CS441 Lab: IEEE 802.11 under Linux

Sarah Harvey

October 12, 2010

Contents

1. Introduction

Linux has matured to a point where nowadays most hardware is supported out of the box, and those hardware that aren't usually get a working driver within 6 months. However, one of the major points of contention is the issue of wireless connectivity. A number of issues make it hard to get decent wifi support in even the most mainstream distributions of Linux:

- Many hardware vendors refuse to provide free and/or open-sourced drivers
- Some of these vendors refuse to at least provide open hardware specifications so volunteers may write drivers; this causes legality issues as many drivers arose through reverse engineering of existing drivers
- Identification of the specific chipsets used in various WiFi NICs/dongles can be hard, as the company that produces the NIC/dongle for retail is often different from the chipset manufacturer, and some manufacturers like releasing different versions of the same model card using radically different chipsets [2]
- Manufacturer-provided drivers occasionally will insist on implementing their network stack instead of conforming to that found in the Linux kernel; this inhibits the ability to use GUI utilities to configure WiFi as a result
- Some manufacturer-provided drivers are occasionally closed-source, and thus are generally not supported by the Linux community due to the fact it creates unknown "blobs" in a kernel core dump that the community cannot debug

1.1 Overview of IEEE 802.11 in Linux

There appears to be an overwhelming portion of the population dedicated to finding and supporting various devices within Linux. Generally many of the drivers provided within the kernel have been written by volunteer developers, although there are a few that have been written by the manufacturers themselves. In addition to the driver, there are also various tools, utilities, and/or features one should be aware of to delve into this further. The entire Linux kernel is licensed under GPLv2[1].

At the time of this writing, the Linux wireless subsystem is in the midst of transitioning to a new subsystem

- Old Wireless Subsystem - `Wireless-Extensions`
  - Original specification written by Jean Tourrilhes of Hewlett-Packard in 1997[4]
  - Originally was a dirty hack in order to get a wireless device to be usable/configurable
  - In the process of being replaced by `mac80211` & `nl80211`[3]
- New Wireless Subsystem[3]
  - `cfg80211` - Basic Wireless Configuration API
    - Set to replace Wireless-Extensions
    - Still under development
    - Has built-in regulatory compliance
  - `mac80211` - Basic Wireless Framework for SoftMAC[1] wireless devices
    - Includes support for IEEE 802.11abgn, IEEE 802.11d, Roaming
    - Includes support for various wireless modes:
      - Access Point Infrastructure mode (BSS)
      - Station Infrastructure mode (BSS)
      - Monitor mode
      - Ad-Hoc mode (IBSS)
      - Wireless Distribution System (WDS)
      - Mesh (802.11s, WMN)
    - TODO: Roaming
    - Quality of Service
  - `nl80211` - 802.11 Netlink interface public header
    - userspace $\longleftrightarrow$ kernelspace wireless driver communication transport
    - Set to replace Wireless-Extensions
    - Still under development

Note that Wireless Tools and Wireless Extensions are slowly being phased out in favor of `cfg80211`, `mac80211` and `nl80211`; Wireless Tools will be replaced by the `iw` package, and Wireless Extensions will be replaced by `cfg80211/mac80211`. As such, this lab will primarily focus on the capabilities provided by `mac80211` as opposed to `Wireless-Extensions`.

[1] SoftMAC devices have the MLME managed in software, as opposed to FullMAC devices which have MLME managed in hardware

1.1.1 Wireless Tools

Wireless Tools[5] is a set of commands used to configure and set the options of wireless devices under Linux. It is currently present in most Linux distributions as part of the standard toolset, and is currently at version 29. Currently it is used to configure any device using Wireless Extensions (which is most of them). Commands in Wireless Tools include:

- `iwconfig` - Basic configuration of wireless adapter
- `iwlist` - Provides options for scanning for networks
- `iwspy` - Get per node link quality
- `iwpriv` - Manipulate parts of Wireless-Extensions (driver-specific)
- `ifrename` - Rename interfaces

1.1.2 `iw`

`iw` is the new wireless configuration utility under Linux based on the `mac80211/nl80211` framework. It is currently present in most newer, updated Linux distributions as part of a standard/optional toolset, and is currently at version 0.9.21. Like Wireless Extensions, it is used to configure any device using `mac80211/nl80211` (which is a decent amount of them).

`iw` relies on `libnl` (which implements `nl80211`) in order to gain access into and control the `mac80211` subsystem. Currently `libnl` is at version 1.1, and is actually used by other applications/libraries/services also, e.g. NetworkManager, `libpcap` (for Wireshark), and `wpa_supplicant`.

1.2 `mac80211`

The new wireless subsystem provides a number of capabilities and features not originally present with `Wireless-Extensions`. One of the fundamental ideas in the new subsystem is the separation of radio transceiver and the driver interface that is seen by the Linux kernel. These transceivers are affectionately referred to as `wiphy`s in the subsystem, whereas the driver interfaces are referred to has `dev`ices. This abstraction thus makes possible the idea of *virtual* wireless devices as seen by the Linux kernel, each able to be individually configured (to an extent), and thus treated in its own separate domain. All incoming IEEE 802.11 frames are forwarded from the `wiphy` to each of the `dev`s, and each of the `dev`s is able to access the `wiphy` in order to write an outgoing IEEE 802.11 frame.

