Wireless networking with Linux and IEEE 802.11b

David Gibson <david@gibson.dropbear.id.au>, IBM LTC OzLabs

March 7, 2002

Abstract

802.11b is an IEEE standard for wireless local area networks. It has become quite popular due to the convenience of wireless networking and the reasonable flexibility and performance of the protocol. It supports both "ad-hoc" and infrastructure modes of operation. In ad-hoc (or IBSS) mode, several nearby stations communicate without designated central co-ordination. In infrastructure mode, stations communicate via a co-ordinating Wireless Access Point (AP). This is frequently, although not necessarily, also attached to a wired network. This mode also supports roaming, where a station can be moved at will between several APs without reconfiguration of its network parameters.

Most of the common 802.11b cards are supported under Linux, but the support is not entirely mature and consolidated. The two most common types of card (both based on Intersil's Prism II chipset, but using different firmwares) are each supported by several different drivers. However, each driver supports a different subset of any particular card's functionality. Only a few drivers support some of the more specialised functionality of the wireless devices such as acting as an AP, or passively monitoring a wireless network. Progress is slowed by the fact the chipset and firmware vendors have provided little information to open source developers about the specifications of the MAC controller and its firmware.

This paper will cover the state of support for the 802.11 protocol and 802.11b devices under Linux. In particular, the internals of the existing kernel code will be discussed, with a particular focus on the `orinoco` device driver.

1 Wireless networking

Wireless networking, obviously, is networking without physical wires between stations. This generally means that data is transferred between stations using radio or infrared modulation.

Wireless technology has of course been in used in telecommunications for a very long time. However what we're interested in is wireless communications used in the place of a normal wired LAN such as Ethernet. This is useful where it is inconvenient or impossible to install network cabling. In some cases it can be cheaper to build a wireless LAN than to install the infrastructure for a wired LAN, although certainly not always, since wireless NICs are significantly more expensive than standard Ethernet NICs.

Many such systems operate via radio in the 2.4GHz "ISM" (Industrial-Scientific-Medical) band. Because this frequency is maximally absorbed by water it has quite a short range. It is also the band used by microwave ovens, making it quite noisy. It is of little value for commercial use and may be used without a license in most parts of the world (at sufficiently low powers, at least).

Wireless networks have to deal with a number of issues which are not usually present in wired networks:

- An unreliable medium:
  - Random transmission errors due to noise are much more common than in wired networks (e.g. Ethernet)
- The hidden transmitter problem:

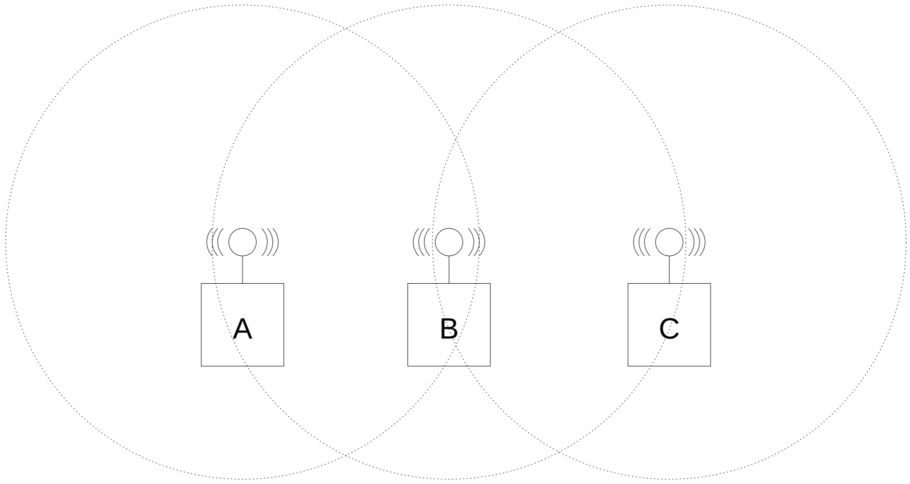

Figure 1: The “hidden transmitter” problem

  - Stations might not be able to receive transmissions from all other stations. e.g. in Figure 1 station B is within range of both the other stations, but station A cannot transmit to station C or vice versa
  - The situation can become even more complex with walls or other obstructions

- Mobile stations:
  - Stations can access the LAN while moving physical location.
  - Changes in the environment can cause radio propagation effects more-or-less indistinguishable from a moving station.

