Porting the BSD `net80211` wireless stack to the Mac OS X kernel

Prashant Vaibhav
p.vaibhav@jacobs-university.de

Jacobs University Bremen
Guided Research Proposal

Spring 2011

Abstract

This project aims to create a robust IEEE 802.11 stack for the XNU kernel used by Mac OS X and Darwin operating systems, which can be used by third party wireless device drivers in a standard way. This will reduce duplication of code, improve code reusability, and provide end-users with a set of standard configuration tools for third party wireless hardware. The wireless stack will largely be based on existing code from the BSD `net80211` infrastructure. The project aims to reuse code imported from the BSD source tree with minimal modifications, so that any bug fixes and enhancements from upstream can be merged back without copious effort. Since the BSD and XNU kernel functions and data structures have diverged over the years, a compatibility layer is needed to enable the `net80211` code to compile and function properly. Furthermore, an adapter will be created to marshall data and configuration parameters back and forth between the format expected by `net80211` and that expected by XNU's networking layer. Finally, OS X user-land configuration tools like `ifconfig` will be modified to allow configuration of wireless interfaces.

1 Introduction

Mac OS X includes the BSD networking stack in its XNU kernel[6]. The XNU kernel currently provides robust support for Ethernet networking. However, support for IEEE 802.11 wireless networking is limited to a set of extensions in the form of `ioctl` calls that the driver must support to allow configuration and reporting. Darwin, the free and open source version of the base operating system used by Mac OS X, entirely omits this support. Third party device drivers for wireless hardware typically have to re-implement this functionality, and masquerade as an Ethernet device with custom configuration GUIs. Ethernet 802.3 packets coming from the operating system are converted to 802.11 frames and vice versa. This requires inclusion of a large amount of 802.11 logic in every driver.

This project aims to create a robust wireless stack for the XNU kernel, which could be used by IEEE 802.11 device drivers in a standard way. This will reduce duplication of code, improve code reusability, and provide end-users with a set of standard configuration tools for third party wireless hardware.

The BSD `net80211` infrastructure is currently one of the most portable and well tested wireless stacks available. The proposed wireless stack will thus largely be based on existing code from `net80211`. We aim to reuse code imported from the BSD source tree with minimal modifications, so that any bug fixes and enhancements from upstream can be merged back without copious effort. Since the BSD and XNU kernel functions and data structures have diverged over the years[5], a compatibility layer is needed to enable the `net80211` code to compile and function properly.

Furthermore, an adapter will be created to marshall data and configuration parameters back and forth between the format expected by `net80211` and that expected by the XNU kernel. Finally, OS X user-land configuration tools like `ifconfig` will be modified to allow configuration of wireless interfaces.

2 Motivation

The wireless stack used in Mac OS X is ad-hoc built for each device driver that Apple includes in their hardware products. A standard base class, `IO80211Controller` exists which adds certain functionality to the base class for ethernet controllers: `IOEthernetController`, allowing configuration parameters to be passed back and forth between the kernel device driver and the configuration GUI. This entire code is completely proprietary. Although the header files to compile against the `IO80211*` interfaces were included with earlier versions of Mac OS X, these have been discontinued in newer iterations. The `IOEthernetController` class, moreover, expects to receive network packets in IEEE 802.3 frame format, and provides the same to the drivers to be sent out. Third party wireless hardware manufacturers therefore have to implement their own wireless stack and configuration tools.

Open source implementations of the Darwin OS (on which Mac OS X is based) exist in the form of OpenDarwin and PureDarwin. These operating systems aim to provide a drop-in replacement for a more traditional Linux / BSD based system. However, these are plagued by extremely limited hardware support, most often limited to the drivers Apple provides as part of source code releases for Mac OS X. Support is entirely absent for wireless networking. Having a standard set of UNIX-compliant configuration tools on the user end, and a standard wireless stack on the kernel end would greatly improve the suitability of Darwin-based operating systems when run on commodity hardware.

