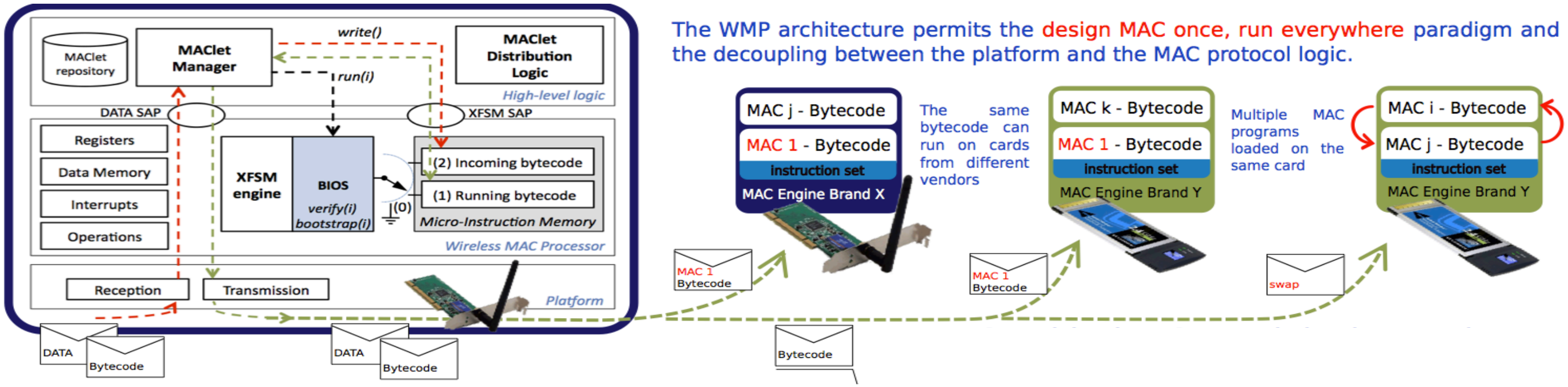
MAClets

Active MAC protocols over hard-coded devices

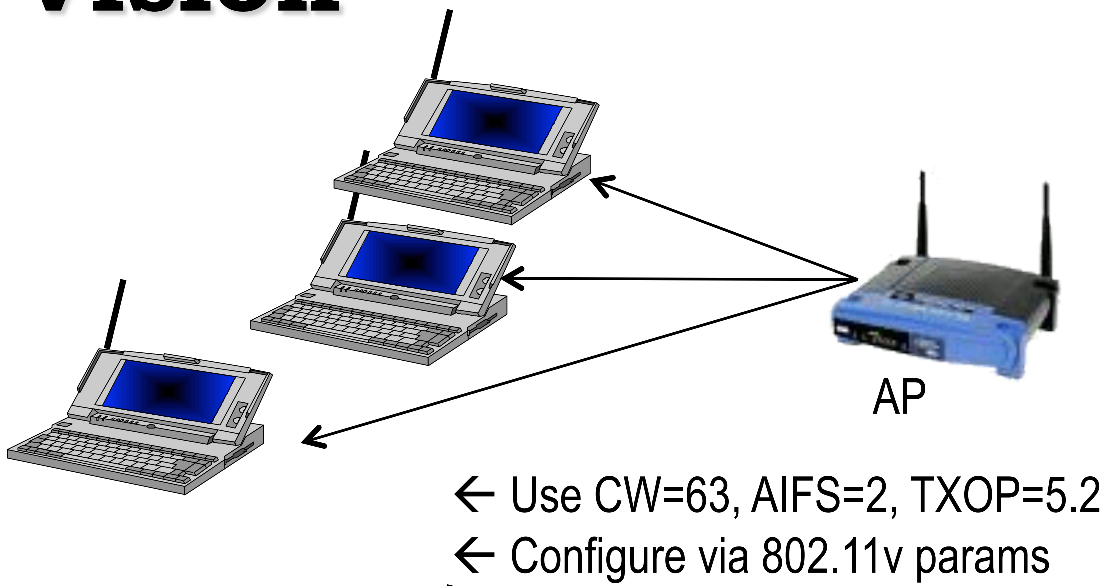
Giuseppe Bianchi, Pierluigi Gallo, Domenico Garlisi,
Fabrizio Giuliano, Francesco Gringoli, Ilenia Tinnirello

CNIT (Univ. Roma Tor Vergata, Palermo, Brescia)

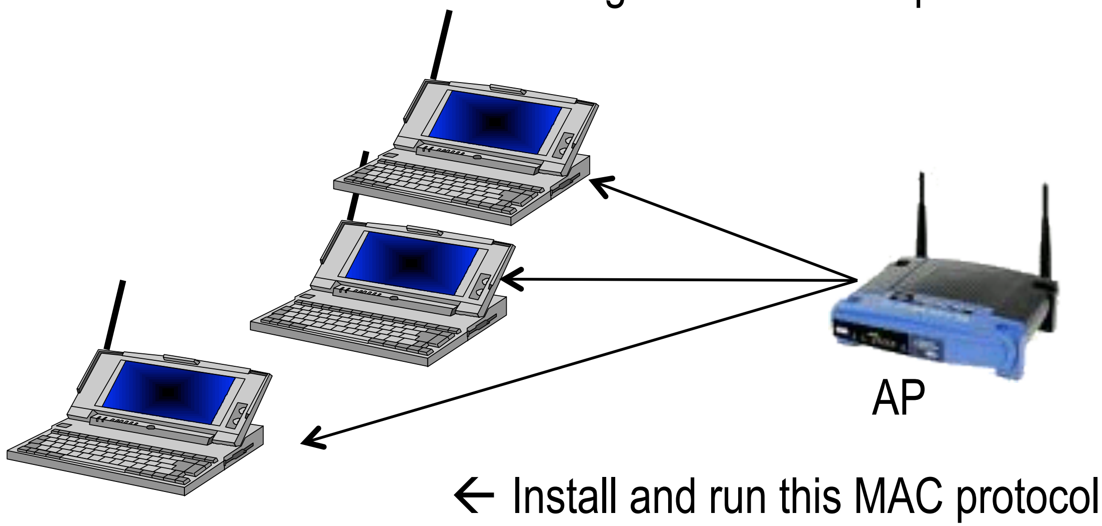


Vision

➔From STA management via parameter settings…



➔To STA management via full MAC stack reprogramming!



<u>Whole</u> MAC protocol stack as a sort of JAVA applet?

…more opportunities…

- ➔ **Flexibily Adapt Access Protocol to scenario/context**
  - ⇨ Dynamic spectrum access
  - ⇨ Niche scenario optimization
    - → home, industrial, …
  - ⇨ Context/application-specific protocol design
  - ⇨ Faster paper-to-field deployment
  - ⇨ Improved support for PHY enhancements
- ➔ **Virtualization**
  - ⇨ Each operator can design its own resource management
    - → frame forging, scheduling, timing, channel switching, PHY selection, …
  - ⇨ Different MAC coexisting on same AP/net

Real world blockers

- ➔ **Lower MAC protocol ops are real time!**
  - ⇨ O(us): TX, RX, slot times, set IFS, set timers, etc
  - ⇨ Driver to NIC interface: too slow → MUST run on NIC

- ➔ **Vendors will HARDLY give us open source, fully programmable, NICs**
  - ⇨ SDR is 20 years old but…
    - → …still no real world commodity SDR NICs
  - ⇨ NIC design extensively leveraging HW
    - → non programmable, unless FPGA NICs…
    - → Your commodity card is NOT an FPGA!
  - ⇨ Why a vendor should renounce to its internal Intellectual Property??

