

802.11 Wireless Network Access at Swinburne Using FreeBSD 5.3

CAIA Technical Report 041221A

[Jason But](#)

December 21st, 2004

Introduction

[FreeBSD](#) is a UNIX oriented operating system which is also the default desktop Operating System (OS) for many at the [Centre for Advanced Internet Architectures \(CAIA\)](#). While networking is a key feature of FreeBSD and other UNIX type OS's, support for integrated 802.11 wireless networking on laptops is problematic. I have recently managed to set up my laptop ([DELL Latitude D505](#)) to:

- Support the Wireless network device on this laptop.
- Connect and disconnect to available wireless Access Points
- Connect to the Swinburne Wireless Network through the Cisco VPN Gateway

In this report I document the steps I took. While your laptop may offer a slightly different configuration, or your wireless network is not the same as that provided by [Swinburne University](#), many steps documented in this report will still be applicable.

This document is broken down into the following sections:

- Enabling the Wireless Network Card driver for FreeBSD 5.3
- Connecting to and associating with Wireless Access Points.
- Connecting to a VPN protected Wireless Network.

Wireless Network Device Drivers for FreeBSD 5.3

[FreeBSD](#) only supports a handful of Wireless Network Devices directly. However a separate project - [Project Evil](#) - (which has recently been added to the standard FreeBSD 5.3 distribution) allows the use of existing Microsoft Windows Wireless Network Card drivers in a FreeBSD environment. As per the FreeBSD Networking [Documentation](#):

Unfortunately, there are still many vendors that do not provide schematics for their drivers to the open source community because they regard such information as trade secrets. Consequently, the developers of FreeBSD and other operating systems are left two choices: develop the drivers by a long and pain-staking process of reverse engineering or using the existing driver binaries available for the Microsoft® Windows platforms. Most developers, including those involved with FreeBSD, have taken the latter approach.

Thanks to the contributions of Bill Paul (wpaul), as of FreeBSD 5.3-RELEASE there is "native" support for the Network Driver Interface Specification (NDIS). The FreeBSD NDISulator (otherwise known as Project Evil) takes a Windows driver binary and basically tricks it into thinking it is running on Windows. This feature is still relatively new, but most test cases seem to work adequately.

The most difficult step in generating the `if_ndis` kernel module is locating the original Windows drivers. You need:

- Windows XP driver binary (.SYS extension)
- Windows XP driver configuration file (.INF extension)
- Any other files required by the Windows XP driver binary

To generate the **if_ndis.ko** Kernel Module, you need to copy all the necessary file into the `/sys/modules/if_ndis` directory and then in that directory execute:

```
ndiscvt -i W32DRIVER.INF -s W32DRIVER.SYS -o ndis_driver_data.h
make
make install
```

NOTE: It is necessary to run the original BSD make, GNU make will not succeed with the pre-existing Makefile

Project Evil constructs a FreeBSD Kernel Module that wraps the existing Microsoft Windows drivers, translating between the Windows NDIS network interface and the FreeBSD ifconfig interface. If there are other files required by the Windows driver, then there are two options:

1. Copy the other files to `/compat/ndis`, they will be loaded as required by the **if_ndis** driver. However, this will only work IF the kernel module is loaded AFTER the system has started and the filesystem has been fully mounted.
2. Use the **ndiscvt** with the `-f` option to generate a shared module that can be loaded during system boot. For further information see `man ndiscvt`

Once the kernel module for your Wireless device has been installed, it is a simple matter of loading the kernel modules `ndis` and `if_ndis` to enable the Wireless Network device.

Specific steps for the Dell Latitude D505

The DELL Latitude D505 laptop computer has an Intel(R) Pro/Wireless 2200BG Network Device. The Windows XP Drivers provided by DELL a supplied as a single executable to install all required files. I found a ZIP file containing the required files at http://www.powernotebooks.com/Support/intel_2200_wlan.zip, copy the files `w22n51.sys` and `w22n51.INF` to `/sys/modules/if_ndis` and execute:

```
ndiscvt -i w22n51.INF -s w22n51.sys -o ndis_driver_data.h
make
make install
```

This driver requires two other files for error reporting purposes, the driver functions perfectly without these extra files but as per Option 1 above, the must be loaded AFTER the system has started.