A further problem surfaces when one wishes to dual-boot a Macintosh with Mac OS X and one of the BSDs. Broadcom wireless chips included with most Macintosh computers today do not have a stable driver for BSD platforms. Under BSD, a user is forced to use either a Windows driver under emulation (`NDISulator`), or replace their wireless card with a well-supported one, e.g. Intel PRO/Wireless. The latter has no official driver support under Mac OS X. The user is thus in a situation where only one or the other operating system will support their wireless hardware. Having a BSD `net80211`-like stack would enable creation of OS X drivers for the vast amount of wireless hardware that BSDs already support.

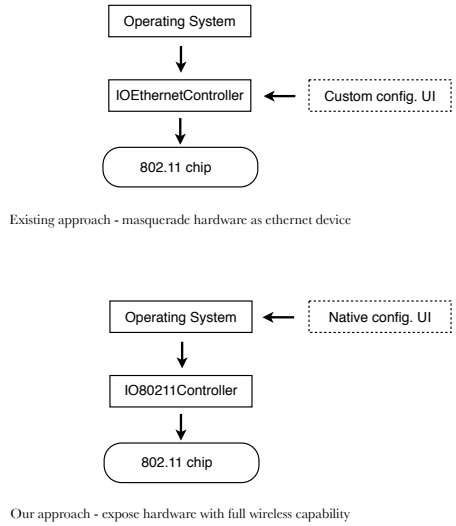

Figure 1: Exposing the hardware to the operating system: comparing two approaches

From an academic point of view, the project will highlight the differences between the explicitly C++ object-oriented driver model of Mac OS X, and the C-based "UNIX-like" driver model of the BSDs. Various design challenges exist when converting source code from one kernel to another within the same family (e.g. OpenBSD to NetBSD). Porting code between two completely different kernel architectures, e.g. development of the adapter layer to wrap the `net80211` input/output functions and data structures will prove to be an interesting case study of code sharing and reuse.

3 Current state of the art

Prior work in this area falls under two categories:

1. Third party wireless device drivers ported to Mac OS X or written from scratch.

2. Porting efforts to bring BSD ethernet and wireless code to other operating systems.

Commercial wireless device drivers are currently only available from Ralink Technologies, which builds 802.11 chips for various USB and PCI form factors. Drivers for the Prism/Prism2 range of 802.11 chips are available from a third party, but this project has been stagnant for several years. These drivers masquerade as a regular wired device, include private 802.11 stacks within the driver, and have their own configuration utilities.

Other open source efforts have typically focused on the Intel range of wireless devices, which has no official support in OS X. The `iwidarwin` project was an effort to port the Linux drivers of some Intel cards to OS X. These drivers came with a custom configuration utility and also appeared as regular wired ethernet device to operating system.

More recently, the author's own "Project Camphor" drivers have attempted to implement Intel drivers from the ground up using FreeBSD driver code as reference[7]. This project included a nano-size 802.11 stack written from scratch and supporting the bare minimum functionality to get STA (station) mode working. This 802.11 stack is available as a separate framework called "VoodooWireless."[1] These drivers also support the OS-native "Airport" configuration interface.

In terms of porting the BSD 802.11 stack to another operating system, Sam Leffler[4], the originator of FreeBSD `net80211` stack, himself ported it to Linux to provide support for Atheros wireless driver for Linux. This port is now maintained under the umbrella of the "MadWiFi" project.

The free operating system Haiku currently includes a compatibility layer for FreeBSD wireless and wired networking drivers[2]. This layer re-implements several portions of the FreeBSD networking stack, including re-implementing most of the `mbuf` routines to manage network packets. This was possible in Haiku's case because their network stack shares little to no code with other BSDs, thus avoiding most naming conflicts.

[1]The project described in this proposal is largely a continuation and evolution of VoodooWireless, replacing the custom 802.11 code with BSD's `net80211`.