- ➔ **But even if stack gets opened…which programmability model?**
  - ⇨ Current practice (in most cases):
    - → *patch/hack existing SW/FW/HW code base*
    - → Huge skills/experience, low level languages, slow development, inter-module dependencies

Our contribution

- ➔ **Exploiting a new abstraction model for run-time MAC protocol reconfigurations!**
  - ⇨ based on the Wireless MAC Processor (WMP)
    - → INFOCOM 2012
- ➔ **Enabling active MAC protocols and remote MAC injection**
  - ⇨ Ultra-fast (below ms) reconfiguration
  - ⇨ MAC multi-threading
  - ⇨ Virtualization

Learn from computing systems?

- ➔ **1: Instruction sets**
  perform elementary tasks on the platform
  - →A-priori given by the platform
  - →Can be VERY rich in special purpose computing platforms
    - » Crypto accelerators, GPUs, DSPs, etc

- ➔ **2: Programming languages**
  sequence of such instructions + conditions
  - ⇨ Convey desired platform's operation or algorithm

- ➔ **3: Central Processing Unit (CPU)**
  execute program over the platform
  - ⇨ Unaware of what the program specifically does
  - ⇨ Fetch/invoke instructions, update registers, etc

Clear decoupling between:
- platform's vendor → implements (closed source!) instruction set & CPU
- programmer → produces SW code in given language

1: Which elementary MAC tasks?
("our" instruction set!)

➔ **ACTIONS**

⇨ **frame management, radio control, time scheduling**

→TX frame, set PHY params, RX frame, set timer, freeze counter, build header, forge frame, switch channel, etc

➔ **EVENTS**

⇨ **available HW/SW signals/interrupts**

→Busy channel signal, RX indication, inqueued frame, end timer, etc

➔ **CONDITIONS**

⇨ **boolean/arithmetic tests on available registers/info**

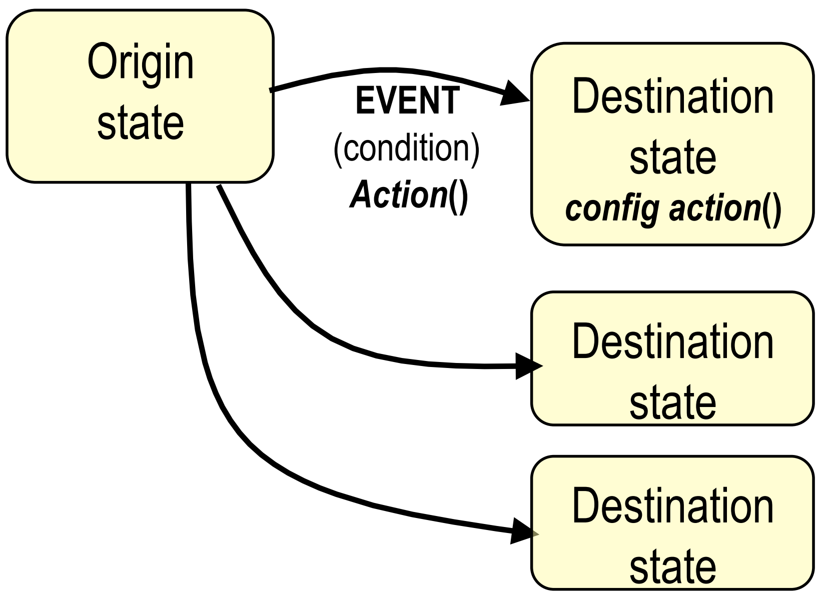
→Frame address == X, queue length >0, ACK received, power level < P, etc

2: How to compose MAC tasks?

("our" programming language!)

➔Convenient "language": XFSM
eXtended Finite State Machines

⇨Compact way for composing available acts/ev/cond to form a custom MAC protocol logic



XFSM formal notation		meaning
S	symbolic states	MAC protocol states
I	input symbols	Events
O	output symbols	MAC actions
D	n-dimensional linear space $D_1 \times \cdots \times D_n$	all possible settings of n configuration registers
F	set of enabling functions $f_i : D \to \{0,1\}$	Conditions to be verified on the configuration registers
U	set of update functions $u_i : D \to D$	Configuration commands, update registers' content
T	transition relation $T : S \times F \times I \to S \times U \times O$	Target state, actions and configuration commands associated to each transition

3: How to run a MAC program?

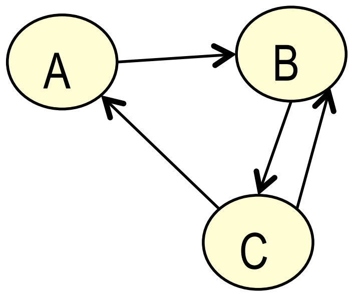
(MAC engine – XFSM onboard executor - our CPU!)

- ➔ **MAC engine: specialized XFSM executor** *(unaware of MAC logic)*
  - ⇨ Fetch state
  - ⇨ Receive events
  - ⇨ Verify conditions
  - ⇨ Perform actions and state transition
- ➔ **Once-for-all "vendor"-implemented in NIC** *(no need for open source)*
  - ⇨ "close" to radio resources = straightforward real-time handling

MAC Bytecode

➔ **MAC description:**

⇨ XFSM



➔ **XFSM ➔ tables**

	A	B	C
A		T(A,B)	
B			T(B,C)
C	T(C,A)	T(C,B)	