The driver functions as two loadable kernel modules, the first `ndis.ko` can be loaded during the boot phase of system startup - to do this add the following line to `/boot/loader.conf`

```
ndis_load = YES
```

It is also possible to specify loading the second kernel module within `/boot/loader.conf` by adding the following line, however this will only work IF the Windows XP Driver for your wireless device does not require any files other than the corresponding *.SYS and *.INF files. In the case of the Dell Latitude D505, there are other files and the `if_ndis.ko` kernel module cannot be loaded from within `/boot/loader.conf`.

```
if_ndis_load = YES
```

Once the system has started, you can load the second kernel module `if_ndis.ko`, this must be done as `root`. To do this execute the command:

```
kldload if_ndis
```

Once loaded, the `if_ndis` kernel module will create the `ndis0` network device, this is what we will configure and use to access the Wireless network.

Configuring the Wireless Network Device

In this section we assume that you have compiled and installed the Project Evil kernel modules AND that both kernel modules (`ndis` and `if_ndis`) have been loaded.

Enabling/Disabling the Wireless Network Device

The `ndis0` device is enabled by executing (as `root`) the command:

```
ifconfig ndis0 up
```

The device is disabled by executing (as `root`) the command:

```
ifconfig ndis0 down
```

Obtaining a list of available Access Points

This task need not be performed as root. To obtain a list of available Access Points, execute the command:

```
wicontrol -i ndis0 -l
```

This will list all available access points, the signal strength, whether or not they are WEP protected and a host of other useful information.

Associating with an Open Access Points

To associate to an Access Point with the Network ID (or SSID) of `Net_ID`, execute (as `root`):

```
ifconfig ndis0 nwid Net_ID
```

To access a WEP protected Access Point, issue the command:

```
ifconfig ndis0 nwid Net_ID wepkey wep_key
```

Obtaining an IP Address on a Wireless Network

Once associated with an Access Point, it is necessary to configure an IP Address on the network device so that the system can access the network resources. There are two options:

1. Use a fixed IP Address.

Configure the `ndis0` device with the `ifconfig` command with a static (fixed) IP address. It will also be necessary to configure the routing tables using `route` so that IP traffic is properly routed. This information will be available from your system administrator.

2. Use a dynamically assigned IP Address.

Use DHCP to be allocated a free IP Address, this will also set up the routing tables on your system. Details of this option are outlined below.

With FreeBSD 5.3, DHCP on a network device is configured using the `dhclient` daemon - this program must be executed as `root`. When executed, the `dhclient` will attempt to obtain an IP address for the Wireless Network Device, configure the routing table to properly route IP traffic over the Wireless Connection, and modify network address resolution information stored in `/etc/resolv.conf`. There are steps that must first be taken before we allow DHCP configuration:

- **Backup `/etc/resolv.conf`** - The `dhclient` application will overwrite the `/etc/resolv.conf` file, we need to back it up so that we can restore it when the network connection is brought down.
- **Backup and delete the default route** - We need to remember the current default route so that it can be re-applied after the network connection is brought down. We also need to delete the current default route so that the `dhclient` application can successfully program a new default route.

A useful set of instructions to execute are:

```
cp /etc/resolv.conf /etc/resolv.conf~  
route -n get default  
route delete default
```

Remembering to note the gateway value reported by the second command.

Once these commands are executed, we can execute (as root):

```
dhclient ndis0
```

This will obtain an IP address for the Wireless device and update the routing tables.

NOTE: A new default route will only be programmed IF there is NO current default route, hence the command to delete the current default route

When we wish to disconnect from the wireless device we need to first kill the `dhclient` daemon (PID stored in `/var/run/dhclient.pid`), and then execute the following commands (where gateway is the value obtained earlier):

```
cp /etc/resolv.conf~ /etc/resolv.conf
route delete default
route add default gateway
```

