oWL Linux SDK User Guide

H&D Wireless

Revision History

Revision	Revision date	Description
PA1	2010-05-14	First issue
PA2	2010-05-25	Updated after review

1 INTRODUCTION

The Wi-Fi device based on HDG104 from HD Wireless is available on platforms ranging from small 8-bit based systems to 32-bit MCU's and ARM-based cores. The Wi-Fi device comes in different hardware configurations to suit different kinds of applications and platforms. The device is available with SPI, SDIO and UART interfaces. The Wi-Fi device can be used with a real time kernel, on a free running system (no OS) or on a Linux-based system. The same Wi-Fi core software library will be used on all configurations.

This document will briefly describe the *Linux* device driver for the HD Wireless Wi-Fi device. Conceptually, this driver could be used on any host platform that supports Linux, however the testing has been focused on the AT91SAM9M10 development kit using an ARM926 core on the Atmel SAM9M10 application processor.

In this document, the name *owl* will be used when referring to the Linux driver for the HD Wireless Wi-Fi device. Any components prefixed with owl can be considered part of the driver.

The architecture of the owl Linux device driver, information on how to build it for a particular platform and guidelines on how to control it through common Linux tools will be outlined in the following chapters. Finally, a short guide on how to do performance measurements will be given.

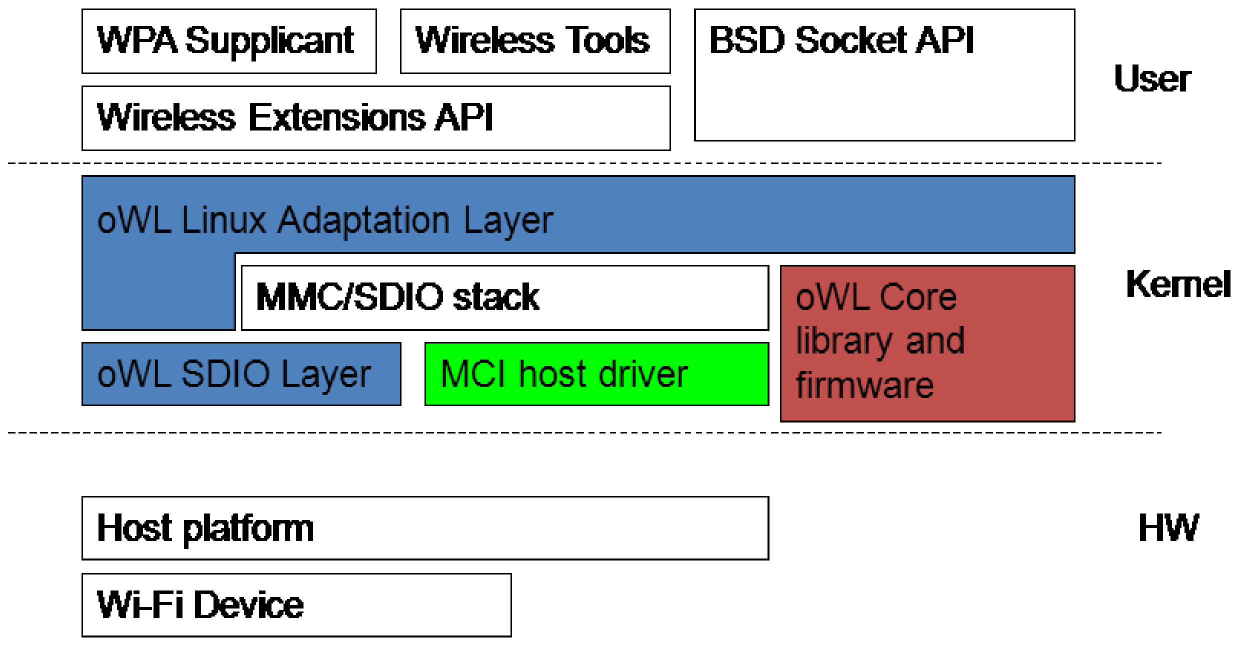
The owl device driver implements and reuses tools and interfaces that are de-facto standard in Linux systems. Therefore, most information presented here can also be found in manual pages and from various web resources. However, the purpose of this document is to act as a quick start guide for evaluating the HD Wireless Wi-Fi device and to highlight any extensions or limitations specific to the owl driver.

