FLAVIA
FLexible Architecture
for Virtualizable wireless future Internet Access

Grant Agreement: FP7 - 257263

Specific Targeted Research Project

FLAVIA

FLexible Architecture for Virtualizable wireless future Internet Access

Deliverable Report

D 4.1.2 – Revision of 802.11 architecture and interfaces specification

Deliverable title	802.11 architecture and interfaces specification
Version	1.0
Due date of deliverable(month)	M24
Actual submission date of the deliverable (dd/mm/yyyy)	02/07/2012
Start date of project (dd/mm/yyyy)	01/07/2010
Duration of the project	36 months
Work Package	WP4
Task	Task 4.1
Leader for this deliverable	IMDEA
Other contributing partners	CNIT, NEC, TID, MOBIMESH, NUIM, AGH, IITP
Authors	Pablo Serrano, Vincenzo Mancuso, Pablo Salvador (IMD), Ilenia Tinnirello, Pierluigi Gallo, Pierpaolo Loreti, Francesco Gringoli, Claudio Pisa (CNIT), Antonio Capone, Stefano Paris (MOBI), Xavier Pérez Costa (NEC), Eduard Gomà, Yan Grunenberger (TID), Ken Duffy, David Malone, Paul Patras (NUIM), Marek Natkaniec, Szymon Szott, Krzysztof Loziak, Janusz Gozdecki, Marek Sikora (AGH), Artem Krasilov (IITP)
Deliverable reviewer	Paul Patras (NUIM), Francesco Gringoli (CNIT), Yan Grunenberger (TID)

Deliverable abstract	The Document reviews the specification of the FLAVIA architecture (modules, functionality and interfaces) for the case of 802.11 technologies
Keywords	802.11, architecture, interface, services, specification, flexibility, modularity, virtualization

Project co-funded by the European Commission within the Seventh Framework Programme		
DISSEMINATION LEVEL		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

PROPRIETARY RIGHTS STATEMENT

STATEMENT OF ORIGINALITY

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

Executive summary

This report reviews the architecture specified for a FLAVIA-based 802.11 [1] system. This document does not aim to be a self-contained description of the final design, but an update on D4.1.1 [2] building on the "intra-workpackage" feedback described in D4.2 [3], and the work carried out in WP2, described in D2.1.1 [4], D2.2.1 [5] and their corresponding updates D2.1.2 [6] and D2.2.2 [7].

The FLAVIA design addresses the structure and functionality of an 802.11 framework able to support the following key features: **modularity**, in terms of defining different 802.11 MAC services; **flexibility**, in terms of dynamic or static configurability of the 802.11 MAC; and **virtualization**, in terms of managing parallel independent 802.11 MACs accessing the same system resources. Based on these three principles, we outline in Section 2 how basic elements defined by the FLAVIA architecture can be instantiated to deploy an 802.11 MAC node. Then, in Section 3 we introduce the Wireless MAC Processor (WMP) entity, key component of the FLAVIA architecture, which enables developing and extending the 802.11 MAC low-level functionalities. We focus on the specification of the set of primitives and, the events, conditions and actions that specify the APIs for contention-based systems. Consequently, Section 4 describes the 802.11 Service and Function modules. First, we expose the functionality of the Service Scheduler and the Function Container, which provide modularity by means of the composition and instantiation of different 802.11 service modules. Second, we present an update on some existing services and add new ones, such as, Misbehaviour Detection and Reaction. Third, we introduce the most representative functions of this 802.11 architecture.

While the above sections deal with the "data plane" of the 802.11 architecture, in Section 5 we outline the 802.11 control subsystem, which builds on three elements: the Consistency Manager, the Information Base and the Virtualization manager. The Consistency Manager coordinates the access by several running services to common resources, avoiding possible inconsistencies. The Information Base is the common knowledge base that stores the configuration parameters and exposes the possible supported values. The Virtualization manager enables the execution of various MAC instances on each virtual interface by scheduling the access to the hardware.

