Prototyping Collision-Free MAC Protocols in Real Hardware

Luis Sanabria-Russo
NeTS Research Group at
Universitat Pompeu Fabra, Barcelona, Spain
Luis.Sanabria@upf.edu

***Abstract*—Carrier Sense Multiple Access with Enhanced Collision Avoidance (CSMA/ECA) is a totally distributed, collision-free MAC protocol for IEEE 802.11 WLANs. It is capable of achieving greater throughput than the current standard, called Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), by means of picking a deterministic backoff after successful transmissions. This work provides a overview on the procedure which led to the first implementation of CSMA/ECA on real hardware using OpenFWWF.**

***Index Terms*—OpenFWWF, WMP, MAC, Collision-free, CSMA/ECA.**

A SHORT WARNING

Prior the introduction, it is appropriate to filter interests. This report assumes a bit of background on WiFi technology and terminology, nevertheless many of the references are detailed at the end of the document.

Procedures described here must be done at your own risk. Wireless cards (as mentioned in some of the references) might get permanently damaged. Nevertheless, all the events and workarounds that were necessary to achieve the final test of CSMA/ECA will be dutifully detailed.

I. INTRODUCTION

A device's firmware is the one managing memory and instructions to make it work as intended. As for most devices, including wireless cards, the firmware is custom made, unique for each architecture and running in the device's own microprocessor.

Even-though the IEEE 802.11 set of WLAN standards define the procedures to guarantee effective communication among hosts, the implementation part is the task of manufacturers. This means that *how* the standard is implemented vary from vendor to vendor and explains why firmware is closely related to the underlying hardware.

To protect manufacturer's commercial interests, firmwares are not usually allowed to be modified until the end of the product's life-cycle. Nevertheless, current efforts both from the industry and the open source community (as in the case of MadWiFi [1] driver and OpenFWWF [2] firmware), had led to interesting opportunities for the research community.

Carrier Sense Multiple Access with Enhanced Collision Avoidance (CSMA/ECA) [3] is a totally distributed MAC protocol for WLANs. It provides an increase in the achieved throughput for contending stations by making simple modifications to the current standard, Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). Although simple, CSMA/ECA requires modifications to very time-sensitive task (backoff procedure), which are handled at firmware level.

This documents details the process of acquiring and pushing a modified version of the open sourced OpenFWWF firmware, containing the code for CSMA/ECA, to an specific model of Wireless Network Interface Controller (WNIC or wireless cards). Most of the procedures described here can be found on the web, other recommendations are relayed from one of the authors of OpenFWWF.

II. WHAT WILL YOU NEED?

As mentioned in [2], it is recommended to use a Linux distribution with kernel version 2.6.27-rc5. Nevertheless, it has proven to be safe for us to use Ubuntu 8.10 [4].

The OpenFWWF is used in combination with the `b43` Linux wireless card driver, which in turn is supported by a limited set of Broadcom cards [5]. For this reason, you will need such card model to continue. We have successfully tested our implementation with a Broadcom BCM4318 card (see Figure 1a), which is available at [6] and connected to a PCI slot with a miniPCI to PCI adapter (see Figure 1b), like the one available at [7].

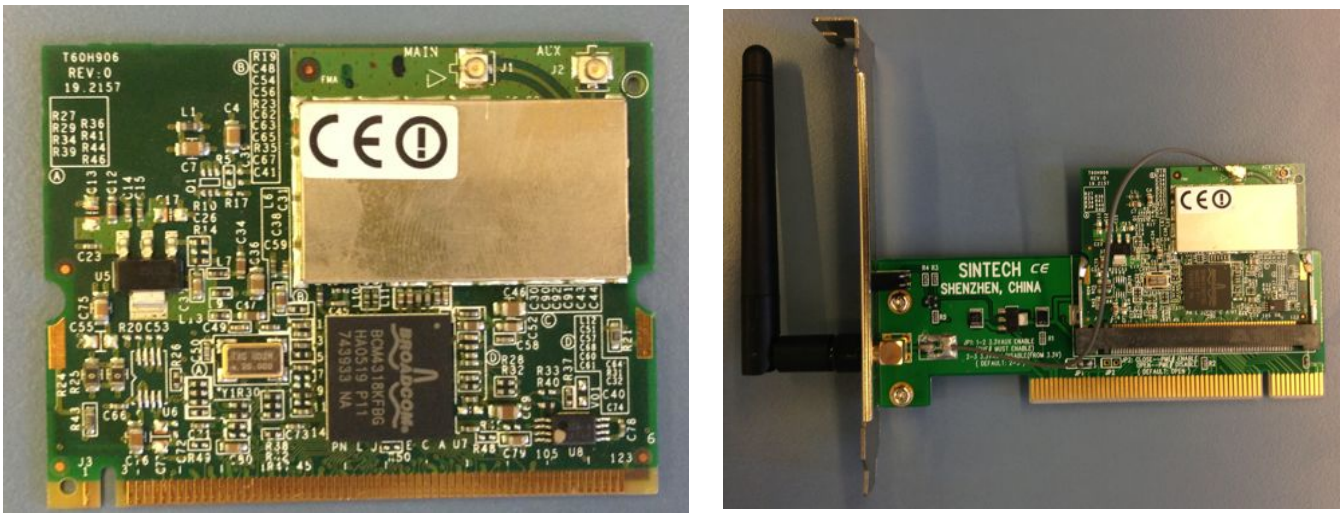
It is also required to have a wired Internet connection to continue further, given that after the Ubuntu installation you are required to fetch the firmware, compiler and prerequisites.

