Technical report: WMP-Editor

Luis Sanabria-Russo

***Abstract*—This report covers the technical aspects of the implementation of CSMA/ECA in Wireless MAC Processors (WMP). Upon each iteration, the abstract will change and its sections will probably lengthen and increase in number. This first version covers the aspects related to WMP-Editor layout and parameters. Also, the last section is composed of the current highlights on the research.**

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

I. Introduction

As was mentioned on the *Future Directions* section in [1], one of the goals in the roadmap is to implement CSMA/ECA in cheap commodity hardware.

This report gathers all the necessary software, equipment and procedures that led to the translation of CSMA/ECA into WMP-Editor [2] [3]. It is intended to anyone who is interested on playing with MAC protocols in real hardware.

II. Flying through WMP's concepts

On the other hand, this report is not written as a survey or an explanation on the subject of Finite State Machines (FSM); and neither details the work of FSM as an abstraction level capable of modifying the default behavior of wireless network cards. Instead the reader will be introduced to the terms used in the documentation as well as how they are appreciated through the graphical interfaces of the different applications used along.

A. Directing the behavior

The WMP allows us to *program* different MAC protocols through a user interface (UI). This interface is what will be called the WMP-Editor from now on. In the WMP-Editor you can build blocks and connect them to other blocks. Some of these blocks can be conditional blocks and redirect the instruction flow according to an evaluation. In turn, every connection between blocks can execute an *action* that can admit *parameters*.

In the WMP-Editor, blocks are *states* of the MAC program waiting for some trigger to unleash further actions and redirect the flow towards another state. Figure 1 shows an overview of the workspace in WMP-Editor.

When following the flow of an example state machine (like the one in Figure 1), it is easy to see how arrows represent *state changes*. These are the ones capable of executing actions during the state transitions. Figure 2, shows how you can edit the parameter of action `0x0D TX_PKT_SCHEDULER` by using the drop-down menu on the left side (setting it to `BK_SLOT=16`).

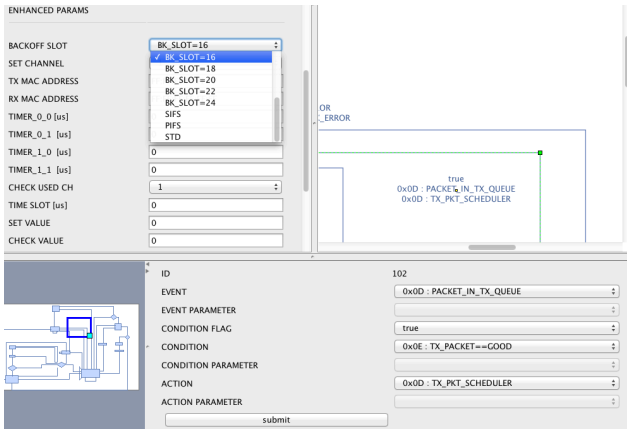

Fig. 2. Changing the parameter of action `0x0D TX_PKT_SCHEDULER`

III. Implementation

It is possible to find everything that is needed to run WMP-Editor at the project's Github repository [3]. But first, it is worthy to mention the exceptions related to hardware.

A. Hardware compatiblity issues

As of the time of this report, it is necessary to have a wireless card with the Broadcom chipset 4311v1 or 4318. This is not an obstacle for testing WMP-Editor, but in order to test the work it is imperative to have a compliant card. Instructions on how to flash the firmware and other compatibility tasks that must be performed, can be found in the documentation folder of [3] (Chapter 8).

B. Running the WMP-Editor

WMP-Editor is written in Java, and also can be downloaded from [3]. Navigating to *wmp-editor/wmpeditor-v2.37/* reveals the `WMPEditor.jar` file. In order to execute it, just `cd` to the mentioned directory and type: `java -jar WMPEditor.jar` to open WMP-Editor.

Once inside, it is possible to open other `.sm` files like the ones under *mac-example/dcf/* through the *file* menu on the top left corner. The `dcf-master.sm` example is shown in Figure 1.

C. Current state of our implementation

We now know that the action `TX_PKT_SCHEDULER` seems to assign a transmission slot. The normal Binary Exponential Backoff (BEB) rule is set when using the option `STD` for this action. `TX_PKT_SCHEDULER` picks a random backoff between 0 and the minimum contention window ($CW_{\min}$) and