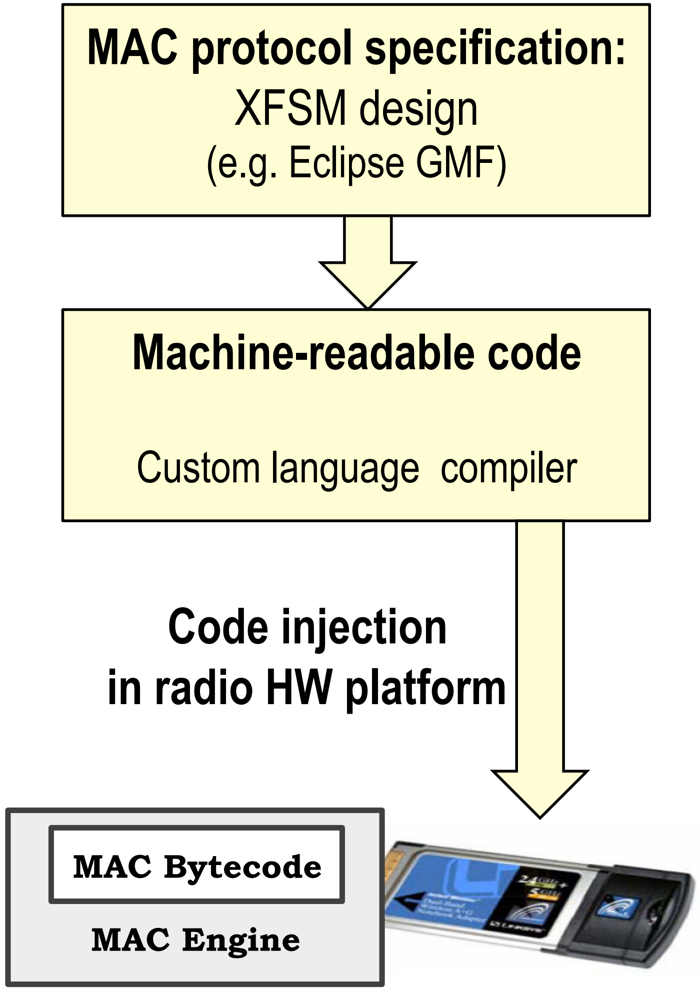
➔ **Transitions**

⇨ «byte»-code event, condition, action

→**Portable over different vendors' devices, as long as API is the same!!**

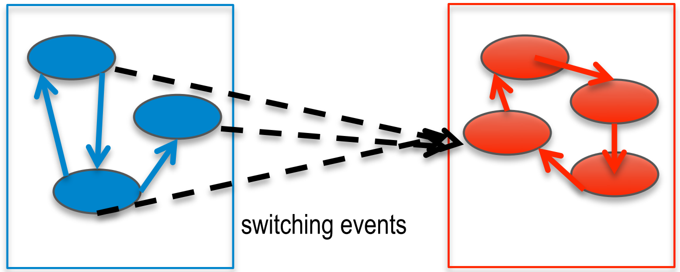
⇨ Pack & optimize in WMP «machine-language» bytecode

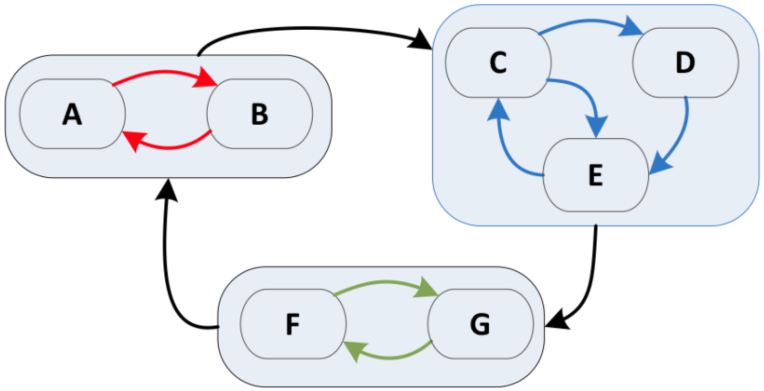
A	T(A,B)	
B	T(B,C)	
C	T(C,A)	T(C,B)



Multi-Thread Support

- ➔ **The MAC Engine does not need to know to which MAC program a new fetched state belongs!**
  - ⇨ Code switching can be easily supported by moving to a state in a different transition table
- ➔ **It is enough to:**
  - ⇨ Define Meta State Machines for programming code switching
  - ⇨ Verify MAC switching events from each state of the program under execution
  - ⇨ Re-load system configuration registers at MAC transitions





From MAC Programs to MAClets

➔ Upload MAC program on NIC from remote

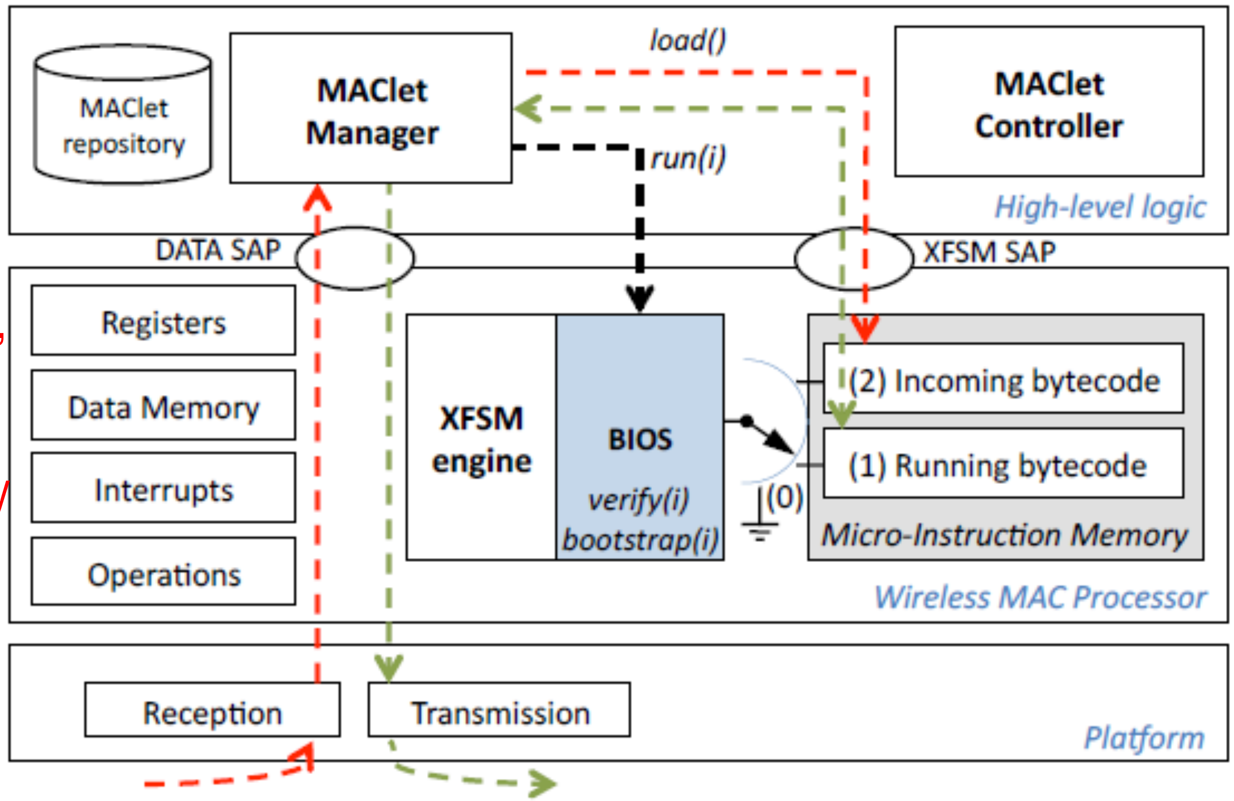
- ⇨ While another MAC is running
- ⇨ Embed code in ordinary packets

➔ WMP Control Primitives

- ⇨ load(XFSM)
- ⇨ run(XFSM)
- ⇨ verify(XFSM)
- ⇨ switch(XFSM1, XFSM2, ev, cond)