Automating Wireless Device Configuration

This is a lot of commands to issue to use our wireless device, I would like to simplify the task somewhat. Unfortunately, while a handful of Wireless Network Management GUI Tools have been written for KDE, they access the network device using the Linux Wireless Extensions and currently only work with Linux installations. Instead, I have developed the following script which should be placed in `/usr/local/etc/rc.d/wireless.sh`. Being in this location, it is executed with the parameter `start` at the end of the system boot phase and with the parameter `stop` when system shutdown is initiated. The script takes four possible parameters, the descriptions follow the script presentation. Finally, the script must be executed as `root`, it will fail with an error message if run by a regular user.

```
#!/bin/sh

#####
# Binary programs used by this script.                                     #
#####
IFCONFIG=`which ifconfig`
WICONTROL=`which wicontrol`
DHCLIENT=`which dhclient`
NETSTAT=`which netstat`
ROUTE=`which route`
DIALOG=`which dialog`
KLDLOAD=`which kldload`
KLDUNLOAD=`which kldunload`

#####
# The network device name and the location of run-time storage.         #
#####
NETDEV="ndis0"
DB_DIR="/var/run/wireless"

#####
# Some parameters used for the User Interface.                           #
#####
DLG_TITLE="Wireless Connection Manager"
DLG_DIMENSIONS="20 50"

#####
# Management of the Wireless Network Device.                             #
#                                                                           #
# There are four possible options to run this script, they are listed below #
# and parsed as a command line option. Finally, this script may ONLY be run #
# as root, a regular user attempting to run this script will receive an error #
# message.                                                                    #
#####
if [ `id -u` -ne "0" ]; then
    echo "ERROR: You appear to be `id -un` but you need to run this script as root"
    exit 1
fi

#####
# Create the run-time storage directory if it doesn't exist.             #
#####
mkdir -p $DB_DIR

#####
# wireless.sh start                                                         #
#                                                                           #
# Called by the system during system start up. We load the if_ndis kernel #
# module that will create the ndis? device. The device is created as being #
# "down", it can be brought up by executing this script with the "connect" #
# option.                                                                      #
#####
case "$1" in
start)
    echo "Creating Wireless Network Device ($NETDEV)"
    $KLDLOAD if_ndis > /dev/null
    ;;

```

```

#####
# wireless.sh connect
#
# The user wishes to bring up the wireless network device AND connect to a
# wireless network. We do the following:
# - Enable the wireless network device ndis?
# - Obtain a list of available access points.
# - Display the list and allow the user to make a selection of which access
# point to associate with.
# - If there are no access points, or the user cancels the selection, bring
# down the network device and exit.
# - Associate the network device to the selected Access Point.
# - Backup the existing /etc/resolv.conf file so it can be restored later.
# - Backup existing default route information so that it be later restored.
# - Run dhclient on the network device to obtain an IP address and set basic
# routing information.
# - Get all routes from the route table associated with the wireless network
# device and display them to the user.
#####
connect)
INFO="Bringing up Wireless network device - $NETDEV..\n"
$DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS
$IFCONFIG $NETDEV up

#####
INFO="$INFO\nScanning for available connection points"
$DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS

for NWID in `$WICONTROL -i $NETDEV -l | grep netname | awk '{print $4}' | sort | uniq`
{
    MENULIST="$MENULIST $NWID Open_Wireless_Access_Point"
}

$DIALOG --title "$DLG_TITLE" \
--menu "Please select the Access Point you would like to attach to:" \
$DLG_DIMENSIONS 12 \
$MENULIST \
2> tempfile

#####
if [ $? -ne 0 ]; then
    $IFCONFIG $NETDEV down
    $DIALOG --title "$DLG_TITLE" \
--msgbox "No connection established, application terminated." $DLG_DIMENSIONS
    exit 0
fi
ACCESS_POINT=`cat tempfile`

#####
INFO="Connecting to Access Point - $ACCESS_POINT\n"
$DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS
$IFCONFIG $NETDEV nwid $ACCESS_POINT

INFO="$INFO\n - Associated $NETDEV with $ACCESS_POINT"
$DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS

#####
cp /etc/resolv.conf $DB_DIR/resolv.conf
INFO="$INFO\n - Backed up /etc/resolv.conf"
$DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS

#####
echo "default `route -n get default | awk '/gateway:/ { print $2 }`'" > $DB_DIR/def_route
INFO="$INFO\n - Backed up route (`cat $DB_DIR/def_route`)"
$DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS

#####
$ROUTE delete default
$DHCLIENT $NETDEV
INFO="$INFO\n - DHCP negotiated on $NETDEV\n - Updating Routing table"
$DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS

#####
INFO="$INFO\n\nNew routes added"
INFO="$INFO\n`$NETSTAT -nrf inet | grep $NETDEV | awk '{ print $1, $2 }`'\n"
INFO="$INFO\nConnected! Press to continue"
$DIALOG --title "$DLG_TITLE" --msgbox "$INFO" $DLG_DIMENSIONS
;;

#####
# wireless.sh hangup
#
# - Kill the dhclient daemon running on the network device.
# - Delete all routes associated with the wireless network device.
# - Restore the original default route.
# - Restore the original /etc/resolv.conf
# - Unload and then reload the if_ndis kernel module, this clears any settings
# from the current session that may impact on future settings.
#####
hangup)
INFO="Hanging up current wireless connection\n"

if [ -e /var/run/dhclient.pid ]; then
    INFO="$INFO\n - Killing DHCP service"
    $DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS
    kill -9 `cat /var/run/dhclient.pid`
fi

#####
if [ -s $DB_DIR/def_route ]; then