2 DEVICE DRIVER ARCHITECTURE

The following diagram shows the owl device driver architecture together with the interfaces and tools that can be used to control a wireless connection.

Once a link is established, either by using WPA Supplicant, Wireless Tools or Wireless Extension API directly (see figure), data can be exchanged using the BSD socket API, in the same way as a wired interface. This enables the usage of standard applications for e.g. web servers, service discovery and video/voice streaming.

From a user/application point of view, WPA Supplicant, Wireless Tools and Wireless Extension API acts as different level of interfaces towards the functionality provided by the device.



The components included in the device driver architecture will be briefly described below in a bottom-up fashion.

Wi-Fi Device

The Wi-Fi device based on HDG104 provided by HD Wireless. It fits into an SD card slot on the platform and supports SDIO, SPI and UART interfaces. The owl firmware, included in the core library, will execute on this device.

Host platform

The hardware that hosts the HD Wireless Wi-Fi device. The owl driver will execute on this platform.

SDIO stack

The open source SDIO stack included in Linux kernel versions starting from 2.6.23. The SDIO stack handles *host* devices and *card* devices. The owl device driver is designed as a card device that should fit right into this stack.

MCI host driver

The device driver for the SDIO/SPI host controller available on the host platform. The MCI host driver is used by the SDIO stack as a *host* device to perform the low level operations. This driver is usually provided by the host platform vendor.

oWL SDIO layer

SDIO driver for the Wi-Fi device. The SDIO driver will be used as a *card* device by the SDIO stack.

oWL core library

An OS-agnostic library compiled for the host platform. The library comes with a simple API that is used to control the Wi-Fi device. The oWL core library API is used by the oWL linux adaptation layer. The library will also manage the firmware that executes on the Wi-Fi device; the owl driver will download the firmware to the device.

oWL Linux adaptation layer

The oWL core library is wrapped by the oWL Linux adaptation layer to fit into the Linux network stack and Wi-Fi management interfaces. This enables the usage of common Linux tools and utilities such as Wireless Tools and WPA Supplicant.

Wireless Extensions API

The Wireless Extension API is the de-facto standard interface for 802.11 device control in Linux. This interface is based on ioctl system calls that offer a more fine-grained control of the device than what is offered by the higher levels. For detailed information see [1].

Wireless Tools

The Wireless Tools provide a set of console commands that can be used from the host platform in order to access a Wi-Fi device through the Wireless Extension API from the command line. Most of the configuration of the wireless device can be done with the provided *iwconfig* and *iwlist* commands. Usage of Wireless Tools will be explained in section 4.

WPA Supplicant

The WPA supplicant is an open source Linux application that is used in Wi-Fi client stations to implement key negotiation with a WPA Authenticator, and it may control the roaming and IEEE 802.11 authentication/association of the Wi-Fi driver. Usage of WPA Supplicant will be explained in section 5.

BSD Socket API

Standard interface to perform network communication from a user application.

3 BUILD, INSTALL AND LOAD THE DRIVER

Note that the oWL SDIO layer requires linux kernel version 2.6.23 or higher. If such kernel version is not available on the host platform, the oWL SDIO layer must be modified.

Also note that the owl driver is tested on linux kernel 2.6.30, Wireless Extensions 22, and Wireless Tools 29.

To compile the owl driver for a particular host platform, a cross compiler and the include files and configuration for the Linux kernel must be available. Make sure to replace `/path/to/kernel` with the actual path to the Linux kernel source tree used on the host platform. Also make sure to replace `/path/to/prefix` with the proper cross compiler prefix, e.g. `arm-linux-` if the name of the GCC binary is `arm-linux-gcc`. If building for a non-arm architecture, make sure to set the ARCH parameter accordingly (however, note that the wl_api core library included in the owl driver package is compiled for specific architecture).

```
$ tar xvzf owl-wifi-1.0.tar.gz
$ cd owl-wifi-1.0
$ make KERNELDIR=/path/to/kernel CROSS_COMPILE=/path/to/prefix ARCH=arm
```

This should produce the `owl.ko` binary.