➔ Further primitives

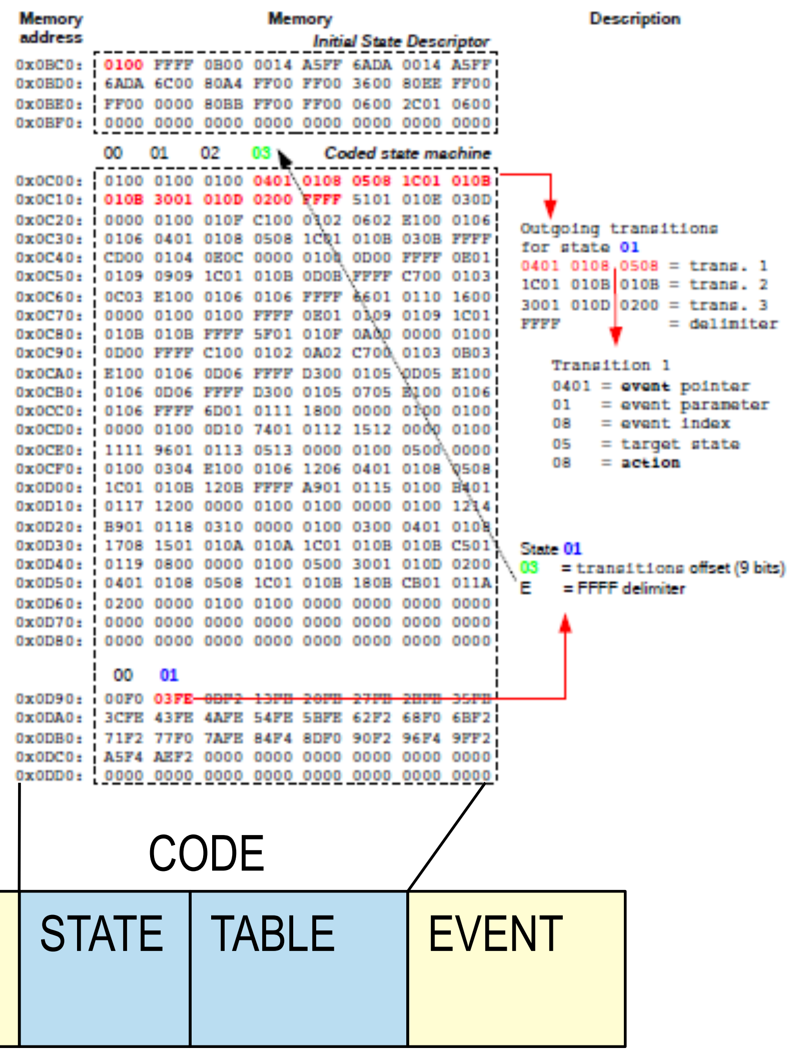
- ⇨ Distribution protocol (run by the MAClet Manager
- ⇨ Synchro support for distributed start of same MAC operation)

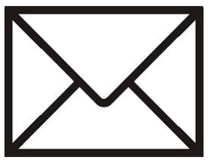


"Bios" state machine: DEFAULT protocol (e.g. wifi) which all terminals understand

MACLets

- ➔ **An entire MAC program can be coded in a single frame!**
  - ⇨ our abstractions and machine codes allow to code DCF in about 500 bytes
- ➔ **Other fields:**
  - ⇨ type (distribution protocol and action messages)
  - ⇨ destination IDs
  - ⇨ initial state
  - ⇨ command (load, run, switch..)
  - ⇨ activation event

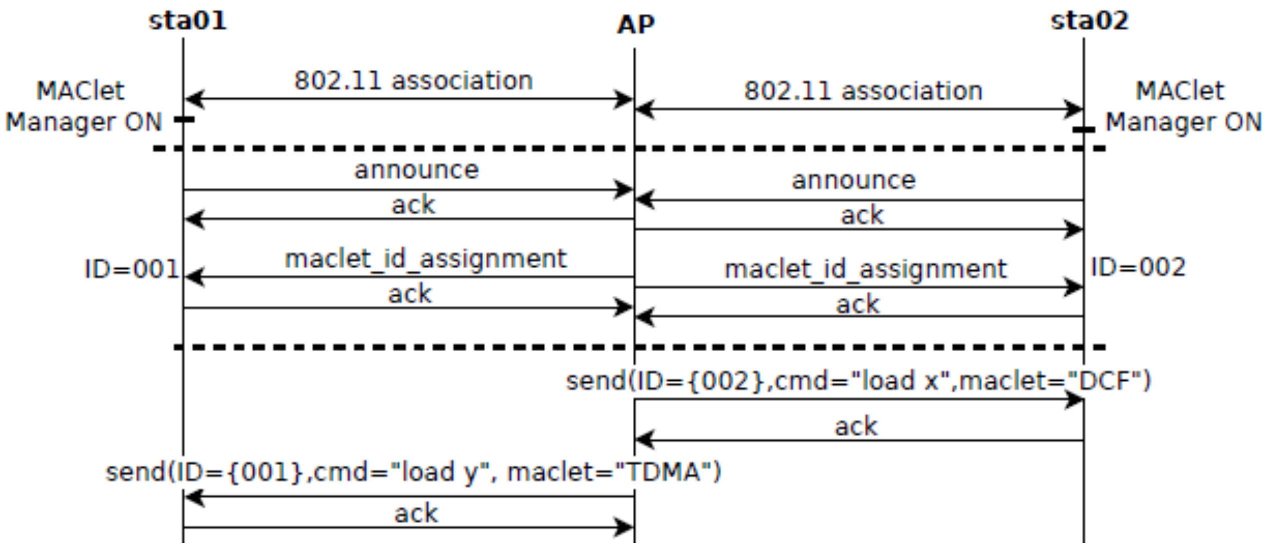


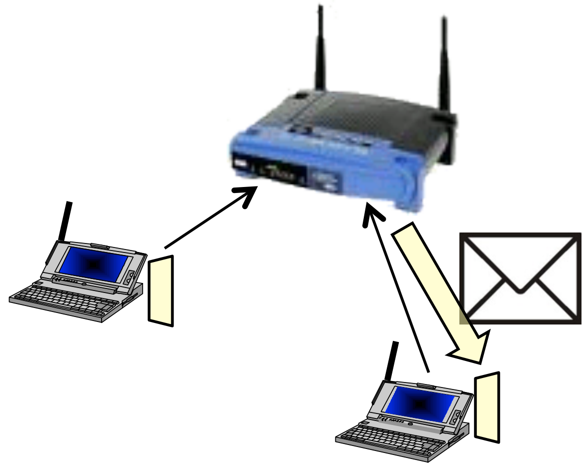

MACLET

MAClet Distribution Protocol

➔ **Defined for allow the AP to remotely access the WMP control interface of the associated nodes**