- No fixed extent:
  - The range of a radio network can’t be easily determined and can vary substantially with changes in the environment.
  - Can’t use physical connectivity as a means of restricting access.

2 IEEE 802.11

Wireless NICs from various vendors (e.g. the Lucent “WaveLAN” ISA cards) have been available for quite a while. Until recently these cards generally used their own proprietary protocols and modulation and would not interoperate with cards from other vendors.

The popularity of wireless networking has increased recently with the introduction of IEEE 802.11 compliant devices. 802.11 is a standard for wireless networks, first standardised by the IEEE in 1997, then revised in 1999, which has led to the availability of interoperable radio NICs from many vendors. In particular wireless LANs have become popular with 802.11b, a refinement of the standard allowing higher bitrates, approved in late 1999.

There are some other competing standards for wireless networking. Notably the HomeRF standard also operates in the 2.4GHz ISM band and is aimed at similar applications to 802.11. However, it hasn’t become as popular as 802.11 and this paper won’t cover it further. Bluetooth (in the 2.4GHz band) and HiperLAN (in the 5GHz band) are aimed at somewhat different application domains to 802.11, and also won’t be covered here.

2.1 How standard is a standard?

In a number of ways the 802.11 standard isn't as standard as one might expect: although it specifies one MAC protocol, there are a number of ways of implementing 802.11 which might not be interoperable. First, 802.11 specifies several different possible physical layers (PHY):

- Frequency-Hopping (FH) spread spectrum radio in the 2.4GHz ISM band at 1 or 2 Mb/s
- Direct sequence spread spectrum (DSSS) radio in the ISM band at 1 or 2 Mb/s
- Infrared transmission at 1 or 2 Mb/s
- Orthogonal frequency division multiplexing (OFDM) radio in the 5GHz band at 6, 9, 12, 18, 24, 36, 48 or 54 Mb/s (added in the 802.11a standard)
- DSSS radio in the ISM band at 5.5 or 11 Mb/s (added in the 802.11b standard)

These PHYs are not interoperable with each other, with the exception that the 802.11b PHY can interoperate with stations using the older 1 or 2 Mb/s DSSS 2.4GHz PHY. From here on, we'll only be considering the 802.11b PHY, since it is the one that's become most popular recently.

There are more places in which 802.11 isn't entirely standard. First, a number of features in the MAC protocol aren't always used and may not be implemented by all stations. The standard does specify that stations all understand enough of the protocol to ensure interoperability. However that doesn't mean that every 802.11 device will support all 802.11's features, which could be important in particular situations. Finally, 802.11 does not specify the distribution system, a concept used in larger 802.11 networks as we'll see later. This can again lead to interoperability problems.

2.2 802.11 features

Unsurprisingly, 802.11 has quite a lot of similarities to other IEEE 802 standards, such as 802.3 (Ethernet[1]). However the 802.11 MAC (Medium Access Control) protocol has a lot of extra features to deal with the vagaries of wireless communication. These include:

- MAC level ACKs, retransmits and fragmentation (unreliable medium)
  - devices can retransmit frames corrupted by noise or break them into smaller chunks which have a better change of getting through
  - allows the protocol to appear reliable at the data link level
- co-ordination using CSMA/CA (hidden transmitter)
  - Carrier Sense Multiple Access with Collision Avoidance
  - hidden transmitter problem means collisions can't be reliably detected by a transmitter
  - MAC protocol uses delays and back-offs to reduce the chance of collisions
- RTS/CTS and virtual carrier sense (hidden transmitter)
  - stations can exchange short Ready-To-Send/Clear-To-Send sequence before exchanging data, to reserve the wireless medium for a period of time
  - this as a "virtual carrier sense" in the sense that when another station knows the medium is reserved it is treats it as though they could physically sense another transmitter
  - a station will consider the medium reserved if it receives a CTS even if it not see an RTS (e.g. because the sending station was out of range) and vice versa

[1] Well, actually Ethernet and 802.3 aren't quite the same thing, but we can ignore the difference for now

- infrastructure networks and the distribution system (DS) (hidden transmitter & mobility)
  - stations can relay transmissions via one or more wireless Access Points (APs)
- MAC level encryption (WEP) (no fixed extent)
  - RC4 based encryption of each frame transmitted
  - stands for "Wired Equivalent Privacy" supposed to provide a similar level of privacy and access control to that available on wired LANs
  - algorithm is flawed and has now been broken
- other esoteric features: PCF, etc...