2. Lab

2.1 Purpose

The purpose of this lab is to:

- Provide a brief overview of the current state of the wireless subsystem(s) under Linux
- Showcase the available command-line configuration and monitoring tools for wireless devices
- Walk through the process of configuring, compiling, and using the wireless dummy driver `mac80211_hwsim`
- Walk through the process of setting up a wireless Access Point using `hostapd`
- Demonstrate previous project that made use of this new wireless stack to measure channel switch latency (if time permits)

2.2 Materials

- Modern GNU/Linux distribution (kernel ¿= 2.6.30)
- Wireless device with `mac80211` support
- Access Point

2.3 Lab Exercises

2.3.1 Configuring wireless devices

There are now two core utilities to configure wireless devices by command-line. One is a suite of tools provided by Wireless Tools, the other is provided by a single command `iw`. We will be covering usage of both in this lab, using physical wireless devices in an attempt to connect to a wireless AP.

You should have been given either a USB wireless stick (or a laptop with a wireless card), and a liveCD with a specialized Linux distribution to use with your machine. Linux should already be up. All the following commands should be typed at a terminal, and you will compare the corresponding output to what is shown in this lab document.

Identifying your Wireless Device

Besides opening up your computer and checking the manufacturer/model of the chipset manually, there are a number of ways to identify what specific chipset is present in your wireless-capable device.

Generally just running `/sbin/lspci -nn` will allow you to identify what wireless chipset you have for onboard, PCI, and PCI-E devices:

```
03:00.0 Network controller [0280]:  RaLink RT2860 [1814:0781]
```

For USB devices, `/sbin/lsusb` will provide the full name of the device, however not the chipset:

```
Bus 002 Device 002:  ID 413c:8104 Dell Computer Corp.  Wireless 1450 Dual-band (802.11a/b/g)
```

For common/mainstream USB devices, more information can be gotten from `dmesg` as the kernel attempts to load the correct module for the device:

```
usb 2-1:  new high speed USB device using ehci_hcd and address 3
usb 2-1:  New USB device found, idVendor=413c, idProduct=8104
usb 2-1:  New USB device strings:  Mfr=1, Product=2, SerialNumber=0
usb 2-1:  Product:  Dell Wireless 1450 Dual-band (802.11a/b/g) USB 2.0 Adapter
usb 2-1:  Manufacturer:  DELL
usb 2-1:  configuration #1 chosen from 1 choice
usb 2-1:  reset high speed USB device using ehci_hcd and address 3
usb 2-1:  firmware:  requesting isl3887usb
phy2:  p54 detected a LM87 firmware
p54:  rx_mtu reduced from 3240 to 2376
phy2:  FW rev 2.13.24.0 - Softmac protocol 5.9
phy2:  cryptographic accelerator WEP:YES, TKIP:YES, CCMP:YES
phy2:  hwaddr 00:14:a5:58:db:50, MAC:isl3892 RF:Xbow
phy2:  Selected rate control algorithm 'minstrel'
Registered led device:  p54-phy2::assoc
Registered led device:  p54-phy2::tx
usb 2-1:  is registered as 'phy2'
usbcore:  registered new interface driver p54usb
```

If observed carefully, one will notice that the `p54usb` module was loaded in order to control this wireless device. Unfortunately, if the kernel fails to autodetect the device in order to load the proper module, one will have to do the corresponding research and find/compile the module manually.

Wireless Tools

Wireless Tools is a suite of tools used to examine wireless availability and configure wireless devices for connectivity. It is somewhat limited in its ability (due to needing to conform to the original specifications by Tourrilhes), but has been generally useful in its task.

1. Determine the network interface that is capable of wireless connectivity
   `iwconfig` is the general all-purpose wireless configuration tool. When run on its own, it will display all the possible interfaces that have wireless capability.

   Figure out the name of the wireless interface by pulling up a terminal, and running the following:
   `sudo iwconfig`

   Possible output:
   ```
   [sarah@kaylee cs441]$ sudo iwconfig
   lo no wireless extensions.

   eth0 no wireless extensions.

   eth1 IEEE 802.11abg ESSID:off/any
   Mode:Managed Access Point:  Not-Associated Tx-Power=0 dBm
   Retry long limit:7 RTS thr:off Fragment thr:off
   ```

```
Encryption key:off
Power Management:off

tun0 no wireless extensions.
```

2. Scan for a network to join
   `iwlist` is used generally to get more detailed information from a device about the wireless environment. Oftentimes it is used to search for access points to connect to.

