{
        if(document.form1.antenna[0].checked){
                document.form1.nantenna.value = "0";
        }
        if(document.form1.antenna[1].checked){
                document.form1.nantenna.value = "1";
        }
        if(document.form1.antenna[2].checked){
                document.form1.nantenna.value = "2";
        }
}
function encmode()
{
        if(document.form1.encryption.checked){
                document.form1.EncryptionSettings.disabled = false;
        } else {
                document.form1.EncryptionSettings.disabled = true;
        }
}


  // -->
</script>
</head>

<body onmousemove="closesubnav(event);">
<div class="skipLinks">skip to: <a href="#content">page content</a> | <a
href="pageNav">links on this page</a> | <a href="#globalNav">site
navigation</a> | <a href="#siteInfo">footer (site information)</a>
</div>
<div id="masthead">
  <h1 id="siteName">ME2000 V3 </h1>
<div id="utility">
<a href="#">Utility Link</a> | <a href="/putty/putty.exe">Putty</a> | <a
href="/cgi-bin/reboot_confirm.pl">Reboot System</a>

EMBEDDED_HTML
if ( -f "/tmp/settings_changed.flag" ) {
        print (" | <a href=\"save_settings.pl\">Save Settings</a><img
src=\"/warning.gif\" alt=\"Settings have changed since last save.\">");
}
print <<EMBEDDED_HTML;
</div>
<div id="globalNav">
<img alt="" src="/gblnav_left.gif" height="32" width="4" id="gnl"> <img
alt="" src="/glbnav_right.gif" height="32" width="4" id="gnr">
<div id="globalLink">
  <a href="/cgi-bin/ifconfig.pl" id="gl1" class="glink"
onmouseover="ehandler(event,menuitem1);">interfaces</a>
```

```
<a href="/cgi-bin/iwconfig.pl" id="gl2" class="glink"
onmouseover="ehandler(event,menuitem2);">wireless</a>
  <a href="#" id="gl3" class="glink"
onmouseover="ehandler(event,menuitem3);">system</a>
<a href="/cgi-bin/route.pl" id="gl4" class="glink"
onmouseover="ehandler(event,menuitem4);">routing</a>
</div>
<!--end globalLinks-->
<form id="search" action="">
  <input name="searchFor" type="text" size="10">
<a href="">search</a>
</form>
  </div>
<!-- end globalNav -->
<div id="subglobal1" class="subglobalNav">
EMBEDDED_HTML

#
# Generate the menus for ip settings for the relevant interfaces
#
$ifaces = `/sbin/ifconfig -a`;


@ifaces = split (/\n/,$ifaces);

foreach ( @ifaces ) {
        if ( index ( $_, "Ethernet" ) ne -1 )
        {
                @line = split (/ /,$_);
                print "<a href=\"/cgi-
bin/interface.pl?iface=$line[0]\">$line[0] settings</a> | \n";
        }
}

print <<EMBEDDED_HTML;
</div>
<div id="subglobal2" class="subglobalNav">
EMBEDDED_HTML

#
# Generate the menus for wireless settings for the relevant interfaces
#

$ifaces = `cat /proc/net/wireless`;

@ifaces = split (/\n/,$ifaces);

foreach ( @ifaces ) {
        @line = split (/\|/,$_);
        if ( ( $line[0] =~ "Inter-" ) || ( $line[0] =~ " face " ) ) {
                    $line[0] = "";
        } else {
                @line = split (/:/,$_);
                $line[0] = ltrim ( $line[0] );
                print "<a href=\"/cgi-
bin/wireless.pl?iface=$line[0]\">$line[0] settings</a> | \n";
        }
```

```
}

print <<EMBEDDED_HTML;
</div>
<div id="subglobal3" class="subglobalNav">
  <a href="/cgi-bin/iwconfig.pl">wireless info</a> |
  <a href="/cgi-bin/meminfo.pl">memory info</a> |
  <a href="/cgi-bin/cpuinfo.pl">cpu info</a> |
  <a href="/cgi-bin/sensors.pl">sensor info</a>
</div>
<div id="subglobal4" class="subglobalNav">
EMBEDDED_HTML