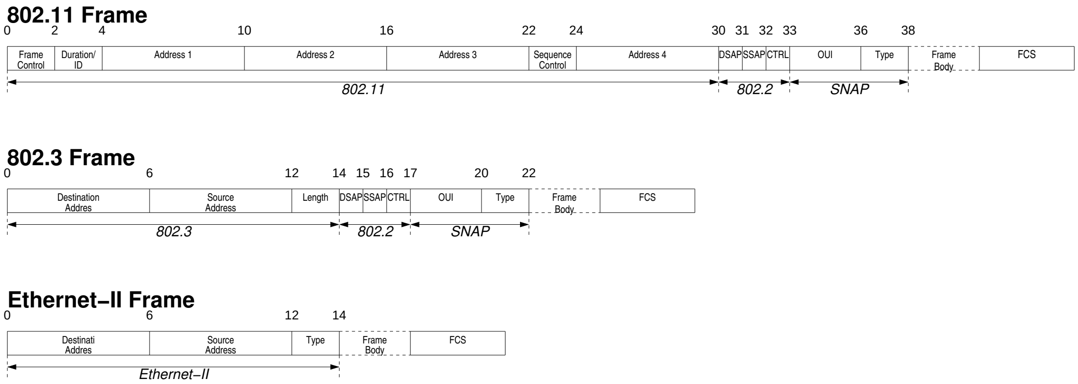

Figure 2: Ethernet vs. 802.11 frame format

All these additional possibilities make 802.11 a complex protocol. It adds a significant overhead meaning that at the application level 11Mb/s 802.11 is usually slower than 10Mb/s Ethernet. For example, Figure 2 shows the frame format for 802.11 along side that for 802.3 and Ethernet-II: the 802.11 MAC header, at 30 bytes is over twice as long as that for Ethernet and can contain up to four different MAC address. Once the logical link headers (802.2 and SNAP[2]) are taken into account, it's nearly three times the length of a plain Ethernet header.

2.3 Architecture of an 802.11 network

802.11 supports both "ad-hoc" and infrastructure modes of operation. The simpler ad-hoc mode requires nothing other than two or more stations in mutual range of each other. Infrastructure mode uses one or more Wireless Access Points (APs) to co-ordinate communications and supports the more complex features of the protocol.

2.3.1 Ad-hoc networks (aka the IBSS)

An ad-hoc network (also know as an Independent BSS or IBSS for reasons that will become clear later) is simply a collection of 802.11 stations which communicate with each other (see Figure 3). There is no special co-ordination or infrastructure. Because of this, problems are likely to occur if there are any hidden transmitters (i.e. any stations that are not within range of every other station). In this configuration stations send frames directly to the other stations in the IBSS.

2.3.2 Infrastructure networks

An infrastructure network contains one or more access points (APs) which co-ordinate communication. In the simplest case, shown in Figure 4 there is one AP. In this case all frames are relayed

[2] Strictly the SNAP header isn't necessary if using a protocol designed to operate directly over 802.2, but in practice it is nearly always used.