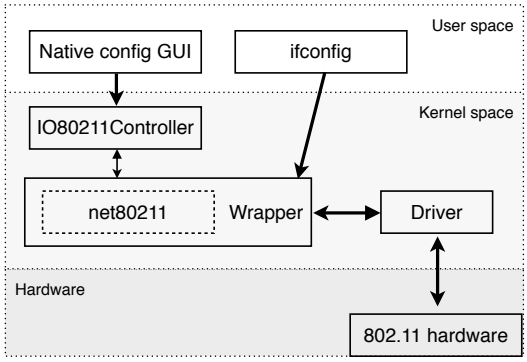

Figure 2: Proposed system model

4 Planned execution

Figure 2 depicts the proposed model. The project consists of three parts:

1. Kernel space: A compatibility layer that wraps around OpenBSD `net80211` code allowing it to run under XNU.

2. Driver: also in kernel space, a test driver based on the OpenBSD Intel 3945 wireless card

3. User space: `ifconfig` configuration utility modified to interface with the in-kernel code

4.1 The `net80211` subsystem and kernel-space wrapper

This is the crux of the entire system. The `net80211` code from the OpenBSD source tree will be used as the basis. Every source code file will then be scanned carefully to identify differing functionality that has a

direct counterpart under XNU, and replacing them with appropriate replacement code. The `ioctl` and `ifnet` related code will then be changed to interface with XNU networking layer so that two-way communication between them can occur transparently.

4.1.1 Choice of `net80211` variant

The different flavors of BSD (FreeBSD, OpenBSD, NetBSD, DragonflyBSD *et al*) all have slightly differing versions of `net80211`. Notably, OpenBSD has a variant of `net80211` based on an early version from FreeBSD. Thereafter, the FreeBSD version included substantial new features which were imported into the remaining BSD flavors.[3] For this project, the OpenBSD variant was chosen due to its simplicity and brevity, while still including most functionality desired by end users and driver developers.

4.1.2 Porting `net80211` code

It is not expected that the `net80211` code from OpenBSD will work without any changes, even if a compatibility layer were implemented. This is because the XNU kernel shares a lot of code with BSD. There are a number of symbol naming conflicts. For example, the `ifnet` structures are used under BSD to represent network interfaces. Under XNU, `IONetworkInterface` and derived classes are used by drivers, however they are in turn based on BSD `ifnet` structure. The XNU kernel programming interface does not allow direct access to this, and to several other structures.

Code must thus be ported manually by scanning all such relevant functionality that the `net80211` layer requires, and modifying them to make use of XNU-native access mechanism. Most changes would need to be done in code that access network packet buffers. The network packet routines in BSD operate directly on `mbuf` structures, while under XNU they have been homogenized to use the new `mbuf_t` data type and its associated functions. These need to be manually converted to ensure safe operation and compatibility with future releases of XNU. Other changes include changing the locking/mutex implementation (which thankfully has been abstracted out in the `net80211` code), replacing timing code (e.g. use of "`hz`" variable which does not exist on tickless kernels like XNU) and re-implementation of several in-kernel cryptography services.

4.1.3 Hooking into XNU networking stack

As described earlier, most of the `net80211` code accesses BSD network stack directly.[1] Access to it is not provided for hardware drivers running under XNU. The interface between `net80211` and the XNU networking stack will therefore need to be specifically written to marshall data and control parameters appropriately. This constitutes the wrapper layer around our modified `net80211`.

Configuration of 802.11 interfaces on OS X is typically done using `ioctl`-like function calls made to the `IO80211Controller` class. The wrapper layer will be implemented as a subclass of the `IO80211Controller`, converting control parameters back and forth between `net80211` code. Functionality to pass network packets between the networking stack and the `net80211` subsystem will also be implemented inside the wrapper, replacing direct `ifnet` access with `IO80211Interface`. Most BSD drivers include their own network packet queue management. The wrapper layer will replace the built-in packet queue management of `net80211` with standard XNU packet queue functionality using `IOOutputQueue` and friends. Finally, power management functionality (i.e. suspend, resume) etc. will be implemented to correctly reset the driver and `net80211` subsystem when the operating system notifies the wrapper of these events.