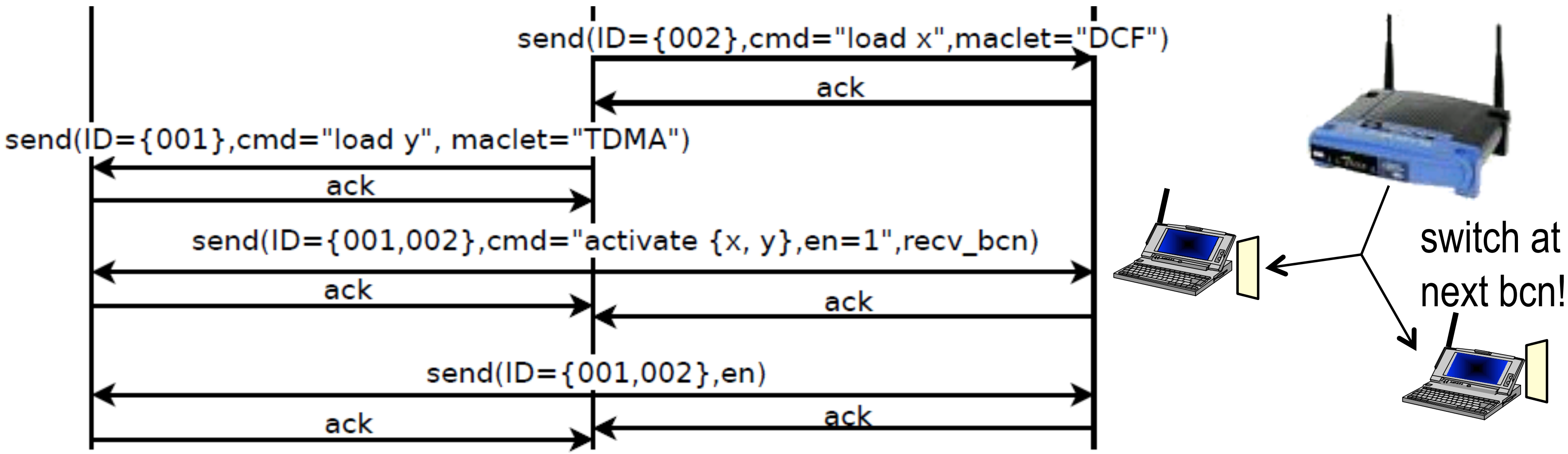
- ⇨ Binding MAClet Managers of each node to the AP MAClet Controller
  - → Notification of activation/de-activation, ID assignment
- ⇨ Transporting Action Messages coding WMP commands (load/run/switch) and MAC programs





Synchronization Primitives

- **When to switch to a new MAC protocol?**
  - Mechansims available, but final solutions left to the MAClet programmers
- **Triggering events and signals**
  - No trigger: asynchronous activation
  - Control frames sent by the AP
  - Expiration of relative or absolute timer
    - Absolute timers built on top of the time-synchronization function included in DCF
  - 1-way or 3-way handshakes

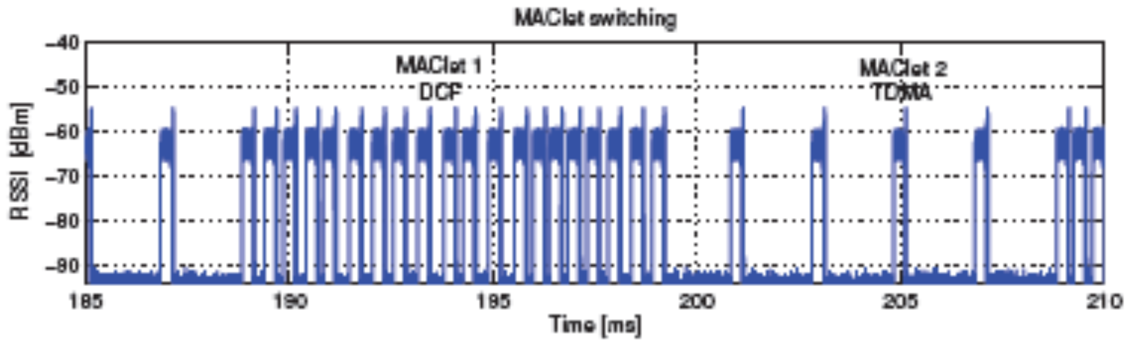


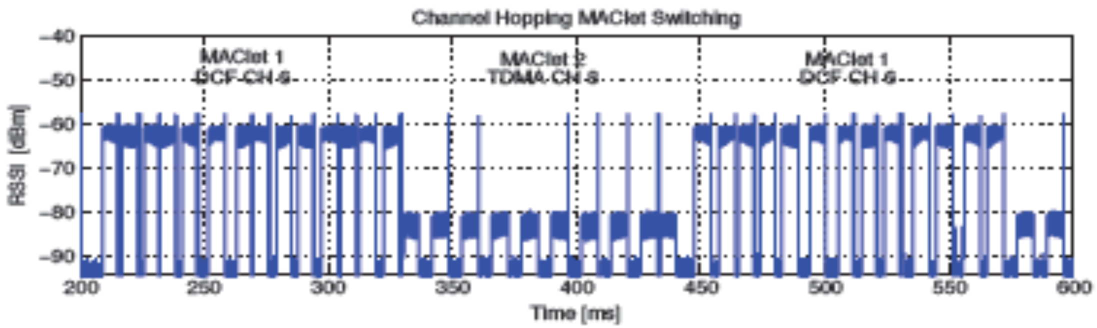
Switching Operation

➔ **From a configuration to another..**

➔ **From a program to another!**

⇨ (with latency of about 1 microsecond)





Implementation at a glance

(on commodity hardware!)

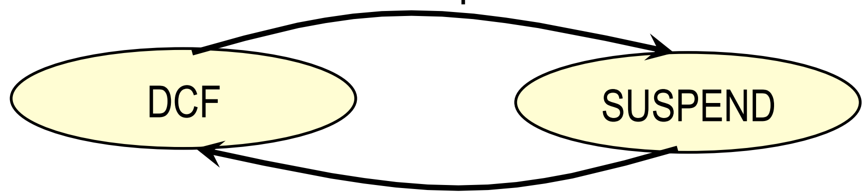
Reference platform: broadcom Airforce54g 4311/4318