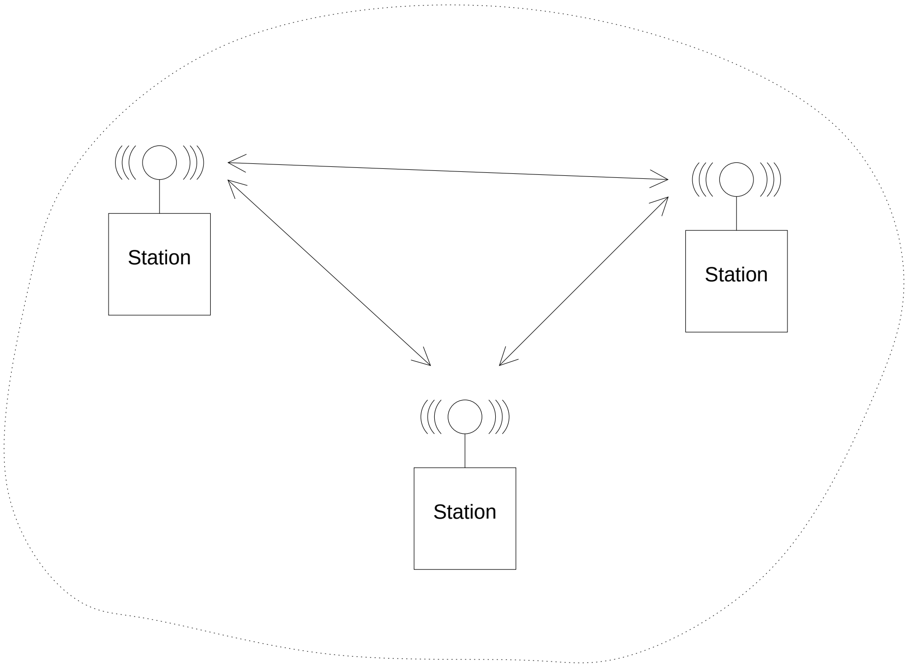

Figure 3: An Ad-hoc network (IBSS)

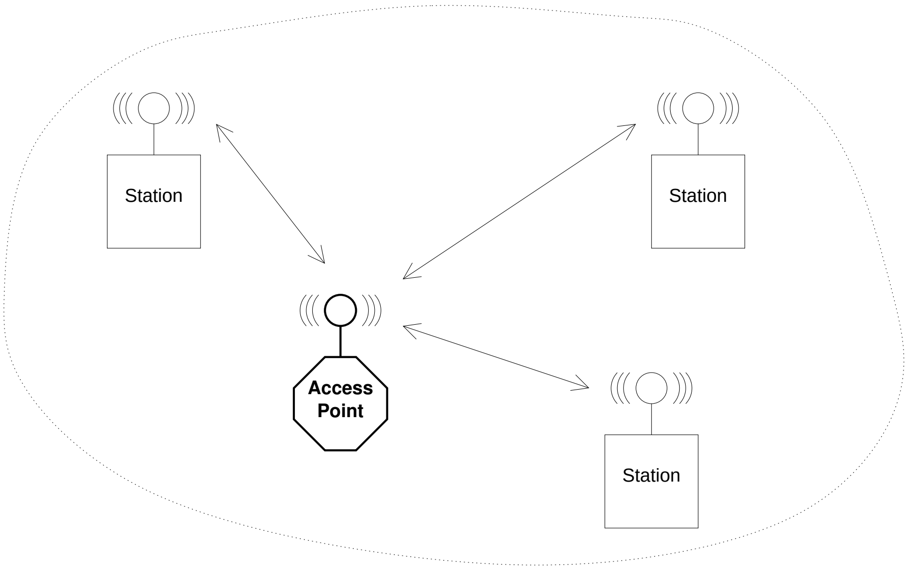

Figure 4: A simple infrastructure network (BSS)

via the AP, so stations do not have to within radio range of each other as long as they are each within range of the AP. For a station to join such a network it must first "associate" with the AP. An AP and its associated stations is called a Basic Service Set (BSS).

