Use of Raspberry Pi as Wireless Access Point

Objective

The goal of the this project was to create a wireless access point using a given USB wireless access point (in this case, the Realtek rtl8191cu based dongle) and a Raspberry Pi Model B running Raspbian.

Implementation and Results

I began by attempting to follow the guidelines provided by a how-to guide at http://elinux.org/RPI-Wireless-Hotspot. Despite my attempted implementation according to these instructions, due to the fact that the wireless dongle I had procured did not have a driver compatible with the Raspberry Pi, that implementation was unsuccessful, in the end. With what I now knew from the previous, incompatible, tutorial, I removed the changes that had been made as a result of the previous tutorial and began by implementing the dhcp server, as that would be necessary for any router. I followed the instructions found at http://www.myguitars.mine.nu/images/rpi_raspbianwheezy_dhcp_server.pdf.

I began by downloading a dhcp server using apt-get.

```
sudo apt-get install isc-dhcp-server
```

Once the server is installed, the config file, /etc/dhcp/dhcpd.conf, must be edited:

```
option domain-name “Dude_the_cat.net”;
option domain-name-server 209.18.47.61, 8.8.8.8, 8.8.4.4
subnet 192.168.3.0 netmask 255.255.255.0{
    range 192.168.3.1 192.168.3.20
    option routers 68.172.218.115
}

host RPI_Router{
    harware ethernet b8:27:eb:df:d1:ea;
    fixed-address 192.168.3.0;
}
```

In this case, we assign the domain name as Dude_the_cat. The domain-name-server values are the ISP's DNS server and public Google DNS servers, respectively. The subnet value will be the local IP address of the router, itself. The range identities what possible local IP values the router will assign to machines connecting through the router. The option router refers to the actual IP address assigned by the ISP. The hardware ethernet is the MAC address of the Raspberry Pi, and the fixed address will be the static IP of the Raspberry Pi while it is acting as a wireless router.

The server can be run with the command:

```
service isc-dhcpd-server start
```

and can be stopped with the command:

```
service isc-dhcpd-server start
```

With the server installed and configured, I moved onto the task of getting the seemingly incompatible wifi dongle to work with the Raspberry Pi (RPi).

As mentioned, the wifi dongle being used for this project is the Realtek rtl8192cu. Though the dongle did come with an CD which contained the driver software, a version compatible with the RPi's processor, the ARM1176JZFS, was not included. Instead, the driver .gz file was obtained from the manufacturer's website using the links program.

Per the driver's extensive documentation, the driver source tar ball needed to be decompressed.

```
tar zxvf rtl8188C_usb_linux_v4.0.2_9000.20130911tar.gz
```

Once decompressed, within the driver folder, the existing Makefile must be edited. Though already populated with a list of possible processors with default settings, the RPi's processor is, unfortunately, not part of the Makefile, by default.

Within the Makefile, following the list of processor platforms, we add:

```
CONFIG_PLATFORM_ARM_1176JZFS = y
```

Then, within the platform settings section, we create a corresponding entry:

```
ifeq ($(CONFIG_PLATFORM_ARM_1176JZFS),y)
EXTRA_CFLAG+=CONFIG_LITTLE_ENDIAN
ARCH:= arm
CROSS_COMPILE:=
CONFIG CPU_32v6:=y
KSRC:= /lib/modules/3.6.11+/built
```

```
MODDESTDIR:= /lib/modules/3.6.11+/kernel/drivers/net/wireless
MODULE_NAME:=wlan
endif
```

The Makefile is then saved. The values assigned to each of the fields indicated above each had to be researched to be sure that they were correct, given the RPi's processor and the Raspbian operating system, with the CONFIG CPU, MODDESTDIR and MODULE_NAME fields not initially listed in the manufacture's default configuration, and only being added after some further research to deal with issues later arising with a discrepancy between the chip architecture version and the assembly command contained within one of the driver files.

The Makefile, located at /lib/modules/3.6.11+/kernel/drivers/net/wireless then had to be appended, per the manufacture's documentation, with the text:

```
obj-$(CONFIG_RTL8192CU) += rtl8192cu/
```

The driver folder, which was decompressed, must be moved from the location of decompression to the kernel, specifically a new folder /lib/modules/3.6.11+/kernel/drivers/net/wireless/rtl8192cu .

Per the manufacturer documentation, a new file, /lib/modules/3.6.11+/kernel/drivers/net/wireless/Kconfig was created containing the line

```
source “/lib/modules/3.6.11+/kernel/drivers/net/wireless/rtl8192cu/Kconfig”
```

With the manufacturer's instructions complete, I attempted to make the module by simply running the command

```
sudo make
```

while in the rtl8192cu folder. The initial attempt proved to be unsuccessful, with an error generated regarding the setting of the chip type as little or big endian, despite what appears to be a setting of this option within the driver's Makefile. Following some attempted editing of the Makefile, to perhaps fix a formatting error, repeated attempts to make the module continued to fail. The next step was to investigate where in the driver files the value of little endian was expected. The check of this variable was eventually found within the autoconf.h file in the driver's include file. Here, I hard coded the endian type as

```
#define CONFIG_LITTLE_ENDIAN 1
```

The file was saved and I attempted another make of the module. It was, again, unsuccessful. After searching for possible resolutions to the compiler error presented, it appeared that the problem was with the kernel itself. The recommended solution was to download a new clean kernel source tree. Again, using the Links program, via the command line, I was able to

download a clean kernel source tree from:

http://raspberrypi.stackexchange.com/questions/5230/getting-no-rule-to-make-target-modules-stop-error-when-running-makefile-w