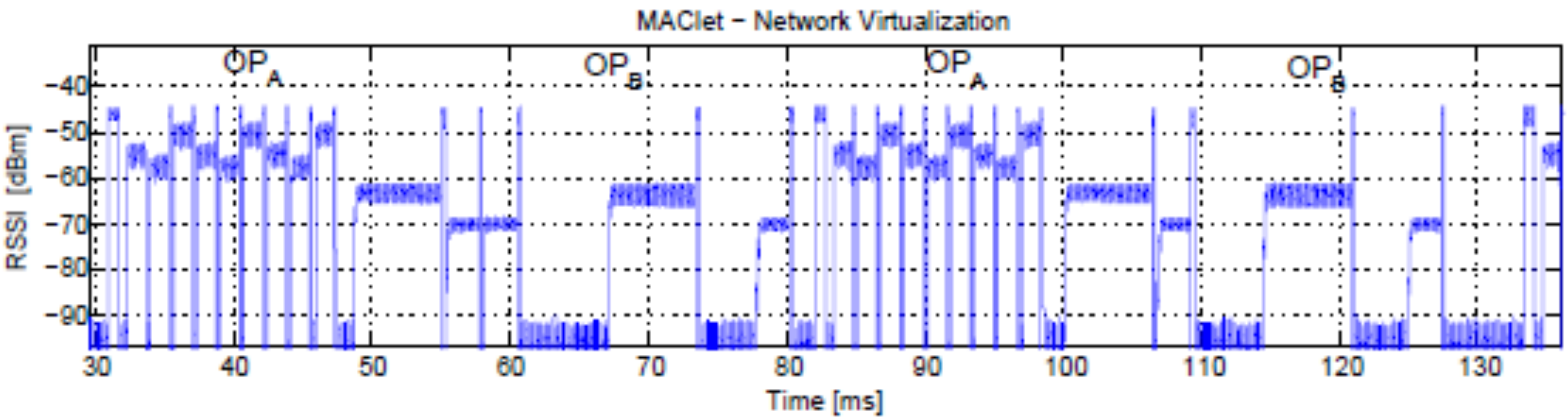
- **WMP:**
  - replace both Broadcom and openFWWF firmware with
    - Implementation of actions, events, conditions
    - MAC engine: XFSM executor
  - Develop “machine language” for MAC engine
    - Custom made “bytecode” specified and implemented
- **WMP Control Architecture:**
  - At firmare level:
    - WMP Control Interface
  - At the application level:
    - MAClet Manager: receive/transmit MAClets and other messages of the MAClet Distribution Protocol
    - MAClet Controller: Intelligent part of the system, dealing with network-level decisions
    - Current implementation based on classical client-server model!

Application Examples

AP Virtualization with MAClets

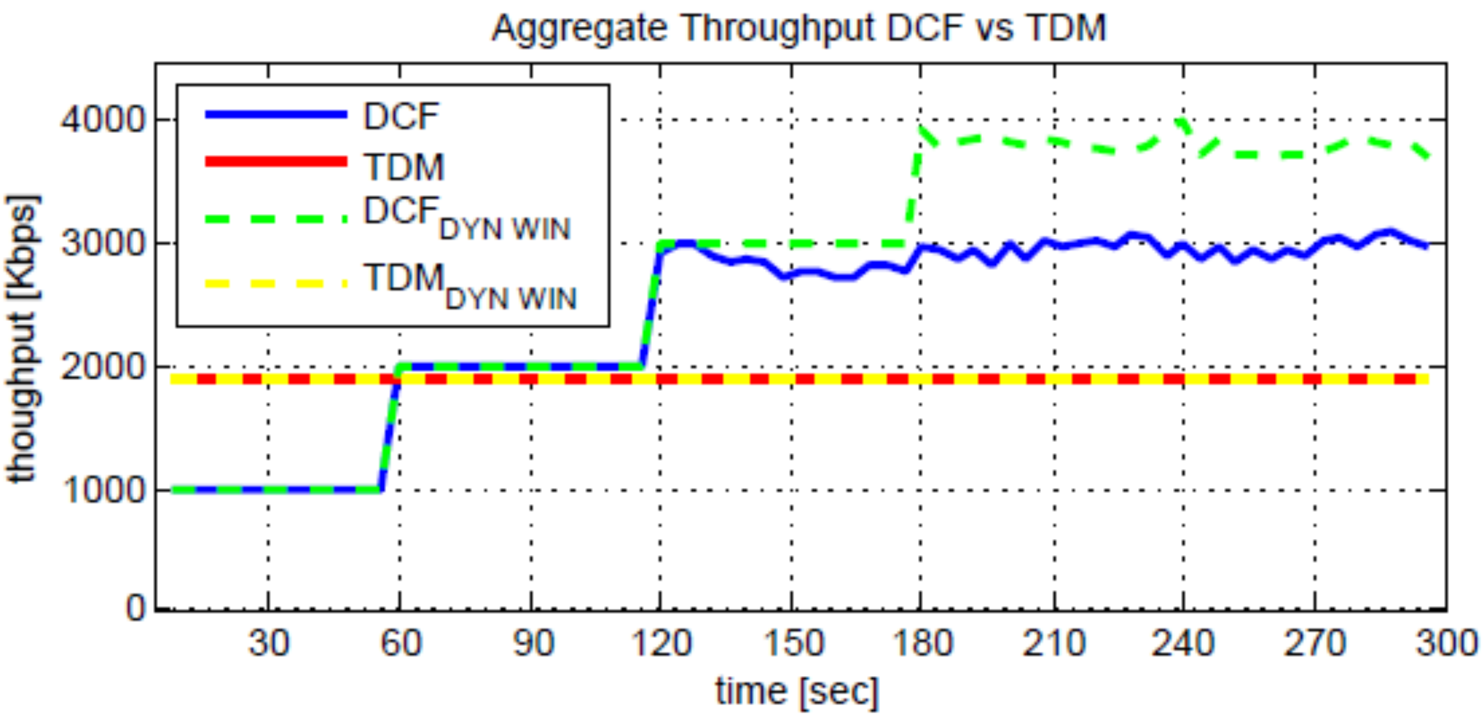
- ➔ **Two operators on same AP/infrastructure**
  - ⇨ A: wants TDM, fixed rate
  - ⇨ B: wants best effort DCF
- ➔ **Trivial with MAClets!**
  - ⇨ Customers of A/B download respective TDM/DCF MAClets!
- ➔ **Isolation via MAClet design**
  - ⇨ Time slicing DESIGNED INTO the MAClets! (static or dynamic)





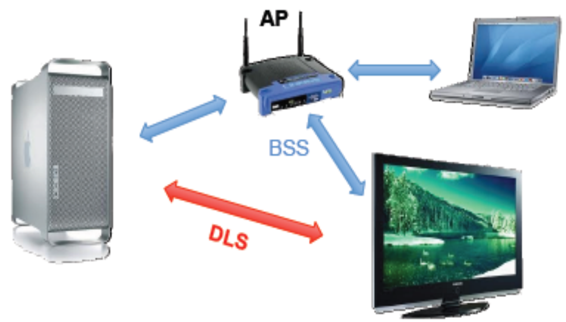
Throughput Performance

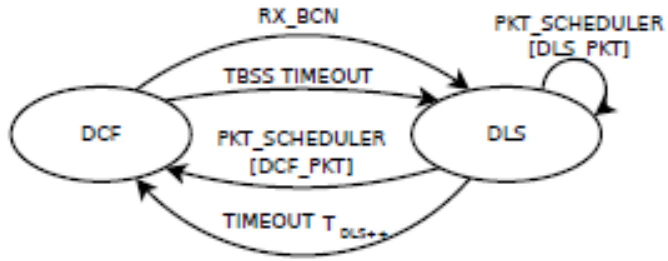
3 FIXED stations @ 0.63 Mbps vs. 5 BEST stations @ 1Mbps