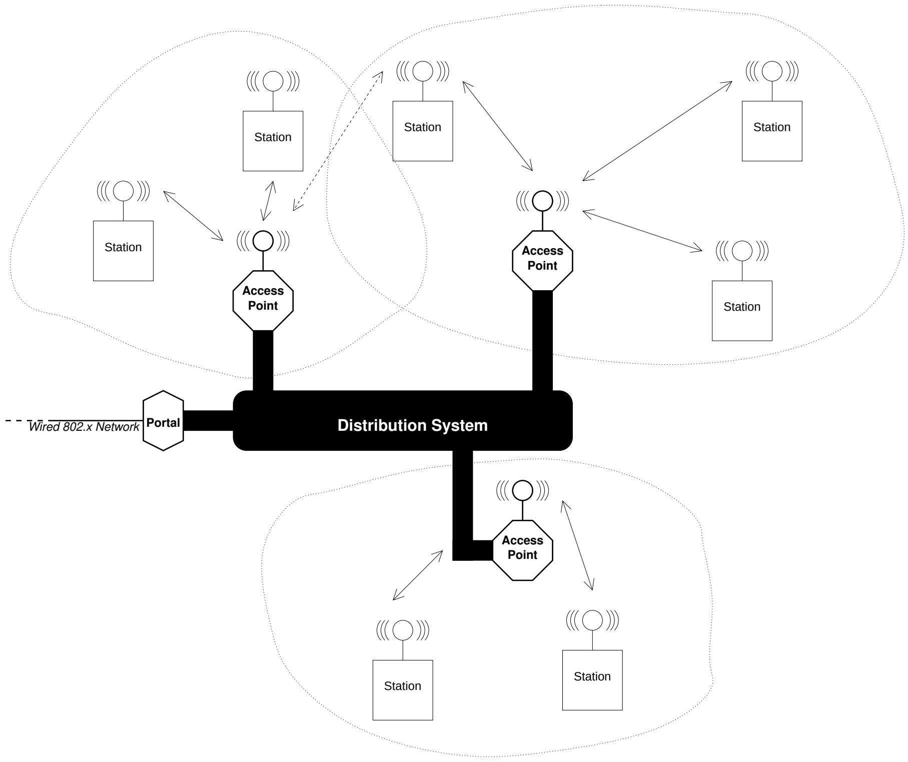

Figure 5: An extended infrastructure network (ESS)

A more complicated infrastructure network, such as that shown in Figure 5 can have multiple APs, connected by some medium known as the *distribution system* (DS). The collection of all the BSSes connected by a DS is called an Extended Service Set (ESS).

The nature of the DS is not specified by the 802.11 standard and could be a wired or a wireless network. Because the DS protocol is not specified, APs from different vendors will not necessarily interoperate to form a DS properly.

The BSSes in an ESS can be physically separated — this allows a single logical network to extend over a greater volume that the range of a single AP. Or, BSSes can overlap partially or completely — this allows an area to have multiple redundant APs. The protocol allows a station to move freely between the various BSSes in an ESS without losing connectivity, reassociating with a new AP if it becomes preferable to the current one (e.g. if the station moves out of range on one AP and into the range of another).

Also connected to the distribution system may be *portals* which bridge the wireless network to a wired 802.x or Ethernet network. In practice this function will nearly always be performed by one or more of the APs, rather than having a separate device. If the wired network is an 802.x network, then there will be one 802.2 logical link layer over the entire integrated (wired and wireless) network. In practice, of course, almost no-one uses 802.3 with an 802.2 link layer — instead they use plain Ethernet because, frankly, 802.2 isn't very useful. Nonetheless, much the same effect can be achieved by using a portal which bridges 802.11 to Ethernet, encapsulating and de-encapsulating as necessary.

It is also possible to have several different ESSes — i.e. distinct logical networks — overlapping in space. This is common when two organisations with their own wireless networks occupy adjacent rooms or buildings. Each ESS has an ID string which a station can use to select which network it wishes to join.

3 Current 802.11b devices

The most common (current) form for 802.11b NICs is as standard PCMCIA cards[3]. ISA and PCI devices are also available, but in most cases these consist of a PCMCIA card bundled with some sort of ISA/PCI to PCMCIA bridge adaptor.

Appliance style 802.11b Access Points are also common (e.g. the Apple Airport Base Station or the D-Link DWL-1000AP). These devices usually contain a 802.11b PCMCIA card along with a PCMCIA controller, an embedded controller and firmware. These can usually be configured to act as bridges or simple routers between wired and wireless networks. Some also have additional features such as including a modem and acting as a home gateway or dial-up server with simple firewall, NAT (aka "masquerading") router and DHCP server.

802.11b PCMCIA cards are available from many vendors, however most of these are just OEM rebadges of a few basic card designs. In fact the market is dominated by cards based on the *Prism 2* chipset and its descendents made my Intersil Corporation (formerly Harris Technologies). This has a number of components, including the low-level radio processors, but we're mostly interested in the MAC controller (the HFA3841, HFA3842, or ISL3873B depending on the generation) which is the component that software drivers communicate with.

The MAC controller is reasonably sophisticated, its firmware performing most of the basic 802.11 protocol handling. There are several somewhat different firmware variants in use, the four most common being those from:

Intersil Probably the most common, it is used in the cards from Netgear, D-Link, LinkSys, SMC and innumerable "no-brand" vendors.

Lucent/Agere/ORiNOCO Used in Lucent's "WaveLAN IEEE" and the later "ORiNOCO" cards and also rebadges such as the Cabletron (now EnteraSys) RoamAbout DS, and the IBM High Rate Wireless LAN.

Symbol Used in Symbol's Spectrum24 cards and also cards by Ericsson, Intel and 3Com.