The intra- and inter-node communication is presented in Section 6, fostering the exchange of information among the 802.11 FLAVIA components and nodes. More specifically, we provide a detailed description of an inter-node communication scheme that builds on existing technology, and describe the architectural components that are interfaced to the MAC services (e.g., the instantiation of a FLAVIA control subsystem specifically meant for 802.11).

Finally, in Appendix A we provide a detailed specification of an Advanced Data Transport service (with backwards compatibility), in order to illustrate the interaction between the different entities in the FLAVIA architecture.

1 Introduction

FLAVIA defines a new architecture supporting modularity, flexibility and virtualization. This is achieved through the specification of new services, functions and programmable interfaces that make the medium access control adaptable, easily and fast reconfigurable. In addition, it also enables the possibility to dynamically load and customize MAC services during real-time operation of the wireless devices.

One main contribution of WP4 is that of "instantiating" the architecture specified in WP2 for the case of 802.11 MAC, that is, to specify and prototype a programmable MAC framework for contention-based technologies based on 802.11 MAC.

Following the work specified in D4.1.1 [2], and after the intra- and inter-WP feedback (within WP4 and from WP2, respectively), we review the FLAVIA architecture to enable full 802.11 MAC support and present the main updates with the aim of providing a more complete report on the final architecture design. Therefore this document is not a "stand-alone" deliverable, but concentrates on the main updates over the previous architectural document.

In this second deliverable of WP4 we analyse an 802.11 node extended with some exemplificative non-standard functionalities. We specify how the different 802.11-based MAC services are decomposed into smaller functions and further commands. In addition, we specify the interfaces required to support MAC operation and to control the behaviour of the wireless device. By considering both "legacy" services, like power saving or MAC management, and "innovative" services like e.g., SuperSense or Misbehaviour Detection and Reaction, we illustrate how the FLAVIA architecture allows to orchestrate existing and new 802.11-like MAC features. Moreover, we describe in detail the MAC services and control plane of contention-based technologies, providing message sequence charts (MSCs) that illustrate the operation and interactions among modules. A detailed specification for each module considered for prototyping and demonstration was provided in D4.2 [3].

As an update, we include in this document the specification of the FLAVIA control subsystem for 802.11 MAC, presenting the set of primitives and interfaces that command and organize the desired system behaviour.

In addition, we include herein the description of the interface to support the inter-node communication, which is based on the Generic Advertisement Service (GAS) protocol defined in the 802.11u standard [8].

2 FLAVIA architecture overview

The 802.11 Architecture proposed is aligned with the general FLAVIA architecture (Figure 1), which has been designed according to the three already known principles: modularity, flexibility and virtualization.

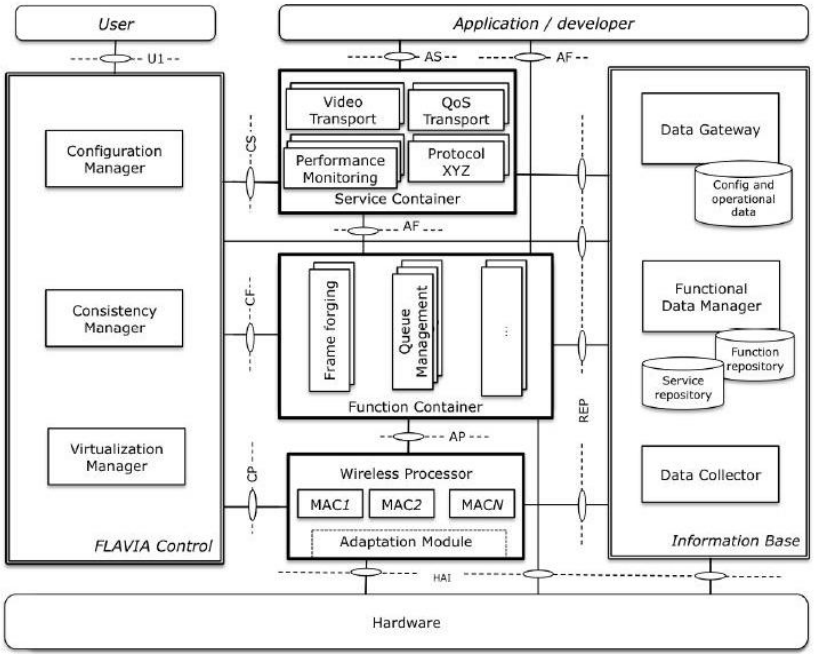


Figure 1: FLAVIA high-level view: framework architecture

The FLAVIA-alike architecture depicted in Figure 2 has been tuned for an 802.11 node that is compound of five main elements:

Wireless MAC processor: The architecture envisioned by FLAVIA could not evolve without hardware support. The Wireless MAC Processor (WMP) is an essential component of the FLAVIA architecture that handles hardware events and executes medium access programs in terms of loadable Finite State Machines. It is responsible for the direct interaction with the hardware modules that represent the lowest level of functional resources of the system. It also works as a kind of function container accelerator, for instantiating the functions heavily interacting (under strict time constraints) with the hardware. An example of these functions is the scheduling of the medium access instants.

Service scheduler: Architecture element in charge of instantiating services, which are composed of functions. A service implements MAC-layer functionalities. New services are specified compared to an 802.11 legacy node, such as SuperSense or Misbehaviour providing to our framework with a higher modularity.

Function container: Architecture element handling the set of running instances of functions. The loading and execution of the functions is done by means of the FLAVIA

control subsystem. A function, as well, makes use of the commands supplied by the hardware. Therefore, the set of functions designed and implemented in the architecture builds on the features already supported by existing hardware, but it also foresees extra capabilities that could enhance contention-based communications in the future.

The existence of the separate sets of services and functions increases the modularity of the architecture, as several instances of the same service are allowed to coexist and each of them may access the function container independently. Moreover, different services may utilize common functions, differentiated by a set of parameters or state variables, which proves the flexibility of the model, and coordinated by the control subsystem.

FLAVIA Control: Entity that manages the loading and changes of context of the different services and functions. It is composed of two entities: the Consistency Manager (CM) and the Virtualization module. The first one is responsible for intra- and inter-node configuration and parameter detection, whereas the last one allows creating and executing several medium access control machines running on top of a unique physical device.

Information Base: Architecture component responsible for managing different data/parameters shared by different services. The data gateway shares the data among the different FLAVIA modules and keeps it consistent. The data collector is the responsible of collecting and gathering different data types, such as system state or hardware parameters. This information is obtained directly from the hardware of by interacting with the WMP.