Copy the `owl.ko` file to the host platform. The details of this step depend on the tools and interfaces available on the host platform. In most cases, `scp`, can be used to transfer files over an ssh connection.

```
$ scp owl.ko user@host:/path/to
```

Load the owl device driver, e.g. by issuing `insmod owl.ko` on the host platform. The owl device driver will output *net owl0: ready* in the kernel log (usually `/var/log/messages`). The owl device driver should now appear as interface *owl0* and should be listed by `ifconfig owl0`.

```
$ insmod owl.ko
$ ifconfig owl0
owl0      Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Note that the device is not usable and will not have a valid MAC address unless it is brought *up*. See section 4 for more information.

4 WIRELESS TOOLS

Wireless Tools can be used for basic interactive Wi-Fi management, directly from the command line. This section will show examples of how to perform the following operations using Wireless Tools.

- Enable the owl network interface
- Connect to an unprotected network
- Assign an IP address using static configuration or DHCP.
- Verify the connection status with the ping command.
- Disconnect from the network.
- Configure the owl device to use WEP encryption.
- Connect to a WEP encrypted network.
- Disable WEP encryption and delete any configured keys
- Enable device power save mode.

All the steps and expected output is provided in detail below. See the end of this section for a complete list of supported commands. For complete documentation on Wireless Tools, see the Wireless Tools documentation [3].

Enable the owl network interface

This will enable the owl device and firmware will be downloaded.

```
$ ifconfig owl0 up
```

Use `ifconfig` to see the interface information.

```
$ ifconfig owl0
owl0      Link encap:Ethernet  HWaddr 7A:C4:0E:A1:DD:9C
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Scan for networks

Scan for neighbouring networks. A list of networks with information on e.g. ssid, encryption mode and signal levels will be shown when the scan is completed.

```
$ iwlist owl0 scan
owl0      Scan completed :
Cell 01 - Address: 68:7F:74:10:5B:4C
          ESSID:"hdwireless"
          Mode:Managed
          Encryption key:off
          Frequency:2.422 GHz (Channel 3)
          Quality:21/30  Signal level:-29 dBm  Noise level:-50 dBm
          Extra:Beacon period: 100 Kusec
          Extra:DTIM period: 1

Cell 02 - Address: 00:23:69:B5:BE:48
          ESSID:"angr2"
          Mode:Managed
          Encryption key:off
          Frequency:2.412 GHz (Channel 1)
          Quality:20/30  Signal level:-30 dBm  Noise level:-50 dBm
          Extra:Beacon period: 100 Kusec
          Extra:DTIM period: 2

Cell 03 - Address: 00:25:9C:6F:58:B0
          ESSID:"TeliaADSL"
          Mode:Managed
          Encryption key:on
          Frequency:2.437 GHz (Channel 6)
          Quality:22/30  Signal level:-45 dBm  Noise level:-67 dBm
          IE: IEEE 802.11i/WPA2 Version 1
              Group Cipher : CCMP
              Pairwise Ciphers (1) : CCMP
              Authentication Suites (1) : PSK
          Extra:Beacon period: 100 Kusec
          Extra:DTIM period: 1

...
```

Connect to an unprotected network

Connect to an unprotected network that was found during the scan. In this case we will choose the ssid *hdwireless*.

```
$ iwconfig owl0 essid hdwireless
```

This will initiate a connect, use the `iwconfig` command again to "poll" the connection status. The `iwconfig` output will look similar to below when not yet connected:

```
$ iwconfig owl0
owl0      IEEE 802.11bg  ESSID:"hdwireless"
          Mode:Managed  Access Point: Not-Associated
          Encryption key:off
          Power Management:off
```

Once the connetion is established, the MAC address of the access point should be displayed in the output:

```
$ iwconfig owl0
owl0      IEEE 802.11bg  ESSID:"hdwireless"
          Mode:Managed Frequency:2.422 GHz Access Point:68:7F:74:10:5B:4C
          Encryption key:off
          Power Management:off
          Link Quality=19/30  Signal level=-27 dBm  Noise 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
```

Note again that the `iwconfig` command will only *initiate* a connection. Another connection request cannot be issued until the previous connection has been successfully established or considered failed. Therefore, if the owl driver receives two connection requests very quickly, the last one will be discarded:

```
$ iwconfig owl0 essid hdwireless; iwconfig owl0 essid foo
Error for wireless request "Set ESSID" (8B1A) :
    SET failed on device owl0 ; Device or resource busy.
```

A wireless *event* that indicates connected or disconnected status will be generated when a connection attempt is completed. However, the details about wireless events are out of the scope for this document. See the Linux kernel header file and related files [1] for more information.