Aironet Used in Cisco's "Aironet" cards. This firmware is quite different from the other three, which have a lot of similarities to each other.

4 Linux support

The majority of current 802.11b devices are supported under Linux, but the support is not entirely mature and consolidated. The Prism 2 chipset is so widespread that supporting 802.11b devices essentially amounts to supporting this chipset and its firmwares. The only significant exception is that some of the earlier Aironet (now Cisco) cards were based on a different MAC controller. Cisco's current products are Prism 2 based, although their firmware is rather different from most of the other firmwares.

There are quite a number of drivers which support at least some of the Prism 2 based cards, with quite a lot of overlap in functionality (see Figure 6). However, each driver tends to support a different subset of any particular card's functionality. Only a few drivers support some of the more specialised functionality of the wireless devices such as acting as an AP, or passively monitoring a wireless network. Not all the drivers are configured the same way, and there is no clearly "best" driver which supports a superset of the functionality of the others.

Unfortunately, as is so often the case, development of free software drivers is hampered by difficulty getting hardware and firmware specifications. Intersil will give information for open source development but require an NDA and Lucent/Agere seem to be extremely unwilling to give

[3] Oddly enough that's 16-bit PCMCIA rather than CardBus.

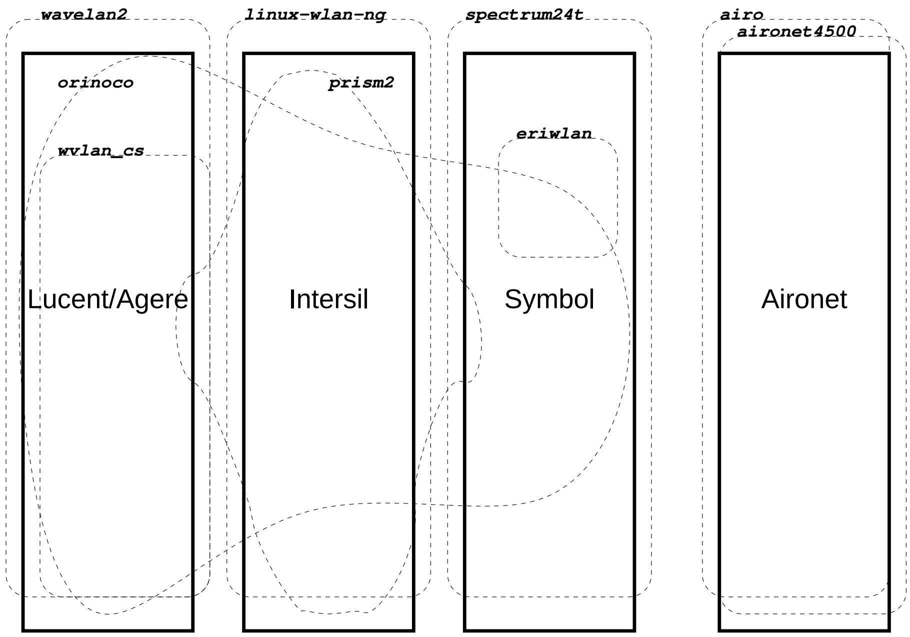

Figure 6: Linux driver coverage

out information on their firmware. The information which has been used to write the drivers has come from various places such as:

HCF-Light library Lucent have published a piece of source card for controlling the basic functions of their WaveLAN IEEE / ORiNOCO cards. This is a cut down version of the "HCF" library used both in their Windows driver and their binary-only Linux driver. The code is difficult to read and integrates poorly into the kernel, but provides useful information nonetheless.

Intersil Driver Programmer's Manual Intersil does not give out programming information without an NDA, however the terms of this NDA don't preclude writing an open source driver based on the information.

Other drivers Several drivers, in particular the `orinoco` driver have been written largely based on information deduced from reading other drivers.

Reverse engineering Always a fallback — poke the card until you figure out what it does. In fact there is just enough programming information available that there hasn't been very much of this, at least not so far.

4.1 The drivers

Because the Agere, Intersil and Symbol firmwares have quite a lot in common, there is a lot of overlap between the drivers that support cards based on these firmwares. A summary of the features of the following drivers is shown in Figure 7.

wvlan_cs One of the oldest 802.11b drivers for Linux, this was originally written by Andreas Neuhaus and links to Lucent's HCF-Light code for low-level functions. It supports Lucent/Agere firmware based Prism 2 cards, but is no longer actively maintained.

orinoco A driver, written (initially) by this paper's author, which attempts to support Intersil, Agere and Symbol firmware based Prism 2 cards. It is included in kernels since Linux 2.4.3, and in recent versions of David Hinds `pcmcia-cs` package.