```

```

        INFO="$INFO\n - Restoring route (`cat $DB_DIR/def_route`)"
        $DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS
        $ROUTE delete default > /dev/null 2>&1
        $ROUTE add $(cat "$DB_DIR/def_route")
    fi

#####
if [ -e $DB_DIR/resolv.conf ]; then
    INFO="$INFO\n - Restoring /etc/resolv.conf"
    $DIALOG --title "$DLG_TITLE" --infobox "$INFO" $DLG_DIMENSIONS
    rm /etc/resolv.conf
    mv /$DB_DIR/resolv.conf /etc/resolv.conf
fi

#####
rm -fR $DB_DIR/*

#####
/sbin/kldunload if_ndis
;;

#####
# wireless.sh hangup
#
# The system is going down, we need to restore some backed up information so
# that it is available the next time the system boots. Routing tables are not
# relevant and will be reset at next system boot time, however we need to:
# - Restore the original /etc/resolv.conf
# - Remove any database files from $DB_DIR
#####
stop)
    if [ -e $DB_DIR/resolv.conf ]; then
        echo "Restoring pre-Wireless Network Device system configuration files..."
        rm /etc/resolv.conf
        mv $DB_DIR/resolv.conf /etc/resolv.conf
        rm -fR $DB_DIR/*
    fi
    ;;

#####
# Default case, print usage information for the shell script.
#####
*)
    echo "Usage: `basename $0` {start|stop|connect|hangup}" >&2
    ;;
esac

exit 0

```

wireless.sh start

Executed during system boot. Simply loads the `if_ndis` kernel module, thereby creating the `ndis0` device. This script will always be executed AFTER the filesystem is mounted and so there will be no problems with loading the kernel module. If you are able to load your `if_ndis` kernel module with `/boot/loader.conf` without an error, the `kldload` line is not required in this script.

wireless.sh connect

Attempts to connect to an available Access Point and obtain an automatic IP address using `dhclient`. Previous settings are stored in files in `/var/run/wireless` so that they can be restored when the connection is terminated. The script:

- Obtains a list of all available Access Points and presents this list as a menu to the user. The user then chooses which Access Point to associate with.
- Associate with the selected Access Point.
- Backup the current `/etc/resolv.conf` file and current default route.
- Delete the current default route.
- Run `dhclient` on `ndis0` to obtain an IP address and update the routing table.

This script assumes the Access Point is NOT WEP protected AND is hosting a DHCP service. Further extensions to this script may check for WEP and ask the user for a key to use.

wireless.sh hangup

Disconnects from the current wireless connection. The script:

- Kills the currently running `dhclient` application.
- Deletes the current default route and restores the original default route.
- Restores the original `/etc/resolv.conf` file.

wireless.sh stop

Executed during system shutdown. At this stage we don't need to restore all settings, since most are dynamic and will be recreated during the next system boot phase. It is necessary to restore the `/etc/resolv.conf` file so that it is correct for the next time the system is started.

VPN Networking

Swinburne 802.11 wireless network access is controlled through a VPN gateway. Once a connection is established with the wireless Access Point, it is necessary to connect to the VPN gateway before open access to the network is granted. To do this we will be installing:

VPNC version 0.2.9

A Command Line tool to form a connection with a Cisco VPN Concentrator.

KVPNC version 0.4.1.2

A KDE Application providing a GUI frontend to VPNC.