Assign an IP address

If a static IP address is used, set it using ifconfig:

```
$ ifconfig owl0 192.168.2.50
```

If DHCP is used, invoke the DHCP client available on the platform:

```
$ udhcpc -i owl0
udhcpc (v1.13.2) started
Sending discover...
Sending select for 192.168.2.102...
Lease of 192.168.2.102 obtained, lease time 172800
adding dns 192.168.2.1
```

The current IP address can now be displayed using `ifconfig`:

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 3E:36:65:BA:6F:BE
          inet addr:192.168.2.102  Bcast:192.168.2.255
          Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6601 errors:173 dropped:0 overruns:0 frame:0
          TX packets:173 errors:0 dropped:0 overruns:0 carrier:0
          collisions:49 txqueuelen:1000
          RX bytes:938610 (916.6 KiB)  TX bytes:19540 (19.0 KiB)
          Interrupt:25 Base address:0xc000
```

Verify the connection

Ping the access point to verify the connection:

```
$ ping -c 3 192.168.2.1
PING 192.168.2.1 (192.168.2.1): 56 data bytes
64 bytes from 192.168.2.1: seq=0 ttl=64 time=1.672 ms
64 bytes from 192.168.2.1: seq=1 ttl=64 time=1.333 ms
64 bytes from 192.168.2.1: seq=2 ttl=64 time=1.342 ms

--- 192.168.2.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.333/1.449/1.672 ms
```

Disconnect from the network

Disconnect from the network by setting the ssid to `off`:

```
$ iwconfig owl0 essid off

$ iwconfig owl0
owl0      IEEE 802.11bg  ESSID:""
          Mode:Managed  Access Point: Not-Associated
          Encryption key:off
          Power Management:off
```

Configure the device to use WEP encryption

Now, we will connect to an access point that uses WEP 64-bit encryption, shared key authentication and with key index 1 set to *0102030405*. Key index 1 should be the default transmit key and no other keys should be configured on the access point.

First, the key must be configured into the owl device:

```
$ iwconfig owl0 key 0102030405 [1] restricted
```

The *[1]* specifies that we are setting key index 1 (as was configured in the access point), *restricted* means that shared key authentication will be used. To list the current key configuration, `iwlist` can be used:

```
$ iwlist owl0 keys
owl0      4 keys available :
                [1]: 0102-0304-05 (40 bits)
                [2]: off
                [3]: off
                [4]: off
          Current Transmit Key: [1]
          Security mode:restricted
```

Connect to an WEP encrypted network

Now it should be possible to connect to the network, set an ip address and verify the connection with ping:

```
$ iwconfig owl0 essid hdwireless

$ iwconfig
owl0      IEEE 802.11bg  ESSID:"hdwireless"
          Mode:Managed Frequency:2.422 GHz Access Point:68:7F:74:10:5B:4C
          Encryption key:0102-0304-05   Security mode:restricted
          Power Management:off
          Link Quality=23/30  Signal level=-20 dBm  Noise level=-43 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

$ ifconfig owl0 192.168.2.50
$ ping -c 3 192.168.2.1
PING 192.168.2.1 (192.168.2.1): 56 data bytes
64 bytes from 192.168.2.1: seq=0 ttl=64 time=1.803 ms
64 bytes from 192.168.2.1: seq=1 ttl=64 time=1.707 ms
64 bytes from 192.168.2.1: seq=2 ttl=64 time=1.503 ms

--- 192.168.2.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.503/1.671/1.803 ms
```

Disable WEP encryption and delete keys

To disable WEP encryption and delete all the configured keys, the `iwconfig` command can be used:

```
$ iwconfig owl0 key off
```

Enable device power save mode

In power save mode, the device will sleep until either the host request to transmit data or until there is buffered incoming data to fetch from the access point. In power save mode, the throughput performance will be slightly degraded and the response latency will depend on the access point *beacon interval* and *DTIM* parameters. In most cases the access point is configured with 100 ms beacon interval and a *DTIM* interval of 1; this results in responses time around 100 ms. Depending on the application, the power consumption of the owl device can be heavily reduced.

Device power save mode can be enabled with the `iwconfig` command:

```
$ iwconfig owl0 power on
```

The current power management configuration can be shown by issuing `iwconfig` again:

```
$ iwconfig owl0

owl0      IEEE 802.11bg  ESSID:"angr"
          Mode:Managed Frequency:2.422 GHz Access Point:68:7F:74:10:5B:4C
          Bit Rate=54 Mb/s
          Encryption key:0102-0304-05   Security mode:restricted
          Power Management timeout:10
          Link Quality=23/30  Signal level=-20 dBm  Noise level=-43 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Power Management timeout:10 indicates that the device is in power save mode with a timeout setting of 10 [ms]. After waking up, the device will wait for timeout [ms] before going back to sleep.