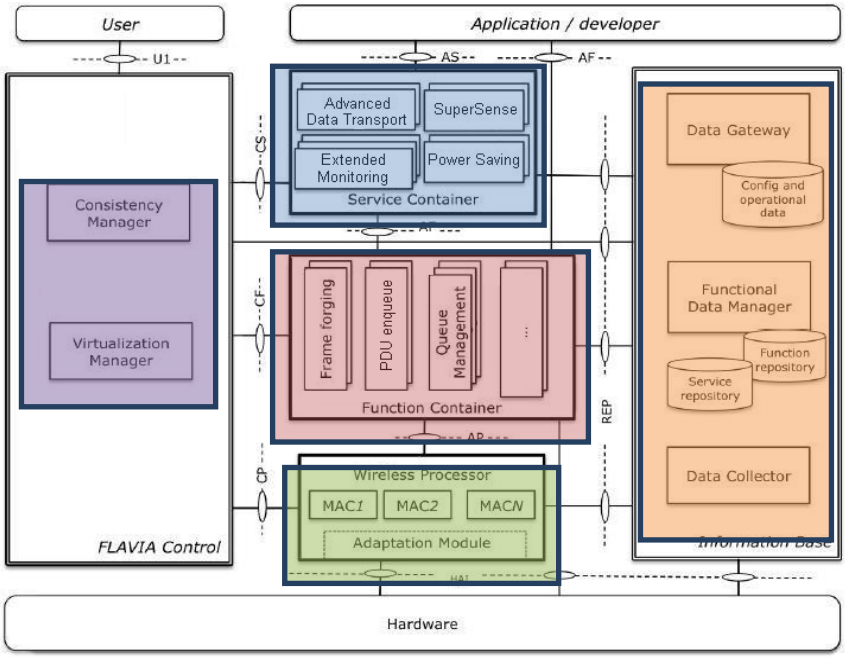


Figure 2: FLAVIA high-level view: 802.11 framework architecture

3 Wireless MAC Processor

The Wireless MAC Processor (WMP) [9][10] is the architecture component devised to run the low-level MAC operations defined in terms of state machines strictly interacting with the hardware. Starting from an initial (default) state, the WMP waits for events that trigger state transitions. The actual transition can be enabled or disabled by the verification of a Boolean condition, and a transition can trigger the execution of one hardware action before moving to the new state. Multiple state machines can be loaded on the WMP in order to simultaneously support different MAC programs(i.e., virtualization) by means of code-switching techniques (as described in D2.2.2 [7]).

This approach supports flexibility in the creation of ad-hoc systems tailored for specific applications, network topologies and environment conditions, without requiring expensive hardware platforms. Indeed, in D4.2 [3] we proved that a WMP can be implemented even on commercial WiFi cards, and that **significant performance gain can be obtained by simply programming the hardware parameterized control** (the configuration registers) **and the action scheduling** rather than the full hardware system (as in the case of software-defined radio). Therefore, the WMP is particularly important for contention-based systems, where:

- Nodes usually have a limited complexity.
- The inter-node coordination and signalling may dramatically change from a network configuration to another (because they are mostly based on peer-to-peer relationships).
- The spectrum availability and medium conditions are very heterogeneous in absence of any centralized planning. For these reasons, the design of the WMP API has originally started from the analysis of contention-based systems and has been lately extended to the scheduled system.

3.1 API for contention-based systems

Although in D2.2.2 [7] we enumerated the list of events, conditions and actions envisioned for a generic programmable radio system, in this report we make such list specific for contention-based systems based on the analysis of DCF lower-MAC operations.

Events

Events are signals generated by the hardware and trigger the execution of actions on the WMP. Starting from the general events defined in the FLAVIA architecture, we specialize and extend some events as shown in Table 1.

General Events	802.11 Events
END_TIMER	END_TIMER
	ACK_TIMEOUT
CH_UP	CH_UP
CH_DOWN	CH_DOWN
RCV_PLCP	RCV_PLCP
RCV_END	RCV_COMPLETE
RCV_DATA	
RCV_ACK	
COLLISION	RX_ERR
	TX_READY
MED_DATA_CONF	TX_ERROR
MED_DATA_START	TX_IN_PROGRESS
MED_DATA_END	TX_END
	TX_10us_ELAPSED
QUEUE_OUT_UP	QUEUE_OUT_UP
QUEUE_IN_OVER	QUEUE_IN_OVER

Table 1: List of WMP events to define contention-based MAC programs

Most of these events have a meaning that can be immediately associated to the event name as described in D4.2 [3]. Comparing the two columns we observe that for defining the DCF low-level operations, we prefer to explicitly specialize a new timer expiration event in terms of ACK timeout, to interpret the collision event as a reception error, and to slightly rename the events describing the signals from the transmitter sub-system (MED_DATA_CONF/ START/ END) for including the event source explicitly.

Note that we also introduce the TX READY event, for signalling the DCF low-level operations to be performed before any transmission have been completed. Specifically, each transmission requires running four different phases:

- Confirm the presence of a frame in the queue and validate the hardware configuration according to the transmission parameters.
- Set up the correct inter-frame space (SIFS, PIFS, DIFS, AIFS) and the backoff when necessary.
- Wait for this time to expire.
- Prepare the frame for actual transmission.

At this point the event TX_READY is generated, activating the transmitter. In addition, the event corresponding to the transmission end is mapped into two events: a signal generated at the end of transmission (TX_END) and a signal generated after a time interval that guarantees that noise measurements at the receiver side are not affected

by the transmitter switch off (TX_10us_ELAPSED).

Conditions

Conditions are applied to state registers that are not explicitly included in the protocol states. These registers correspond both to the hardware configuration and to additional global parameters. Therefore, for the DCF low-level operations we specify all the envisioned global parameters to be explicitly considered for enabling or not state transitions. Note that, the concept of condition actually includes the logical operation to be applied to the relevant register, which is usually expressed in terms of a threshold or a value comparison.