@ifaces = `ifconfig -a | grep Ethernet | cut -f1 -d' '`;
foreach ( @ifaces ) {
        print "<a href="#">$_ settings</a> |";
}

print <<EMBEDDED_HTML;
</div>
</div>
<!-- end masthead -->
<div id="pagecell1">
  <!--pagecell1-->
<p><img alt="" src="/tl_curve_white.gif" height="6" width="6" id="tl">
<img alt="" src="/tr_curve_white.gif" height="6" width="6" id="tr"></p>
  <p> 
</p>
  <div id="content">
EMBEDDED_HTML
}
1;


sub footer {

print <<EMBEDDED_HTML;
  </div>
   <div id="siteInfo">
   <a href="http://www.me2000.net">About ME2000</a> | &copy;2006
    Dave Hunt</div>
</div>
<!--end pagecell1-->
<br>
<script type="text/javascript">
    <!--
      var menuitem1 = new menu(7,1,"hidden");
                            var menuitem2 = new menu(7,2,"hidden");
                            var menuitem3 = new menu(7,3,"hidden");
                            var menuitem4 = new menu(7,4,"hidden");
                            var menuitem5 = new menu(7,5,"hidden");
                            var menuitem6 = new menu(7,6,"hidden");
                            var menuitem7 = new menu(7,7,"hidden");
    // -->
    </script>
</body>
</html>
```

```
EMBEDDED_HTML
}
1;
```

B.5 Sample Daemon start/stop script

This script is an example of the start/stop script for a deamon that is started at boot time, and stopped at shutdown. As is standard across many flavours of linux, there is a script in the /etc/init.d directory that takes a 'start' or 'stop' parameter. These scripts are called at boot and shutdown to ensure clean operation of those daemons. The script shown is for the thttpd daemon, which is the web-server daemon used in the case study. The 'start' parameter causes wrapper script that keeps the daemon running to be started, and the 'stop' parameter causes the daemon to be killed.

```sh
#!/bin/sh
#
# thttpd.sh - startup script for thttpd on FreeBSD
#
# This goes in /usr/local/etc/rc.d and gets run at boot-time.

case "$1" in

    start)
    if [ -x /sbin/thttpd_wrapper ] ; then
        echo -n " thttpd"
        /sbin/thttpd_wrapper &
    fi
    ;;

    stop)
    kill -USR1 `cat /var/run/thttpd.pid`
    ;;

    *)
    echo "usage: $0 { start | stop }" >&2
    exit 1
    ;;