Once downloaded to /usr/src/linux-rpi-3.6.y.tar.gz, the files were extracted with

```
sudo tar -xzf /usr/src/linux-rpi-3.6.y.tar.gz
```

The extraction was somewhat lengthy, time-wise. Once extracted, the directory should be renamed to /linux-rpi-3.6.11+ (this was discovered after several failed makes). The new files are then linked with the existing kernel:

```
ln -s /usr/src/linux-rpi-3.6.11+ /lib/modules/3.6.11+/build
ln -s /usr/src/linux-rpi-3.6.11+ /lib/modules/3.6.11+/src
```

A copy of the current kernel config is then copied into the new directory:

```
cp /proc/config.gz /usr/src/linux-rpi-3.6.11+
cd /usr/scr/linux-rpi-3.6.11+
gunzip config.gz
mv config .config
make oldconfig && make prepare
```

While this improved the compilation of the the rtl8192cu Makefile, errors were still reported, specifically:

```
WARNING: Symbol version dump /usr/src/linux-rpi-3.6.11+/Module.symvers is missing; modules will have no dependencies and subversions
```

While many discussion boards, when faced with this same problem, wrote off this error as irrelevant, since the compilation appears to continue despite this error message, later makes suggested that such an error could not go unresolved. To fix this error message, I ran

```
make modules
```

in the /src/usr/linux-rpi-3.6.11+ directory. This make takes several hours to complete on the RPi. Once it is compete, a new file Module.symvers is created within the rtl8192cu file. Another run of the make command in the rtl8192cu directory now occurs without the missing

symvers message.

An error of

```
/bin/sh: script/mod/modpost: not found
```

was observed. According to a post found at http://vishnulinux.wordpress.com/2012/04/30/binsh-scriptsmodmodpost-not-found/, a

```
make modules_prepare
```

command run in /usr/src/linux-rpi-3.6.11+ would clear this error. When another module compile was attempted, the error was resolved.

On several occasions, the rtl8192cu directory had to be completely recreated from the original .gz file, since the partial makes followed by edits made to the kernel or other files meant starting over was the only option. Eventually, though, a module was successfully created when the wlan.ko file appeared in the rtl8192cu folder following a successful make. Unfortunately, despite repeated attempts to edit the Makefiles of both the driver and the clean kernel, each attempt to load the module using the

```
insmod wlan.ko
```

command only resulted in the following error:

```
ERROR: could not insert module wlan.ko: Invalid module format
```

As of yet, this error, and its exact reasoning has not been resolved, despite extensive online searches and repeated attempts to change the kernel settings, short of changing the kernel version completely.

Were this error to be resolved, the remainder of the project would be relatively straight forward. Likely a shell script to allow for the updating of the ISP assigned IP address would be required to provide the dhcp server with the ability to connect to any network. A daemon could allow for the automatic start up of the server with Raspbian.

Conclusion

While this project appeared as though it would be very simple to implement, given the basic pieces required, and abundant documentation of this type of usage of the Raspberry Pi

(for whatever reason), the actual implementation proved to be quite a bit more challenging. The challenge resulted, almost entirely, from the lack of readily available drivers/modules for these two pieces of hardware (the RPi and rtl8192cu). Because of this, what began as a project to create a wifi router became a project to create a working module, thereby making the rtl8192cu compatible with this particular version of Raspbian on this particular version of the Raspberry Pi. Through this process, I became much more familiar with the Linux command line environment, leading to speed and comfort ability, perhaps even approaching confidence. As this was my first experience working with Linux, gaining a basic familiarity with the terminology and commands use within the bash Linux environment was a learning experience in itself. I became very familiar with the structure of this particular OS, and can now say that I somewhat understand what the kernel encompasses. Prior to this project, I was unaware that a driver and a module are one in the same, or how to move, copy, delete, rename, compile, unzip (to name a few) files in the Linux command line. This project offered a chance to become familiar with the process of using SSH to access the command line of another machine and to better understand some of the basics of the addressing system used on a local network and within a dhcp server. The project also offered a chance to use the project collaboration and file storage website, GitHub.com.

Given enough time, this project may have resulted in a functional module to support this very specific dongle with the RPi, though given the fact that compatible dongles are available for the RPi at relatively low prices, a continuation of this project would serve only to identify what the specific trouble was with the compatibility of this module with this particular version or Raspbian. While this debugging could, theoretically, prove useful in the long term as practice for debugging other Linux modules, the use of this particular module would be, no doubt, limited and primarily instructive in the workings and structure of this kernel and perhaps Linux kernels, more broadly. It has, in some ways, piqued my curiosity in this seemingly wide-open operating system, but this particular project likely does not have a functional future.

Bibliography

- http://elinux.org/RPI-Wireless-Hotspot
  [accessed on 10/14/2013]

- Realtek 8192cu driver from http://www.realtek.com.tw/downloads/
  [accessed on 11/1/2013]
- isc-dhcp-server from apt

- http://www.myguitars.mine.nu/images/rpi_raspbianwheezy_dhcp_server.pdf
  [accessed on 11/15/2013]

- Kernel source tree for linux 3.6.11
  http://raspberrypi.stackexchange.com/questions/5230/getting-no-rule-to-make-target-modules-stop-error-when-running-makefile-w
  [accessed on 11/3/2013]

- http://www.mjmwired.net/kernel/Documentation/kbuild/modules.txt
  [accessed on 11/30/2013]

- http://unix.stackexchange.com/questions/24704/how-to-generate-module-symvers
  [accessed on 12/1/2013]