General Conditions	802.11 Conditions
dstaddr == value	dstaddr == value
	srcaddr == value
myaddr == value	
	timer(i) == on
queue_length > value	queue_length > value
queue_type == value	
cw < value	cw < value
cwmin == value	cwmin == value
cwmax == value	cwmax == value
backoff == 0	backoff == 0
frame_length > value	
frame_type == value	rx_frame == value
channel == value	channel == value
power > value	
ACK_on == value	need_wait_ack == true
	need_send_ack == true
	incoming_packet == good

Table 2: List of WMP conditions to define contention-based MAC programs

As in the previous case, the 802.11 conditions list presents differences with respect to the general WMP condition list (see Table 2). For example, the conditions on the queue type or the queue length (used for supporting QoS or enabling the RTS/CTS handshake) are not included in the core of DCF operations, since standard DCF does not support QoS. As well, we add a condition on the source address and, specify the values of each possible register (e.g., the frame type: ACK, BEACON, DATA), and differentiate the ACK activation condition at the receiver and transmitter sides. We also include a condition on the state of the enqueued packets. As the WMP may rely on multiple timers, an explicit register is dedicated to signal the state (activated or not) of each timer.

Actions

Actions are the hardware-specific operations performed by the WMP during state transitions. As well as in previous cases, actions, presented in Table 3, are specialized by considering a reference 802.11 transceiver (e.g.: the Broadcom card used during the prototyping activities).

The set/get configuration actions have been currently limited to the selection of the operating channel (mapped into a dedicated *set_channel()* action) and to the configuration of the transmission rate according to the parameters indicated by the descriptor associated to each outgoing packet (*tx_info_update()*). In addition, we add an explicit action for performing noise measurements, noise_measurement(), as most of the transceivers allow a similar operation that can be useful for many applications, e.g.: setting gain control. The actions related to the reception process have been divided into two phases:

- The reception of the PLCP (*rx_PLCP()*) dedicated to the identification of a valid preamble and to the preparation of the MPDU demodulation.
- The demodulation of the MPDU during which the ACK frame should be prepared in case the received packet requires it (*rx_complete()*).

General Actions	802.11 Actions
set/get(reg,value)	tx_info_update()
	set_channel(value)
switch_RX()	
	rx_plcp()
	rx_complete()
	noise_measurement()
tx_frame(type)	tx_frame(type)
switch_TX()	
set_timer(value)	set_timer(value, i)
set_bk()	set_bk()
freeze_bk()	
update_retry()	
	contention_params_update()
more_frag()	
prepare_header()	rx_frame == value
	report_TX_to_host()
	manage_rx_error()
	manage_tx_error()
	remove_frame()

Table 3: List of WMP actions to define contention-based MAC programs

The implementation of the *update_retry()* action and the setting up of the contention window are embedded into a single *contention_params_update()* to be invoked at the end of each transmission for updating (according to the DCF exponential backoff rules) the contention parameters as a function of the transmission outcome. Moreover, we add some error management functions for dealing with transmission and reception errors performing the necessary reset operations, and errors on the format of the enqueued packets (*remove_frame()*). Finally, a report action is included for notifying the outcome of each frame transmission to the FLAVIA Information Base, assuming that this type of events is always subscribed by the upper DCF operations.

3.2 Interactions between Lower and Upper MAC

Different MAC machines can be loaded on the WMP and enabled/disenabled according to some switching events programmed by the user (by means of the FLAVIA Control System interface, IAP4) or by upper services. This feature allows to immediately extend DCF for supporting PCF (or HCCA), by loading DCF and PCF state machines and by opportunistically triggering the state machine switching (e.g. starting PCF at each beacon reception and switching back to DCF after the reception of the CF-END packet and CF period expiration). The WMP exposes a set of primitives to the FLAVIA control system for uploading the state machines, specifying the activating conditions and enabling code switching.

The diagrams depicted from Figure 3 to Figure 6 summarize the message sequences for performing these operations.