Installing VPNC v0.2.9

The [kvpnc](#) application lists version 0.2.9 of `vpnc` as a requirement. While it may function with older versions of `vpnc`, this is not certain. Further, [kvpnc](#) specifically states that it will not function with any version of `vpnc` newer than version 0.2.9, specifically versions 0.3.x

So, how to install version 0.2.9 of `vpnc`:

An existing FreeBSD port/package entitled `vpnc` is located in `/usr/ports/security/vpnc`. The current version can easily be installed via:

```
pkg_add -r vpnc
```

The current packages for `vpnc` will install version 0.2.8 of the application, which is not the latest version. However, the version installed by this command can change at any time. The packages for versions 0.2.8 and 0.3.2 are available on the [FreeBSD website](#), but not for version 0.2.9. To ensure that the package for version 0.2.8 is not overwritten, we make this package available [here](ftp://ftp.freebsd.org/pub/FreeBSD/ports/i386/packages-5.3-release/All/vpnc-0.2_8.tbz). After downloading the package, it can be installed by executing:

```
pkg_add vpnc-0.2_8.tbz
```

This is necessary as the **Makefile** for the VPNC source tarball does not have a **make install** option. Adding the FreeBSD package will ensure that subsidiary files are installed in the correct location.

The source code package for VPNC can be obtained [here](#). The filename we want to download is **vpnc-0.2-rm+zomb.1.tar.gz**. Once extracted, we need to execute **make** in the directory to make the application, and then copy the generated `vpnc` binary to the `/usr/local/sbin` sub-directory.

Finally, the VPNC source code provides two script files **vpnc-connect** and **vpnc-disconnect** that are not installed as part of the FreeBSD package. These scripts are Linux specific and do not function as-is under FreeBSD. I have re-written them to work, they are available here:

- [vpnc-connect](#)
- [vpnc-disconnect](#)

vpnc-connect

This script wraps a call to `vpnc` such that it connects to the VPN Gateway, and updates the routing tables so that packets are forwarded to the newly created VPN tunnel.

vpnc-disconnect

This script kills the currently running `vpnc` daemon and restored the original routing tables so that communications will function as they did before the VPN tunnel was created.

Installing KVPNC v0.4.1.2

KVPNC is available at <http://home.gna.org/kvpnc>, the source code can be downloaded [here](#).

Once extracted you need to execute the following commands to install on FreeBSD:

```
./configure --prefix=/usr/local
make
make install (as root)
```

The "--prefix" option for **configure** is important, otherwise support files will be installed in the incorrect directory and KVPNC will not function correctly. Once installed, KVPNC will be available as an option on the Start->Internet menu on your KDE Desktop. When executed, it will ask for the root password (as **vpnc** needs to run as root in order to request port 500 and to update the routing tables). Be sure to set the configuration with the correct location of **vpnc**, **vpnc-connect** and **vpnc-disconnect**.

Swinburne Wireless Configuration

All connections to the VPN assume that we are already connected to the Swinburne Wireless Network (swinnet), that we have been allocated an IP address, and that the routing tables have been correctly programmed. Whether attempting to connect via the command line or via **kvpnc**, you must be root.

Connecting via the command line

As **root** execute:

```
vpnc-connect --gateway VPNGATEWAY --id GROUPID --username USERNAME
```

Where:

- **VPNGATEWAY** - IP Address of the Cisco VPN Concentrator
- **GROUPID** - The Group ID of the Cisco VPN Concentrator
- **USERNAME** - Your username on the Cisco VPN Concentrator

The script will ask you for the Group Password and your personal User password.

Connecting via **kvpnc**

Enter the details into the correct fields, ensure that the locations of the **vpnc-*** scripts are correct, and then click connect.

Values for Swinburne University

The IP Address of the Cisco VPN Concentrator used for wireless access at Swinburne University is:

- **VPNGATEWAY** - 136.186.13.6

Details of the usernames and passwords to use can be found [here](#). This web page is password protected and only available to Swinburne staff and students. Note that usernames and passwords are case sensitive.

Conclusion

In conclusion, while this document specifically outlines the steps required to get 802.11 wireless networking functioning on my Dell Latitude D505 running FreeBSD 5.3, many aspects of the procedure can be applied to other laptops.

- Project Evil allows the compilation of kernel modules to support any NDIS type network driver for Windows XP, the procedure of loading the kernel module is dependent on whether the Windows driver for your laptop requires more than two files.
 - The **wireless.sh** script can be modified to provide support for WEP protected networks and for static IP address situations.
 - VPNC and KVPNC are only necessary when operating in a VPN protected wireless environment.
-