III. SETTING UP THE ENVIRONMENT

After the installation of the Ubuntu 8.10, restrain from installing updates. It is better to complete the procedure as detailed here and then go on your own.

The required Ubuntu 8.10 software sources cannot be reached with the default configuration, it is needed to update the `/etc/apt/sources.list` file to include a repository where you can download and install the prerequisites. This can be achieved in the following way:

1) Open a Terminal window and introduce the following command: `sudo gedit /etc/apt/sources.list`
   This command will cause the Gedit text editor to open the specified file with root credentials (you may be prompted for the root password).



(a) (b)

Fig. 1: 1a) Broadcom BCM418 miniPCI. 1b) Card correctly placed into the PCI adapter.

2) Go to the end of the file and add the following line: `deb http://ubuntu.mirror.cambrium.nl/ubuntu/ lucid main`.
3) Save and close the file.
4) Go back to the Terminal window and update the sources list by issuing: `sudo apt-get update`
   Depending on your Internet connection, this may take a while.

After the update completes, you are now able to download the prerequisites. It is required to install the `git-core` package to fetch the code from remote sources, `g++` to compile C++ code, `bison` which is a parser generator and `flex` (Fast Lexical Analyzer). Issue the following command in a terminal window: `sudo apt-get install git-core bison flex g++` (you may be prompted for the root password). This takes some time to complete.

After all the above, your computer is ready for the card and firmware installation.

A. Installing the required hardware

If you are using the card referred to in Section II, it is necessary to plug it into a miniPCI to PCI adapter (unless you have a miniPCI slot on your computer. If so, please jump to the following subsection).

This is a very delicate task, and you should avoid forcing the parts into place. Please refer to Figure 1 to see the finished result.

B. Setting OpenFWWF to work

There are various ways for completing this part of the procedure. Nevertheless, the one provided by [8] is both simple and it works.

Summarizing (links refer to the actual commands for layout limitations):

1) Download the latest version of the firmware source code by issuing the command contained in `http://pastebin.com/VwdKEhZ1` into a Terminal window.
   This will create a directory named openfwwf-5.2.tar.gz containing the OpenFWWF firmware. Unpack it issuing the following command: `tar -zxvf openfwwf-5.2.tar.gz`.
2) Download the assembly language compiler (b43-tools [9]) from its git repository issuing the command found in `http://pastebin.com/8RC6c01g`
3) Once `b43-tools` is downloaded, issue `cd b43-tools/assembler` to get into the assembler directory. Then build it typing the `make` command.
   This step will create two files, namely: `b43-asm` and `b43-asm.bin`.
4) Copy `b43-asm` and `b43-asm.bin` into the openfwwf-5.2 directory by typing `cp b43-tools/assembler/b43-asm* openfwwf-5.2/`
5) You need to modify the Makefile (openfwwf-5.2/Makefile) replacing `BCMASM = b43-asm` by `BCMASM = ./b43-asm`.
6) Now just build the firmware and install it by typing `make` and then `sudo make install`.
7) As recommended in [2], edit /etc/modprobe.d/arch/i386 file issuing the following command to open it `sudo gedit /etc/modprobe.d/arch/i386` and then add the following line to the end of the file: `options b43 qos=0 nohwcrypt=1`. Save and close the file.
8) Restart your computer.