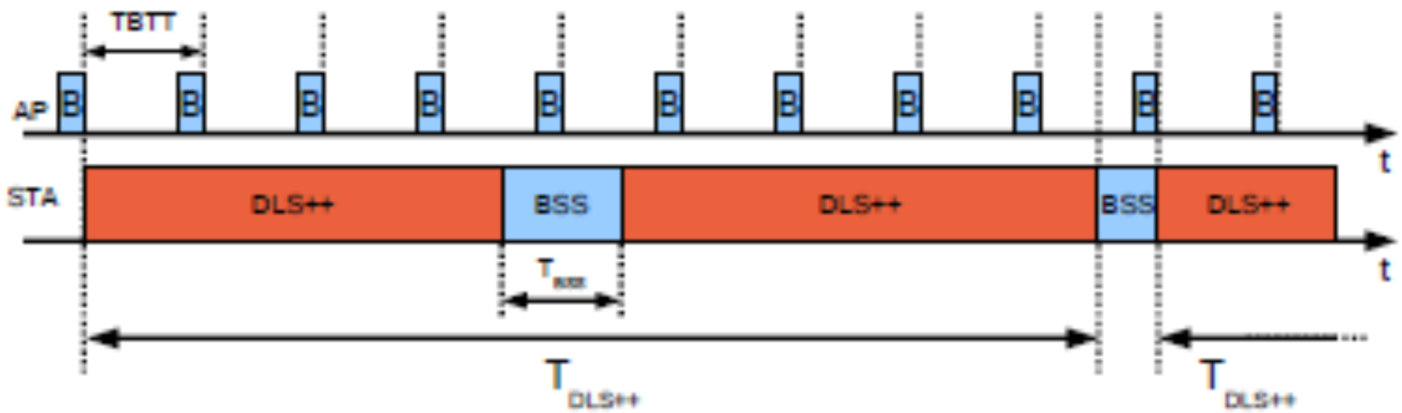


Home Networks with MAClets

- ➔ **Heterogeneous applications at home**
  - ⇨ E.g. Video streaming and web browsing
- ➔ **Trivial with MAClets!**
  - ⇨ The Smart TV is not expected to implement any specific standard amendment
  - ⇨ DLS protocol can be loaded when necessary
  - ⇨ The network owner can push further optimizations:
    - → additional channel for direct link channel, without losing association
    - → Additional channel for direct link with greedy backoff



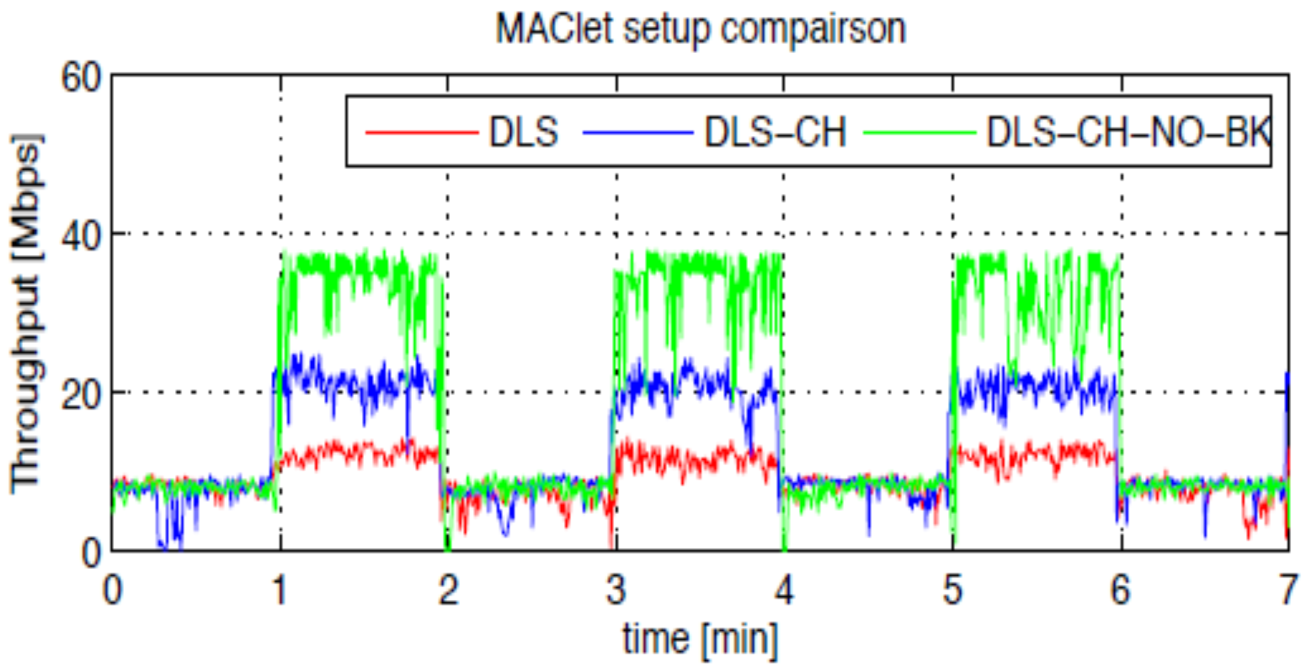




Throughput Performance

➔ **Experiment with a periodic switching from DLS++ to DCF**

⇨ For testing multithreading and synchronization mechanisms



Conclusions

- ➔ **New vision:**
  - ⇨ MAC no more an all-size-fits-all protocol
  - ⇨ Can be made context-dependent
  - ⇨ Complex scenarios (e.g. virtualization) become trivial!
- ➔ **Very simple and viable model**
  - ⇨ Byte-coded XFSM injection
  - ⇨ Does NOT require open source NICs!

- ➔ **Next steps**
  - ⇨ We focused on the «act» phase; what about the decision and cognitive plane using such new weapons?
  - ⇨ can we think to networks which «self-program» themselves?
    - →Not too far, as it just suffices to generate and inject a state machine…