4.2 Test driver

To test the `net80211` subsystem along with its wrapper, we could either implement a dummy driver, or a live working one. The latter is preferred as it would not only test every aspect of the project thoroughly but also provide something useful in the end.

To this effect, the "Intel PRO/Wireless 3945ABG" card has been chosen. This 802.11 chip by Intel has very good Linux, BSD and Windows support, but lacks support for OS X. Since well-documented and

stable driver along with source code is available for this card, it should be possible to port it to OS X with fairly reasonable effort. Again the OpenBSD variant was chosen, as it not only relies on OpenBSD's `net80211` variant, but is also more feature-complete compared to other BSDs.

The BSD driver needs access to the hardware, and does so using BSD hardware access services. These can either be converted to native XNU model (I/OKit), or wrapped around another compatibility later. We choose the latter option as it will allow maximum portability with other BSD wireless drivers. Hardware access, DMI memory allocation and related functionality will be wrapped into an emulation/compatibility layer using native XNU kernel interfaces.

4.3 User-space configuration tool

Although subclassing `IO80211Controller` will give us access to OS X's native configuration utility, XNU/Darwin-based operating systems like PureDarwin do not include this. The standard method of configuring network interfaces on most UNIX-like operating systems is via a command-line tool like `ifconfig`. OS X and other Darwin-derived operating systems include a modified version of this tool, but exclude capability to configure 802.11 adapters. Thus, `ifconfig` will be extended to include configuration parameters for 802.11 adapters, generating appropriate `ioctl` calls. These calls can directly be passed to the `net80211` wrapper. The hardware can then be configured from user-space using this configuration tool.

5 Testing methodology and evaluation

The final deliverables for the project would be a working 802.11 framework based on `net80211`, using which wireless drivers can be written for OS X. In addition, a test driver for the Intel 3945 adapter, as well as a command-line user-space configuration tool will be implemented. The primary test of the port would be proper functioning of the test driver described in section 4.2. Benchmarks will be run to measure throughput and reliability under stress of the Intel 3945 wireless adapter, compared to the same hardware running under OpenBSD.

Ultimately, this project is self-evaluating in the sense that achieving a state of stable usage is enough to verify proper working. Nevertheless, some benchmarking will help establish the practicality of this project.

References

[1] Gangwal, P. *Implementation and experimental study of rate adaptation algorithms in IEEE 802.11 wireless networks.* PhD thesis, Iowa State University, 2009. pages 3–6.

[2] Guenther, C. Haiku: Wifi stack prototype works. `http://www.haiku-os.org/blog/coling/2009-07-12/wifi_stack_prototype_works`, 2009.

[3] Leffler, S. Wireless networking in the open source community. `http://people.freebsd.org/~sam/SANE2006-Wireless.pdf`, 2006. The 5th System Administration and Network Engineering Conference.

[4] Leffler, S. Multiband atheros driver for wifi (madwifi), 2005, 2008.

[5] McKusick, M. K., and Neville-Neil, G. V. *Design And Implementation Of The FreeBSD Operating System.* Addison Wesley, 2004.

[6] Singh, A. *Mac OS X Internals.* Addison-Wesley Professional, 2006.

[7] Vaibhav, P. Project camphor. `http://projectcamphor.mercurysquad.com/`.

[8] Vipin, M., and Srikanth, S. Analysis of open source drivers for ieee 802.11 wlans. In *Wireless Communication and Sensor Computing, 2010. ICWCSC 2010. International Conference on* (jan. 2010), pp. 1 –5.