List of supported Wireless Tools commands and options

```
iwconfig owl0 key <key 5 or 13 hex digits> [idx] [authmode] [off]
iwconfig owl0 essid <ssid|off>
iwconfig owl0 ap <mac>
iwconfig owl0 power on|off
iwlist owl0 keys
iwlist owl0 scan
```

5 WPA SUPPLICANT

The WPA supplicant can be configured to control the roaming and IEEE 802.11 authentication/association of the owl device. The configuration is usually performed in a configuration file, e.g. `/etc/wpa_supplicant.conf`.

It is also possible to directly issue commands to the WPA Supplicant, using a dedicated shell command, `wpa_cli`. The usage of `wpa_cli` is out of the scope of this document, but is described in detail in the WPA supplicant documentation [4].

This section will show examples of how to perform the following operations using WPA Supplicant.

- Connect to an unprotected network
- Connect to a WPA protected network that uses TKIP encryption
- Connect to a WPA2 protected network that uses CCMP encryption
- Connect to a network that uses any WPA/WPA2 protocol and TKIP/CCMP encryption.

All the steps and expected output is provided in detail below. See the end of this section for a complete list of supported configurations. For complete documentation on WPA Supplicant, see the WPA Supplicant documentation [4].

Connect to an unencrypted network

To simply instruct the WPA Supplicant to connect to any unencrypted network with ssid *hdwireless*, the following configuration file should be enough:

```
ctrl_interface=/var/run/wpa_supplicant
network={
        ssid="hdwireless"
        key_mgmt=NONE
}
```

The path to the configuration file and the interface name (owl0) should then be passed as parameters when starting the WPA Supplicant:

```
$ wpa_supplicant -Dwext -iowl0 -c /etc/wpa_supplicant.conf -B
```

The paramater `-Dwext` informs the WPA Supplicant that the standard Wireless Extensions interface should be used to control the network interface. For detailed information on how to configure and run the WPA supplicant, see the WPA supplicant documentation [4].

The WPA Supplicant will now periodically scan for networks until one that matches the configuration is found. Once found, a connection will be established. The WPA Supplicant will also handle reconnect if the connection is lost. Therefore, opposed to Wirieless Tools, when using the WPA Supplicant, it is not necessary to perform manual scanning and network selection.

Note that the WPA Supplicant configuration can hold several networks and the WPA Supplicant will choose and roam amongst them. However, most importantly, the WPA supplicant implements the key negotiation with a WPA Authenticators.

Connect to a WPA protected network that uses TKIP encryption

To connect to a network using WPA key management and TKIP encryption, the following network configuration can be specified in the configuration file:

```
network={
        ssid="hdwireless"
        key_mgmt=WPA-PSK
        group=TKIP
        pairwise=TKIP
        proto=WPA
        psk="hdwirelesskey"
}
```

The key configured on the access point should be *hdwirelesskey*

To force the WPA Supplicant to re-read its configuration file wpa_cli can be used

```
$ wpa_cli reconfigure
```

One should remember that all wireless operations performed by both the WPA supplicant and Wireless Tools are done through the same Wireless Extensions API. This means that it will still be possible to e.g. check the connection status with `iwconfig`:

```
$ iwconfig
owl0      IEEE 802.11bg  ESSID:"angr"
          Mode:Managed Frequency:2.422 GHz Access Point:68:7F:74:10:5B:4C
          Bit Rate=54 Mb/s
          Encryption key:472A-7E38-C465-D4EB-6DA7-BAE6-4700-0960-EDB1-
          40DE-18CC-5A02-4AE1-EA96-F3EE-142A   Security mode:open
          Power Management timeout:10
          Link Quality=24/30  Signal level=-20 dBm  Noise level=-44 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Once connected, it is possible to obtain an ip address and perform the ping test:

```
$ udhcpc -i owl0
Sending discover...
Sending select for 192.168.2.102...
Lease of 192.168.2.102 obtained, lease time 172800
adding dns 192.168.2.1