Feature	orinoco (v0.08a)	prism2 (2001-09-27)	linux-wlan-ng (0.1.8-pre13)	wvlan_cs (1.0.7)	wavelan2 (6.10)	eriwlan (1.0)	spectrum24t (1.03)
Supports Lucent/Agere cards?	✓	—	—	✓	✓	—	—
Supports Intersil cards?	✓	✓	✓	—	—	—	—
Supports Symbol cards?	✓	—	—	—	—	✓[a]	✓
Free software?	✓	✓	✓	✓	—	✓	✓
Monitor mode?	—	✓	✓	—	—	—	—
Software AP?	—	✓	✓	—	—	—	—
Control 802.11 retransmits?	—	—	✓	—	—	—	—
Full 802.11 MIB exposed?	—	—	✓	—	—	—	—
In kernel?	✓	—	—	—	—	—	—
In `pcmcia-cs`?	✓	—	—	✓	—	—	—
Actively maintained?	✓	✓	✓	—	✓	✓	✓

[a] Ericsson WLAN only

Figure 7: Driver features

linux-wlan-ng A driver written by Mark S. Matthews with Intersil's support providing the most complete support for cards based on the Intersil firmware.

prism2 A driver written by Jouni Malinen (who also has Intersil documentation under NDA) which provides support for acting as an Access Point on Intersil cards and is much smaller and simpler than the `linux-wlan-ng` driver.

spectrum24t A driver for Symbol based cards which was originally written by TriplePoint and is now maintained by Lee John Keyser-Allen.

eriwlan A driver specifically for the Ericsson WLAN cards (Symbol based), written by Christian Olrog (an Ericsson employee).

wavelan2 A proprietary, binary only driver from Lucent for WaveLAN IEEE/ORiNOCO cards.

In addition to these drivers there are two main drivers available supporting the Aironet firmware cards (and also some of the earlier Cisco/Aironet cards based on a different MAC controller). These are:

airo_cs Written by Benjamin Reed, this supports all of the more recent Aironet and Cisco cards and is included in the `pcmcia-cs` package and the kernel.

aironet4500 Another driver, written by Elmer Joandi while employed by SpectrumWireless and recently released under the GPL, this is also included in the kernel source.

5 The `orinoco` driver

The `orinoco` driver was originally written out of frustration with various problems and limitations in the `wvlan_cs` driver. The `wvlan_cs` driver itself would have been difficult to fix because of its reliance on the Lucent HCF-Light code which is difficult to read and doesn't observe Linux kernel coding conventions. The driver's low-level functions were originally based loosely on the BSD `if_wi` driver, and the higher-level functions, such as wireless extensions support, on the `wvlan_cs` driver.

5.1 Driver structure

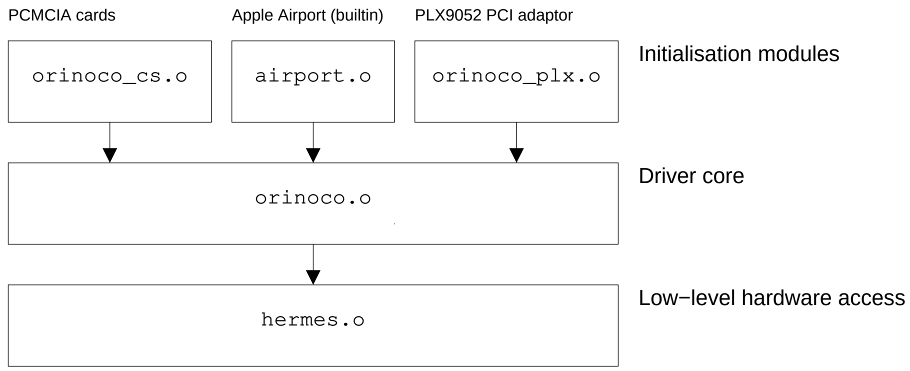

Figure 8: Structure of the `orinoco` driver

The `orinoco` driver now supports most Prism 2 cards based on the Lucent/Agere or Intersil firmware, and many based on the Symbol firmware. Not all features are supported properly, particular for the Intersil and Symbol firmware — more work on merging some of this support from other drivers is needed.