   Using the information gathered earlier, run the following:
   `sudo iwlist <interface> scan`

   Possible output:
```
[sarah@kaylee cs441]$ sudo iwlist eth1 scan
eth1 Scan completed :
Cell 01 - Address:  00:40:96:49:44:72
ESSID:"SIUC-iv"
Protocol:IEEE 802.11b
Mode:Master
Frequency:2.412 GHz (Channel 1)
Encryption key:off
Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s
Quality=27/100 Signal level=-53 dBm
Extra:  Last beacon:  183ms ago
......
```

3. Connect to a network
   `iwconfig` can also be used to associated the wireless device with a particular access point. We are going to simply try to connect to an unencrypted network (SIUC-iv). Try connecting to SIUC-iv:
   `sudo iwconfig <interface> essid SIUC-iv`

   Running `iwconfig <interface>` should then show the access point that the device is associated to:

```
 [sarah@kaylee cs441]$ sudo iwconfig eth1
eth1 IEEE 802.11abg ESSID:"SIUC-iv"
Mode:Managed Frequency:2.412 GHz Access Point:  00:40:96:49:44:72
Bit Rate=1 Mb/s Tx-Power=20 dBm
Retry long limit:7 RTS thr:off Fragment thr:off
Encryption key:off
Power Management:off
Link Quality=64/70 Signal level=-46 dBm
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0
```

`iw`

`iw` is the new tool for configuration of wireless devices by interfacing through `nl80211` with `mac80211`. It offers a slightly new interface, and a bit more power in configuring the device.

The difference between iw and Wireless Tools is that iw makes use of an additional layer of abstraction between the physical device and the interface that is seen in the kernel.

1. Determine the network interface that is capable of wireless connectivity

   Run the following command to determine the primary interface you will be working on:
   `sudo iw dev`

   Possible output:
   ```
   [sarah@shhlinux cs441]$ sudo iw dev
   phy#1
   Interface wlan0
   ifindex 6
   type managed
   ```

   Here we see that we have an interface `wlan0` provided by the (wiphy) `phy1`. `wlan0` is therefore the primary interface.

2. Scan for a network to join

   Run the following command to scan for possible networks to join:
   `sudo iw dev <interface> scan`

   Possible output:
   ```
   [sarah@shhlinux cs441]$ sudo iw dev wlan0 scan
   BSS 00:40:96:49:44:72 (on wlan0)
   TSF: 11636938752439 usec (134d, 16:28:58)
   freq:  2412
   beacon interval:  100
   capability:  ESS ShortPreamble (0x0021)
   signal:  -85.00 dBm
   last seen:  2236 ms ago
   Information elements from Probe Response frame:
   SSID: SIUC-iv
   Supported rates:  1.0* 2.0* 5.5* 11.0*
   DS Parameter set:  channel 1
   WMM: * Parameter version 1
   u-APSD
   BE: CW 31-255, AIFSN 6
   BK: CW 31-255, AIFSN 2
   VI: CW 15-63, AIFSN 1, TXOP 6016 usec
   VO: CW 7-127, AIFSN 1, TXOP 3264 usec
   ....
   ```

3. Connect to a network

   Run the following command to connect to SIUC-iv:
   `sudo iw <interface> connect SIUC-iv`

   This may take a while, so wait around a minute before running the following to check to see if the association was successful: `sudo iw dev <interface> station dump`

   ```
   [sarah@shhlinux Downloads]$ sudo iw dev wlan0 station dump
   Station 00:40:96:49:44:72 (on wlan0)
   inactive time:  15173 ms
   rx bytes:  33460
   ```

```
rx packets:  245
tx bytes:  612
tx packets:  7
signal:  -54 dBm
tx bitrate:  1.0 MBit/s
```

4. Adding another virtual monitoring device
   The wonderful thing about `mac80211` is that it allows for the possibility of virtual interfaces interacting with the physical device. To demonstrate, we are going to create an 802.11 frame monitoring device, and monitor the output using dumpcap or wireshark. Remember that phy designator we saw earlier? We will be using that in order to add an additional virtual device.

   Run the following command to add an additional monitoring device:
   `sudo iw phy <wiphy> interface add mon0 type monitor`

   You can check to verify that this worked by running the following:
   `sudo iw dev`

   Possible output:
   ```
   [sarah@shhlinux cs441]$ sudo iw dev
   phy#2
   Interface mon0
   ifindex 8
   type monitor
   Interface wlan0
   ifindex 7
   type managed
   ```

   Now we can examine the frames that are being seen by our monitoring interface by loading up wireshark.

2.3.2 Compiling the wireless dummy driver `mac80211_hwsim`

Having a brief overview of how to very simply set up and configure wireless devices, we are now going to try to compile and run the wireless dummy driver

Bibliography

[1] Free Software Foundation, Inc. GNU General Public License, version 2. http://www.gnu.org/licenses/gpl-2.0.html, 1991.

[2] Nicolai Langfeldt. A survey of Linux and WiFi. http://users.linpro.no/janl/hardware/wifi.html, April 2006.

[3] Linux Wireless. http://wireless.kernel.org.

[4] Jean Tourrilhes. Wireless Extensions for Linux. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html, January 1997.

[5] Jean Tourrilhes. Wireless Tools for Linux. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html, August 2008.