$ ping -c 3 192.168.2.1
...
```

Connect to a WPA2 enabled network that uses CCMP encryption

To connect to a network using the WPA2 protocol and CCMP encryption, the following network configuration can be specified in the configuration file:

```
network={
        ssid="hdwireless"
        key_mgmt=WPA-PSK
        group=CCMP
        pairwise=CCMP
        proto=WPA2
        psk="hdwirelesskey"
}
```

Connect to a network that uses any WPA/WPA2 protocol and TKIP/CCMP encryption

Note that several encryption parameters can be specified on a single line, allowing connections to a specific ssid using a range of encryption methods. The configuration file below should allow connections to the *hdwireless* access point regardless of whether the WPA or WPA2 protocol is used or whether CCMP or TKIP is used for pairwise and group key encryption. The actual encryption method used will be the most secure one that is supported by the access point.

```
network={
        ssid="hdwireless"
        key_mgmt=WPA-PSK
        group=TKIP CCMP
        pairwise=TKIP CCMP
        proto=WPA WPA2
        psk="hdwirelesskey"
}
```

List of supported WPA Supplicant network options

Key management (`key_mgmt`):	`WPA-PSK, NONE`
Group key encryption (`group`):	`CCMP, TKIP`
Pairwise key encryption (`pairwise`):	`CCMP, TKIP`
Protocol (`proto`):	`WPA, WPA2`

6 IFUP AND IFDOWN TOOLS

In Linux, network device configuration is usually specified in `/etc/network/interfaces`. This allows the usage of the tools `ifup` and `ifdown`. Wired as well as wireless interfaces can be configured in the same file.

To configure the owl0 interface to use the WPA Supplicant, a particular `wpa_supplicant.conf` file and specific IP settings the following entry can be added to `/etc/network/interfaces`:

```
iface owl0 inet dhcp
        wpa-driver wext
        wpa-conf /etc/wpa_supplicant.conf
```

Now, when the owl0 interface is brought up using `ifup owl0`, the WPA Supplicant will be started with the specified configuration file and as soon as a connection is established, an IP address will be assigned using DHCP.

Note that Wireless Tools can also be invoked through `ifup` and `ifdown`, if declared in `/etc/network/interfaces`. See the manual pages for `ifup`, `ifdown` and `interfaces` for more information.

7 PERFORMANCE EVALUATION

This section will briefly describe how to evaluate the throughput performance of the owl device driver and the HD Wireless Wi-Fi device.

The throughput performance evaluation is performed by sending TCP data between the Wi-Fi device (DUT) and a PC. The DUT and PC should be connected to the same access point. To avoid any limitations introduced by the access point and the PC, the PC should have a wired connection to the access point. For maximum performance, make sure that the selected channel is not used by any other equipment and that no other client is connected to the AP.

Using the TCP protocol to test throughput performance certainly introduces quite some protocol related overhead such as extra headers, retransmissions and acknowledgements. Therefore the actual throughput results will be highly dependent on the platform dynamic responsiveness (e.g. to handle incoming TCP acknowledgements) since TCP packets will be sent in both directions to successfully complete a single direction data transfer. The actual raw data throughput can be considered significantly better; e.g. using the UDP protocol would give a higher throughput of actual payload data.

The throughput measurement setup and execution will be performed in the following steps:

- Obtain and configure the software tools
- Establish a link between the DUT and the PC
- Test TX throughput
- Test RX throughput

All the steps and expected output is provided in detail below.

Obtain and configure the software tools

In this case, to transmit and receive TCP data, the standard tool TTCP [5] has been chosen. It can be obtained from e.g. http://www.netcore.fi/pekkas/linux/ipv6/ttcp.c.

To compile ttcp for the host platform to which the Wi-Fi device is connected (DUT), use the cross compiler for the platform. The cross compiler used here is the same as the one used to compile the owl device driver. Make sure to replace `/path/to/gcc` with the proper cross compiler, e.g. `arm-linux-gcc` if that is the name of the GCC binary.