esac
```

Annotated Bibliography

[1] Buildroot Embedded Linux Website - http://buildroot.uclibc.org/ - Accessed 25th September 25, 2006. No published date.

> The Buildroot system is the basis of the operating system developed as part of the case study. The Buildroot website was an invaluable resource for the framework around which the operating system was built. It contains a subversion repository for many buildroot specific makefiles and patches, which allow easy inclusion of specific packages into the operating system. Also included is documentation on how to add new packages (such as the wireless card drivers) so that the operating system can be customized to suite the case study's target situation.

[2] BusyBox: The Swiss Army Knife of Embedded Linux http://www.busybox.net- Accessed 25th September 25, 2006. No published date.

> With Buildroot as the basis for the operating framework, the BusyBox website provided the basis for many of the basic operating system binaries. Everything from the *init* process to many of the more commonly used binaries, *ls*, *pwd*, etc. are provided by busybox. The website gives details of each utility available, and how to include these in the target build.

[3] Gateworks Corporation Website - http://www.gateworks.com/- Accessed 25th September 25, 2006. No published date.

> The Gateworks Corporation website is the home website of the manufacturer of one of the target boards used in the case study. It contains details of the various boards, should the case study be extended in the future to include more target boards. A ready-patched linux kernel is also available, greatly easing the implementation of the case study on that particular board. It also contains instructions on how to join the Avila mailing list, which was an invaluable resource in helping resolve some issues during the implementation of the case study.

[4] Mikrotik Website - http://www.mikrotik.com/- Accessed 25th September 25, 2006. No published date.

> The Mikrotik website is the home website of the manufacturere of another of the target boards used in the case study. It contains many useful faqs on the board in question (RB532), and a sample linux kernel for loading onto the board. While this kernel was not very feature rich, it was an invaluable starting point in getting the initial case-study operating system running on the board. The website also contains useful documentation on the boot loader used on the RB532 board, and instructions on how to load new operating systems onto either the built in flash memory, or an external Compact Flash card.

[5] PC Engines - Embedded PC Design - http://www.pcengines.ch/- Accessed 25th September 25, 2006. No published date.

This is a link to another single board computer manufacturer website who's board was used as part of the case study. As with the others, this contained a reference linux kernel for use on the board, along with instructions on how to get it running. This was another invaluable resource while implementing the case study.

[6] ME2000 Wireless Router Node System - http://www.me2000.net- Accessed 25th September 25, 2006. No published date.

ME2000 was the name given to the operating system developed as part of the case study, and the ME2000 website was where many of the daily notes were logged, any problems encountered and any solutions discovered were recorded on the ME2000 wiki pages. Also some thoughts on the direction of the capstone paper were recorded to help the author's thought process through the case study implementation.

[7] Vonk. C.J.S. (2005) - Secure Internet Appliance for Small Office / Home Office HOWTO – Retrieved on 19th April 2006 from http://users.gotsky.com/cvonk/linux/siso/

This HOWTO gives instructions on how to implement an operating system on one of the target platforms in the case study, the WRAP 2c single board computer. Rather than supplying a turnkey, ready-made solution, it shows how to develop an operating system from scratch for the WRAP board, and was a good starting point for implementing the case study on all three boards selected. It also helped with parts of the operating system not covered in any of the other websites, such as the character and block device nodes in the /dev directory.

[8] Redboot Website - RedBoot Debug and Bootstrap Firmware – Accessed 18th March 2007 from http://www.ecoscentric.com/ecos/redboot.shtml

Each single board computer had a different boot loader. On the Gateworks board, it's recommended bootloader was Redboot, and this website contains information on the Redboot boot loader. It gives examples on how to use the tftp features for loading the kernel and compressed filesystem into memory from a remote tftp server across the network, as well as how to burn those files onto the internal flash memory or onto an external Compact Flash card.

[9] thttpd website - tiny/turbo/throttling HTTP server - Accessed 18th March 2007 from http://www.acme.com/software/thttpd/

The http daemon chosen for the implementation of the case study was the thttpd daemon, mainly because of it's compact size. The thttpd website lists the daemons

features, as well as a comparison chart with many other http daemons. It's binary size is approximately 50Kbytes, with many other daemons being 500K or larger. In an embedded system, size is important, and although it may not be as feature-rich as a web-server like apache, it was considered suitable for use in the case study.

[10] IrishWAN community website – Accessed 18th March 2007 from http://www.irishwan.ie

The IrishWAN community website is a source of information on wireless networking, and was one of the main inspirations behind the case study. The experience and knowledge gained as part of that community over the last several years has helped the author see the need for research into the area of open source operating systems and the possibility of producing an operating system that could be customized to that community's needs, while also allowing peer review of it's security aspects.

[11] GNU GRUB – GRand Unified Bootooader website - Accessed 7th May 2007 from http://www.gnu.org/software/grub/