You can browse and edit the firmware by modifying the openfwwf-5.2/ucode5.asm file. To test the OpenFWWF, build and install the firmware (as in Step 6 above) and restart your computer.

Now you should be able to see the wireless interface at the top right corner of Ubuntu's menu bar when you log back in

into your user account.

IV. Prototyping and testing CSMA/ECA

Carrier Sense Multiple Access with Enhanced Collision Avoidance (CSMA/ECA) was first introduced by Barcelo et. al. in [3]. The way CSMA/ECA works allows it to build a collision-free state among contenting stations, which has a favorable impact on throughput. Further modifications [10] allowed CSMA/ECA to increase the number of contenders able to maintain the collision-free state.

The way CSMA/ECA works is described bellow.

1) When a packet arrives at a previously empty MAC queue, a CSMA/ECA node generates a random backoff, $B \in [0, CW(k)]$. Where $CW(k) = 2^k CW_{min}$ is the Contention Window at backoff stage $k \in [0, m]$ and CW_{min} the minimum contention window.
2) Each passing empty slot decrements B in one. When the backoff expires ($B = 0$), the node will attempt transmission.
3) If an ACKowledgement (ACK) from the receiver is received: the backoff stage is reset ($k = 0$), and if there are other packets in the MAC queue, CSMA/ECA instructs the node to pick a deterministic backoff, $B_d = CW_{min}/2$.
4) If no ACK is received, a collision is assumed. The backoff stage is incremented in one (k+=1) and the backoff process is restarted (goes to Step 1).

CSMA/ECA just picks a deterministic backoff $B_d = CW_{min}/2$ after successful transmissions; while collisions force the node to increase its backoff stage (k) and pick a random backoff, B.

CSMA/ECA differs from CSMA/CA in the fact that a deterministic backoff is chosen after successful transmissions, as opposed to CSMA/CA where the backoff is always chosen at random.

A short example of what is proposed in [3] is shown in Figure 2.

In Figure 2, each station (STA) generates a random backoff at startup. The red outline indicates that at least two of the generated backoff values are the same and stations will consequently collide. On the other hand, successful stations will generate a deterministic backoff (7 in the case of example Figure 2) and effectively avoid collisions with other successful stations in future cycles, achieving a collision-free state.

The simplicity of CSMA/ECA makes it very easy to implement in OpenFWWF, as will be described in the following.

A. Implementing CSMA/ECA

To replicate CSMA/ECA behaviour, each station must pick the same deterministic backoff after successful transmissions. The backoff parameters are set in the `set_backoff_time` function of the *ucode.asm* file (near the end).

To make sure CSMA/ECA stations pick a deterministic backoff after successful transmission, we first check that the current contention window is the minimum contention window (see Item 3 in the CSMA/ECA description). Then, a deterministic backoff is assigned. The following shows the replaced code as well as the code lines that would convert CSMA/CA into CSMA/ECA in OpenFWWF:

- Random backoff generation: **and** SPR_TSF_Random, CUR_CONTENTION_WIN, GP_REG5; is replaced by: **je** CUR_CONTENTION_WIN, DEFAULT_MIN_CW, deterministic_backoff;.
  This replacement just checks if the current contention window is equal to the minimum contention window (**if**(CUR_CONTENTION_WIN == DEFAUL_MIN_CW)). In a saturated scenario, it can be assumed that the station either successfully transmitted a packet.
- If the above result is positive, the flow is redirected to a new deterministic_backoff section of the `set_backoff_time` function, which contains the following instruction: **or** 0x100, 0x000, GP_REG5;. Assigning 0x100 to the GP_REG5 register and successfully changing the backoff value.

B. Testing CSMA/ECA

A simple testing scenario was built in order to check whether the modifications performed matched the expected CSMA/ECA behaviour. This was composed of two Ubuntu 8.10 PCs with Broadcom BCM4318 cards running OpenFWWF firmware as WLAN STAtions (STA) connected to an Access Point (AP). To make performance tests, Iperf [11] tool generates UDP streams at 65 Mbps from both STAs to a Server wired to the access point using Ethernet. At the Server, Wireshark [12] captures all packets from the STAs. Figure 3 provides an overview of the testing scenario.