As well as the common PCMCIA cards it also supports the Apple Airport card built into some recent Macintosh computers (which is essentially an ORiNOCO card with a custom bus interface) and "PCI" cards based on the PLX9052 adaptor. The PLX chip is a limited PCI to PCMCIA bridge — because it is not a full PCMCIA controller it is not handled by the PCMCIA subsystem and needs a separate driver. A number of vendors (e.g. Netgear, LinkSys) provide PCI wireless NICs in the form of a PCMCIA card with one of these adaptors.

Because of the fairly wide range of hardware supported, the driver has an interesting structure, comprising 5 different modules, shown in Figure 8. Two core modules: `hermes.o` and `orinoco.o` handle low-level hardware access and network driver logic respectively. However, these don't actually contain any code to probe for or configure a device. That is provided by higher level "initialisation" modules: `orinoco_cs.o` contains code for a PCMCIA Card Services client to support PCMCIA cards (the most common), `airport.o` (written by Ben Herrenschmidt) contains code specific to the Apple Airport and `orinoco_plx.o` (written by Daniel Barlow) contains a driver for the PLX9052 adaptor. Each of the initialisation modules performs the necessary hardware probing and then initialises the `orinoco.o` module with the correct I/O and IRQ information.

6 Further work

As we've seen Linux's 802.11 support is somewhat immature at the moment. Below we look at the main areas in which work is needed to reach a better level of support.

6.1 Consolidation of drivers

While alternate implementations are usually a good thing in the free software world, the current amount of overlap between drivers is a bit excessive. Some work on consolidation is taking place: First, the `wvlan_cs` driver is now essentially deprecated in favour of the `orinoco` driver. Secondly, it's planned that the `orinoco` and `prism2` drivers will work towards each other, possibly leading to a complete merge eventually.

6.2 Protocol support — `net/80211`

At present, the 802.11 drivers encapsulate and de-encapsulate 802.11 frames so that they appear like Ethernet to the rest of the kernel. This simple approach means that existing kernel infrastructure for handling Ethernet can be re-used, but it's not ideal. While 802.11 has quite a lot of similarity to Ethernet, there are still significant differences. When using some of the more specialised features of 802.11, attempting to treat them as the same becomes an inconvenient limitation.

Having helper functions in the kernel for handling the 802.11 protocol, much like the existing functions for Ethernet support would have a number of advantages:

- easier to write drivers for new hardware
- 802.11 headers could be seen in `tcpdump`, `ethereal` etc., making it easier to debug MAC level problems
- possible to write a (mostly) driver independent implementation of a software Access Point and/or 802.11 portal

6.3 Configuration

Because of the many MAC protocol features, there are a lot more potential configuration options for 802.11 devices than for most other network adaptors. Most of the drivers don't allow the administrator to set all the potential parameters and the parameters that can be set aren't always set in the same way.

The Wireless Extensions, written by Jean Tourrilhes, are emerging as the main way of configuring wireless devices. This provides a set of `ioctl()` calls for setting wireless related parameters, and a `/proc` interface for retrieving signal quality information. There are some problems still with this interface though: the extensions also support a number of non-802.11 wireless devices and the parameters don't always map ideally onto 802.11. Furthermore they rely on `ioctl()` calls, which Linus has rightly decreed should be avoided and eventually eliminated.

Acknowledgements

Thanks must go to my employer, IBM, for their support of my work on this paper and the `orinoco` driver. Thanks also to Paul "Rusty" Russell for his advice on drafts of this paper, and to Jean Tourrilhes for answering a number of questions about 802.11 for me.

References

[1] IEEE, "IEEE Standards for Local and Metropolitan Networks: Overview and Architecture", IEEE Standard 802-1990.

[2] IEEE, "IEEE Standards for Local and Metropolitan Networks: Logical Link Control", IEEE Standard 802.2-1998.

[3] IEEE, "IEEE Standards for Local and Metropolitan Networks: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Standard 802.11-1999.

[4] IEEE, "IEEE Standards for Local and Metropolitan Networks: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Higher-Speed Physical Layer Extension in the 2.4 GHz Band", IEEE Standard 802.11b-1999.

[5] Tourrilhes, Jean, "Linux Wireless-HOWTO", `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/`.