This is the main website for the boot loader used on one of the single board computers in the case study. The WRAP board boots similar to a PC, so it looks for a Master Boot Record on the Compact Flash card and then loads the bootstrap program. Grub was selected as the boot loader on the wrap because of it's popularity in the Linux world. The website contains instructions on how to install it on a hard drive, but it was very similar to implement on a Compact Flash card once it was mounted in a Linux laptop.

[12] Hoepman, J. and Jacobs, B. 2007. Increased security through open source. Commun. ACM 50, 1 (Jan. 2007), 79-83.

The author discusses the increased security in a system when open source is used. One of the main points of the article is that
"Open source enables users to evaluate the security by themselves, or to hire a party of their choice to evaluate the security for them. Open source even enables several different and independent teams of people to evaluate the security of the system."
This peer-review of a system helps alleviate fears that there are hidden security holes or compromised passwords in a system, as the source is there for anyone to investigate.

http://portal.acm.org.dml.regis.edu/ft_gateway.cfm?id=1188921&type=pdf&coll=Portal&dl=GUIDE&CFID=21824199&CFTOKEN=89871550

[13] XORP project website – Open Source IP Router - Accessed 7th May 2007 at http://www.xorp.org

XORP is an open source IP router operating system. The XORP Vision is to develop an operating system that appeals to researchers, educators, application writers and equipment vendors alike. While this may seem similar to the operating system developed as part of the case study, the goals are slightly different. The case study implementation as part of this capstone project is multi-platform, to allow a wide variety if hardware to be used in a wireless networking environment. XORP's aim is a wider user base, but on x86 based hardware only.

[14] Quagga project website – Quagga Routing Software Suite, GPL licensed IPv4/IPv6 routing software. - Accessed 7th May 2007 at http://www.quagga.net/

Because the case study is to form part of a large wireless operating system, it is important to have a selection of routing daemons available. Quagga is the most up-to-date of the routing daemon suites, having been branched from the popular Zebra routing daemon. Quagga contains routing daemons to handle OSPF, BGP, and RIP, as well as the Zebra routing manager daemon. The Quagga website contains documentation on each of these daemons, their use, and their configuration.

[15] Valemount Website – StarOS Operating System – Accessed on 10th May 2007 at http://www.staros.com/

The Valemount Website contains information on the StarOS operating system, which is a commercial (closed-source) operating system in wide use in the wireless networking community. Although StarOS uses several open-source components, the source code to the operating system itself is not available. The website details the many fearures available and is a good basis for comparison between a commercial solution and the open-source solution developed as part of the case study.

[16] VeriSign Website – *StarOS Static Provate Key Usage Vulnerability* – Accessed on 10th May 2007 at http://www.verisign.com/security-intelligence-service/current-intelligence/vulnerability-advisories/2003/86.html

The VeriSign website contains information regarding a vulnerability discovered in the StarOS operating system. The author feels that in situations like these that the operating system should be open to peer-review to help tighten up security, and avoid situations like this occurring.

[17] Lehrbaum (2000, July). Using Linux in Embedded and Real-Time Systems. Linux Journal Volume 2000 , Issue 75

In this article, the author compares the options for embedded systems developers, and why Linux is becoming more and more important in that field. Real Time Operating System (RTOS) vendors are struggling to keep up with new chipset developments, and some 'heavier' operating systems are not suited to embedded environments. The main relevant point in this article is that the operating system is

open source, and so lends itself to modification should the need arise. Also, the choice of Linux variants means the developer can chose a distribution that suits them. The author founded LinuxDevices.com, and would probably be biased towards Linux as the choice for embedded systems.

http://portal.acm.org.dml.regis.edu/ft_gateway.cfm?id=349542&type=html&coll=portal&dl=ACM&CFID=1895080&CFTOKEN=21357233

[18] Herlien (1998). Linux in an Embedded Communications Gateway. Linux Journal Volume 1998 , Issue 54

This article describes the reasons behind choosing Linux over several other operating systems for an embedded communications gateway. One main contributing factor was the cost of alternatives, but also because of the robustness and available of tools and drivers. It also mentions the portability of the Linux to a wide variety of embedded devices. The conclusions of this article show Linux to be a reliable choice for an embedded system, and the fact that significant cost savings are to be had over commercial alternatives. The Author developed their own embedded system for use in the company in which he was employed, and seems reasonable as an unbiased reference.

http://portal.acm.org.dml.regis.edu/ft_gateway.cfm?id=327500&type=html&coll=portal&dl=ACM&CFID=1895080&CFTOKEN=21357233

[19] Mockus, A., Fielding, R. T., and Herbsleb, J. 2000. A case study of open source software development: the Apache server. In *Proceedings of the 22nd international Conference on Software Engineering* (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM Press, New York, NY, 263-272.

Mockus et. al. look at a successful example of open source development, the Apache Web Server. One of the concluding hypotheses are particularly relevant to the case study in this capstone project:
"OSS developments exhibit very rapid responses to customer problems."
The author feels that this is of particular importance to the reason behind the case study implementation, as the ability of developers to react quickly in an open source environment may allow quicker response than commercial organisations who may have other priorities.

http://portal.acm.org/citation.cfm?doid=337180.337209