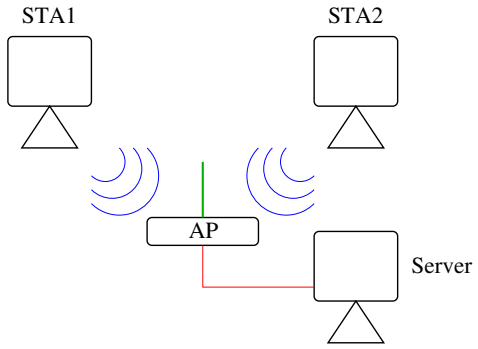


Fig. 3: Test setup

Two simple test were performed, the first tries to reveal evidence of the deterministic backoff counter, while the second aims at looking at the achieved throughput.

Figures 4 and 5 show a random set of a hundred server-received packets from STAs running CSMA/CA and CSMA/ECA, respectively.

C. Results

Backoff mechanism: As it is expected, CSMA/CA's randomized backoff mechanism is easy to appreciate in Figure 4, where the "Transmitters" line shows the transmitter of a

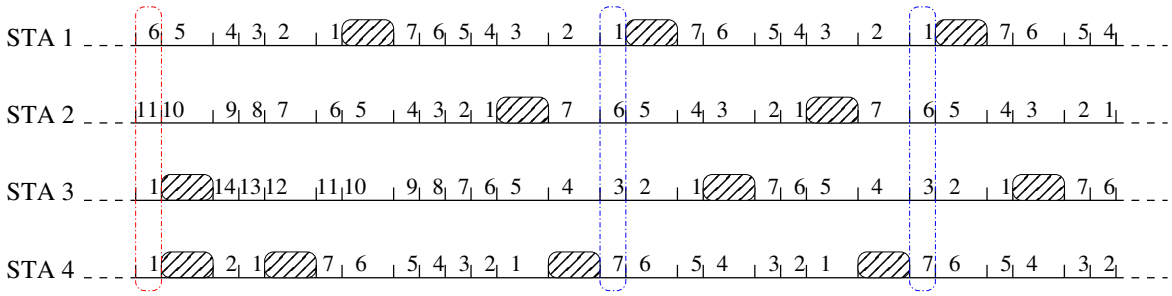


Fig. 2: CSMA/ECA example in saturation

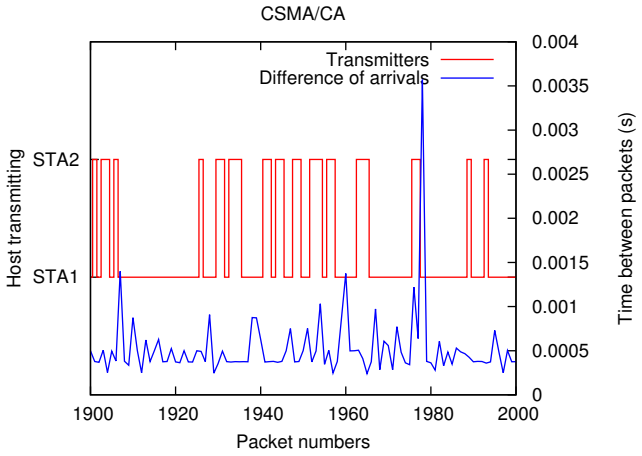


Fig. 4: CSMA/CA transmission turns between STA1 and STA2

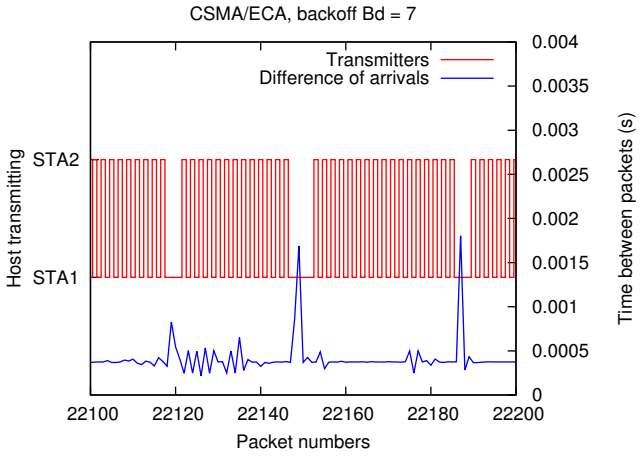


Fig. 5: CSMA/ECA transmission turns between STA1 and STA2

given packet. Whereas in CSMA/ECA (Figure 5), transmitters almost alternate transmissions.