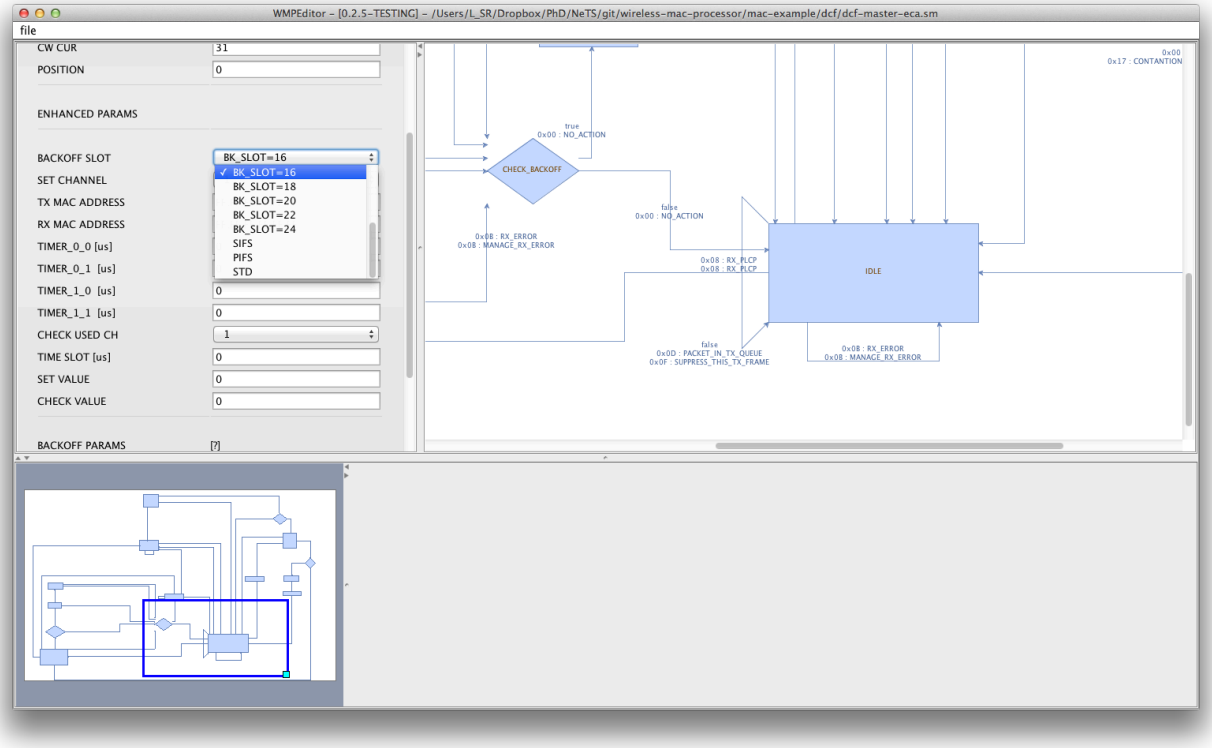

Fig. 1. WMP-Editor layout

decrements it by every passing empty slot until the counter expires. When this happens, a tranmission is attempted.

There is also an option to further modify the contention parameters. Actions `CONTENTION_PARAMS_UPDATE_SUCCESS` and `CONTENTION_PARAMS_UPDATE_FAIL` modify the contention window according to the result of the transmission. It is determined as a success when an ACK is received, and a failure otherwise.

According to the documentation, each of the mentioned actions is ruled by the *Backoff Params* section on the lower left-hand side of WMP-Editor. Successes update the contention window value according to (1), while failures do so as (2).

$$CW = 2 \times \texttt{INFLATION\_MUL} + \texttt{INFLATION\_ADD} \quad (1)$$

$$CW = \frac{2}{\texttt{DEFLATION\_DIV}} - \texttt{DEFLATION\_SUB} \quad (2)$$

The terms in the equation are defined as:

- `INFLATION_MUL`: is a multiplier with default value of 2 up to 8.
- `INFLATION_ADD`: with default value of 1 up to 65535.
- `DEFLATION_DIV`: with a default value of 1 up to 8.
- `DEFLATION_SUB`: with a default value of 65535.

What is curious about these calculations is that they seem to be wrong. After a successful transmission (2), the contention window is reset to zero because the evaluation results in a CW below the allowed limit.

In the failure case (1), assuming $CW_{\min} = 16$, after the action `CONTENTION_PARAMS_UPDATE_FAIL`, $CW = 5$ evertyime. This is obviously wrong.

Furthermore, after a successful or a failed transmission attempt the flow redirects the MAC to the Idle state. Supposing the station has another packet in the queue, then it will execute the `TX_PKT_SCHEDULER` action: recomputing a backoff.

From here, some questions are derived:

- Suppose we adjust `DEFLATION_DIV` and `DEFLATION_SUB` so the contention window (CW) is not reset after a successful transmission (*Hysteresis*). Will `TX_PKT_SCHEDULER` recompute a random backoff $B \in [0, CW]$?
- Can we assign a backoff slot for transmission that depends on the current value of CW?
- How does `STD` work? Can we modify this rule?

REFERENCES

[1] L. Sanabria-Russo, J. Barcelo, and B. Bellalta, "Fairness in Collision-Free WLANs," *ArXiv e-prints*, Feb. 2013.

[2] G. Bianchi. (2010) FLAVIA Project: Flexible Architecture for Virtualizable future wireless Internet Access. Webpage. [Online]. Available: http://www.ict-flavia.eu

[3] D. Garlisi, F. Giuliano, and P. Gallo. (2012) Wireless MAC Processor: a Github code repository. [Online]. Available: https://github.com/ict-flavia/wireless-mac-processor