Public-domain Platform

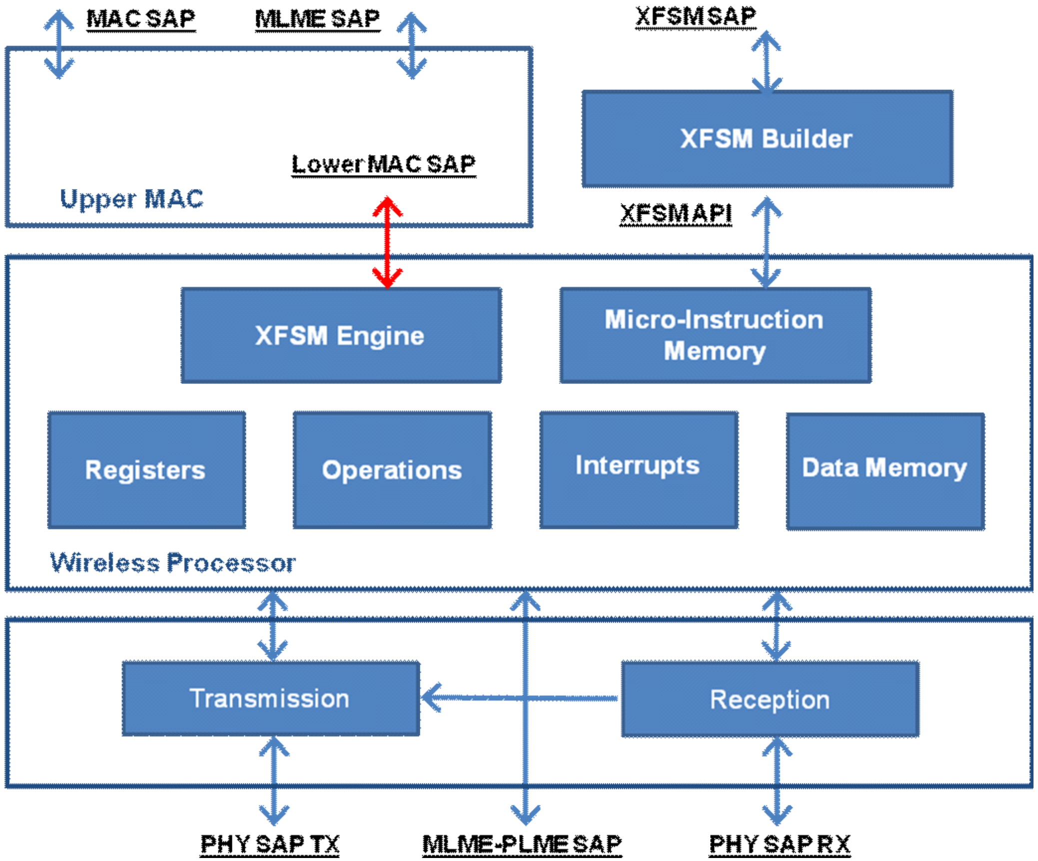
- **Supported by the FLAVIA EU FP7 project**
  - http://www.ict-flavia.eu/
    - general coordinator: giuseppe.bianchi@uniroma2.it
    - Technical coordinator: ilenia.tinnirello@tti.unipa.it
- **Public domain release in alpha version**
  - https://github.com/ict-flavia/Wireless-MAC-Processor.git
  - Developer team:
    - ilenia.tinnirello@tti.unipa.it
    - domenico.garlisi@dieet.unipa.it
    - fabrizio.giuliano@dieet.unipa.it
    - francesco.gringoli@ing.unibs.it
- **Released distribution:**
  - Binary image for WMP
  - Source code for MAClet Manager
  - You DO NOT need it open source!
    Remember the "hard-coded" device philosophy…
    - Conveniently mounted and run on Linksis or Alix
  - Source code for everything else
  - Manual & documentation, sample programs

FLAVIA

SEVENTH FRAMEWORK PROGRAMME

WMP Overall architecture

from protocol-specific hard-coded device to protocol executor

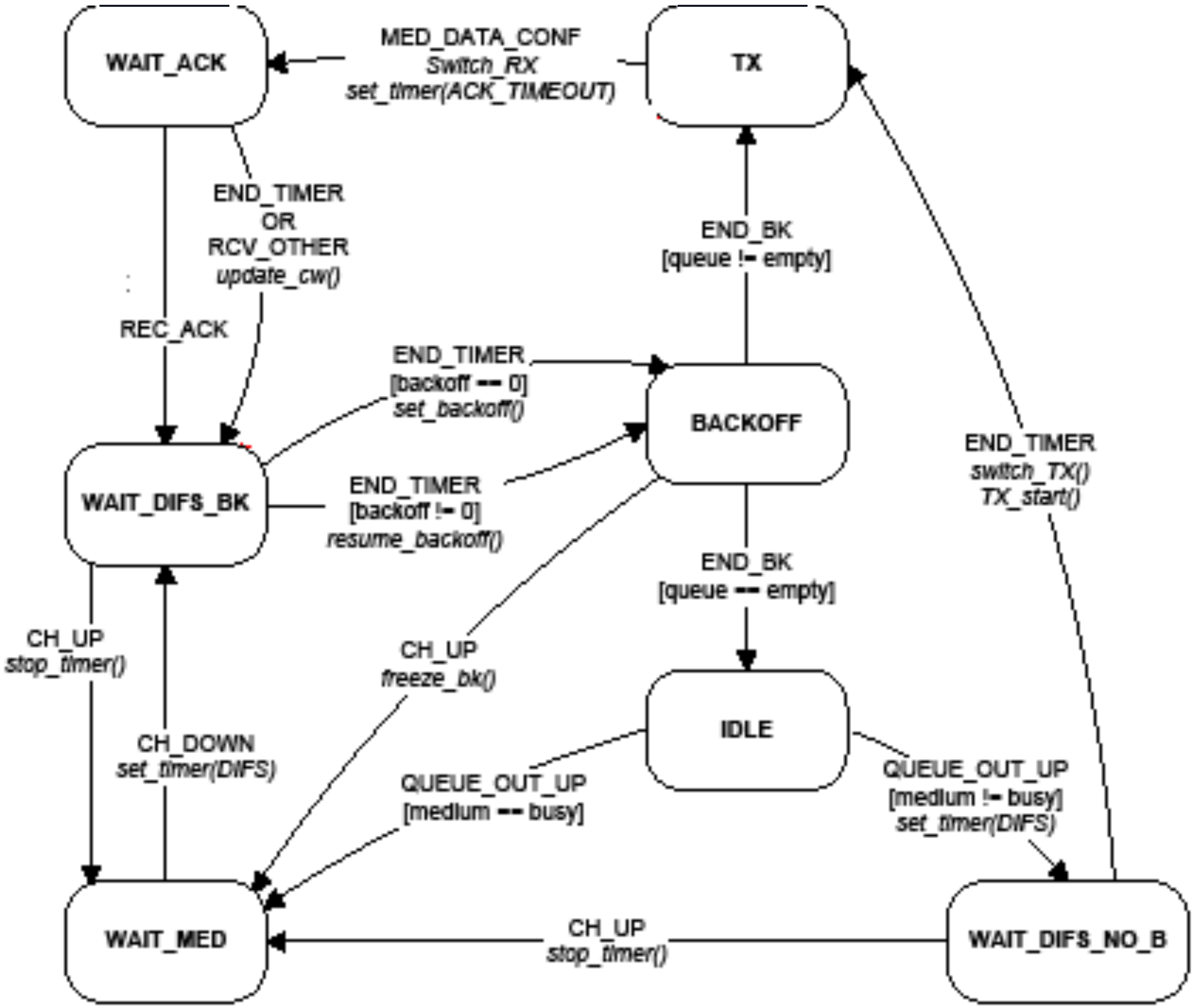


- ➔ **<u>MAC Engine:</u> XFSM executor**
- ➔ **<u>Memory blocks:</u> data, prog**
- ➔ **<u>Registers:</u> save system state (*conditions*);**
- ➔ **<u>Interrupts block</u> passing HW signals to Engine *(events)*;**
- ➔ **<u>Operations</u> invoked by the engine for driving the hardware (*actions*)**

The MAC engine works as a Virtual MAC Machine

XFSM example: legacy DCF

simplified for graphical convenience



Actions:
set_timer, stop_timer, set_backoff, resume_backoff, update_cw, switch_TX, TX_start

Events:
END_TIMER, QUEUE_OUT_UP, CH_DOWN, CH_UP, END_BK, MED_DATA_CONF

Conditions:
medium, backoff, queue