Throughput: From the "Time between packets" curve, we can appreciate that with CSMA/CA the average time between arrived packets is greater than with CSMA/ECA. This means that CSMA/ECA stations are able to access the channel more often than those running CSMA/CA, resulting in an increased throughput. Figure 6 shows the average throughput achieved by each station while attempting to transmit generic UDP packets at 65 Mbps to the Server.

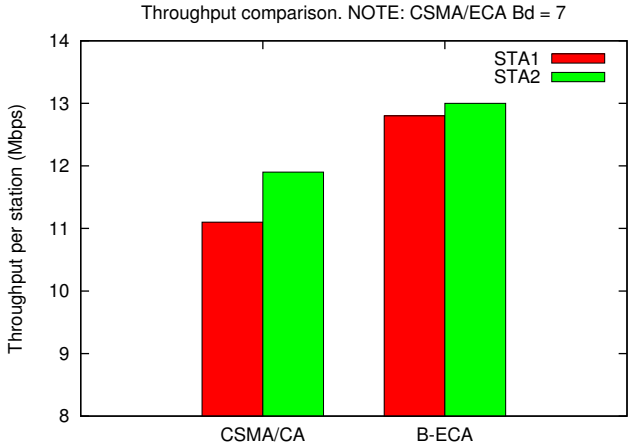


Fig. 6: Throughput

V. Conclusion

OpenFWWF is a open alternative for researchers wanting to test new MAC protocols in real hardware. In this document the process of obtaining, installing and testing CSMA/ECA is described and evaluated alongside with encouraging results.

Carrier Sense Multiple Access with Enhanced Collision Avoidance (CSMA/ECA) attempts to build a collision-free WLAN environment, which has significant benefits in terms of throughput. Results from these tests show evidence of an effective modification of CSMA/CA's backoff mechanism alongside with the throughput increase of CSMA/ECA.

This is the first real hardware implementation of CSMA/ECA. Further work might involve the design of more accurate testing scenarios, collision detection and the extension of CSMA/ECA to work with more contenders [10].

VI. Acknowledgement

The author would like to extend appreciation to Francesco Gringoli for his insight and counsel, as well to Germán Corrales Madueño for his ever-pushing and contagious motivation.

REFERENCES

[1] The MADWifi Project. (2013) Multiband Atheros Driver for Wireless Fidelity. Webpage. [Online]. Available: http://madwifi-project.org/

[2] F. Gringoli and L. Nava. (2010) Open Firmware for WiFi Networks. Webpage. [Online]. Available: http://www.ing.unibs.it/openfwwf/

[3] J. Barcelo, B. Bellalta, C. Cano, and M. Oliver, "Learning-BEB: Avoiding Collisions in WLAN," in *Eunice*, 2008.

[4] Canonical Ltd. (2008) Ubuntu 8.10 (Intrepid Ibex). Webpage. [Online]. Available: http://old-releases.ubuntu.com/releases/intrepid/

[5] Linux Wireless. b43 and b43legacy. Webpage. [Online]. Available: http://wireless.kernel.org/en/users/Drivers/b43

[6] Amazon UK. (2013) Broadcom BCM4318 miniPCI W-LAN. Webpage. [Online]. Available: http://amzn.to/16aaNcw

[7] ——. (2013) Sourcingmap Mini-PCI to Standard PCI Adapter WiFi Wireless LAN w/ Antenna. Webpage. [Online]. Available: http://amzn.to/1aYBqEj

[8] gNewSense. b43. Webpage. [Online]. Available: http://www.gnewsense.org/Documentation/Wireless

[9] Büsch, M. Github repository: b43-tools. Webpage. [Online]. Available: https://github.com/mbuesch/b43-tools

[10] L. Sanabria-Russo, A. Faridi, B. Bellalta, J. Barceló, and M. Oliver, "Future Evolution of CSMA Protocols for the IEEE 802.11 Standard," *CoRR*, vol. abs/1303.3734, 2013, Presented at 2nd IEEE Workshop on Telecommunications Standards: From Research to Stadards. Budapest, Hungary.

[11] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The TCP/UDP bandwidth measurement tool," *http://dast.nlanr.net/Projects*, 2005.

[12] Combs, Gerald and others. (2007) Wireshark. [Online]. Available: http://www.wireshark.org