```
$ /path/to/gcc ttcp.c -o ttcp
```

Make sure to copy the `ttcp` binary to the host platform. `ttcp` is also needed on the PC. On a Linux PC, the same TTCP source code can used:

```
$ gcc ttcp.c -o ttcp
```

If a windows PC is used, `pcattcp` should be used and can be downloaded from http://www.pcausa.com/Utilities/pcattcp.htm.

During test execution, the TTCP tool will be invoked on both the DUT and PC, one acting as the transmitter and the other one as the receiver. The transmitter will send a fixed (but configurable) amount of data to the receiver using the TCP protocol. The throughput will then be calculated based on the amount of *payload* data transferred in during the elapsed time.

Establish a link between the DUT and the PC

Make sure that the PC gets an IP address from the access point (or use a static IP) when the network cable is connected. In Windows, this should happen automatically as long as DHCP is enabled in the network configuration for the ethernet port. In Linux the DHCP client might have to be started manually, depending on the distribution and configuration. Now connect the DUT to the same access point, as explained in section 4 and 5.

It should now be possible to ping the PC from the DUT through the access point by using the `ping` command in the DUT console (make sure that the actual IP address of the PC is replaced in the example below)

```
$ ping <ip>
...
```

Test TX throughput

First start `ttcp` on the PC in receive mode. In windows use the `pcattcp` tool:

```
PC> pcattcp -r -s
```

In Linux, use the `ttcp` tool:

```
PC> ttcp -r -s
```

Then the transmitter should be started on the DUT by using the `ttcp` command on the DUT:

```
$ ttcp -t –s –n128000 <ip address of PC>
```

This will start the transfer of 128 Mb payload data. When the transfer is completed, throughput information will be printed on both the DUT console and the PC console. Se below for example output shown on DUT followed by output on PC.

```
$ ttcp -t –s –n128000 <ip address of PC>
ttcp-t: nbuf=128000, buflen=1024, port=2000
ttcp-t: socket
ttcp-t: connect
ttcp-t: 0.1user 3.8sys 0:52real 7% 0i+0d 0maxrss 0+2pf 6185+37534csw
ttcp-t: 131072000 bytes processed
ttcp-t: 3.94 CPU sec = 32487.3 KB/cpu sec, 259898 Kbits/cpu sec
ttcp-t: 52.0811 real sec = 2457.71 KB/real sec, 19661.6Kbits/sec

PC> ttcp -r -s
ttcp-r: nbuf=1024, buflen=1024, port=2000
ttcp-r: socket
ttcp-r: accept
ttcp-r: 0.0user 0.5sys 0:52real 1% 0i+0d 392maxrss 0+0pf 91562+27csw
ttcp-r: 131072000 bytes processed
ttcp-r: 0.564035 CPU sec = 226936 KB/cpu sec, 1.81549e+06 Kbits/cpu sec
ttcp-r: 52.0956 real sec = 2457.02 KB/real sec,   19656.2 Kbits/sec
```

Test RX throughput

First start `ttcp` on the DUT in receive:

```
$ ttcp -r -s
```

Then the transmitter should be started on the PC. In windows, use the `pcattcp` tool:

```
PC> pcattcp -t -s -n128000 <ip address of DUT>
```

In linux, use the `ttcp` tool:

```
PC> ttcp -t -s -n128000 <ip address of DUT>
```

This will start the transfer of 128 Mb payload data. When the transfer is completed, throughput information will be printed on both the DUT console and the PC console. . Se below for example output shown on DUT followed by output on PC.

```
$ ttcp -r -s
ttcp-r: nbuf=1024, buflen=1024, port=2000
ttcp-r: socket
ttcp-r: accept
ttcp-r: 0.4user 13.1sys 0:58real 23% 0i+0d 0maxrss 0+1pf 85295+399984csw
ttcp-r: 131072000 bytes processed
ttcp-r: 13.6 CPU sec  =   9411.76 KB/cpu sec,    75294.1 Kbits/cpu sec
ttcp-r: 58.4456 real sec =   2190.07 KB/real sec,   17520.6 Kbits/sec

PC> ttcp -t -s -n128000 <ip address of DUT>
ttcp-t: nbuf=128000, buflen=1024, port=2000
ttcp-t: socket
ttcp-t: connect
ttcp-t: 0.0user 0.1sys 0:58real 0% 0i+0d 392maxrss 0+1pf 7118+26csw
ttcp-t: 131072000 bytes processed
ttcp-t: 0.156009 CPU sec  = 820465 KB/cpu sec,  6.56372e+06 Kbits/cpu sec
ttcp-t: 58.4284 real sec = 2190.71 KB/real sec,   17525.7 Kbits/sec
```

8 REFERENCES

[1] Linux kernel include-file: `include/linux/wireless.h`
[2] wl_api reference documentation
[3] http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html
[4] http://hostap.epitest.fi/wpa_supplicant/