Loading

A new state machine can be loaded by means of the *write(bytecode,i)* primitive, whose parameters are the machine bytecode representation and the program slot. The Wireless MAC Processor confirms the success (or not) of the loading operation, *confirm(ready_code, i)*, according to the state of the machine slot indicated by the write primitive.

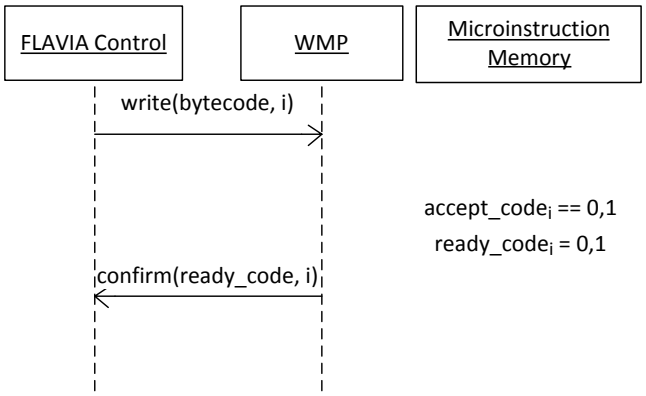


Figure 3: MSC loading operation

Starting

The MAC machine loaded on the i-th slot is started by means of the *run(i)* primitive, whose effect is updating the program pointer memory_slot and calling the *bootstrap(i)* primitive for the initialization of the machine. A confirmation message is sent back to the FLAVIA control system.

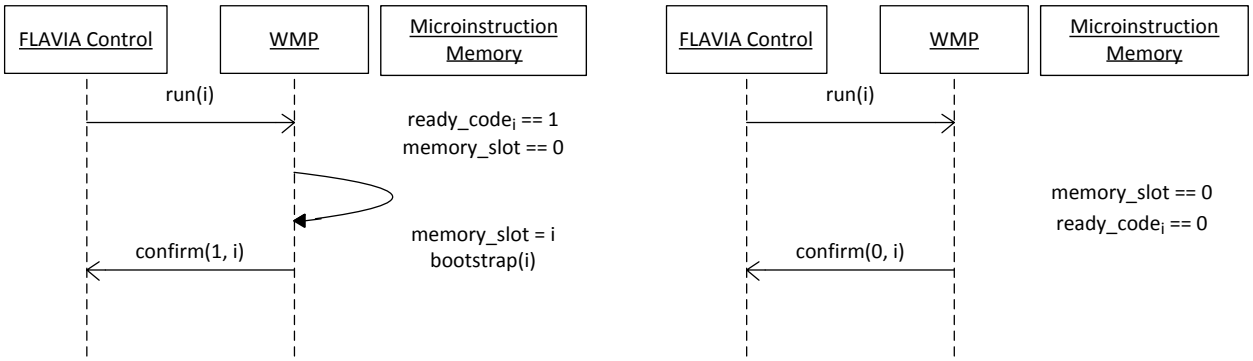


Figure 4: MSC starting operation

Switching

The switching operation is similar to the starting operation, but in this case the *run(i)* primitive triggers the update of the program pointer from a non-null condition (i.e., the pointer was containing a valid program slot). Alternatively, the switching signal can be generated internally by the machine under execution.

Synchronizing

The FLAVIA Control system can register an event for triggering a code switching by means of the *write_sync(event)* primitive, whose parameter represents the desired synchronization signal.

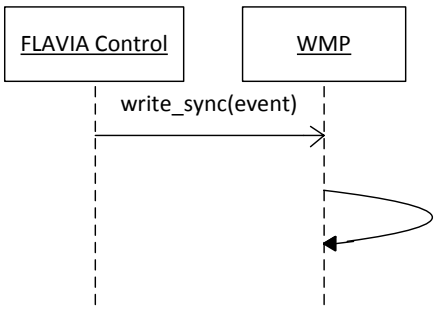


Figure 5: MSC synchronizing operation

Verifying

Optionally, the WMP can internally run the *verify()* primitive before accepting an incoming bytecode for recognizing trusted bytecode sources.

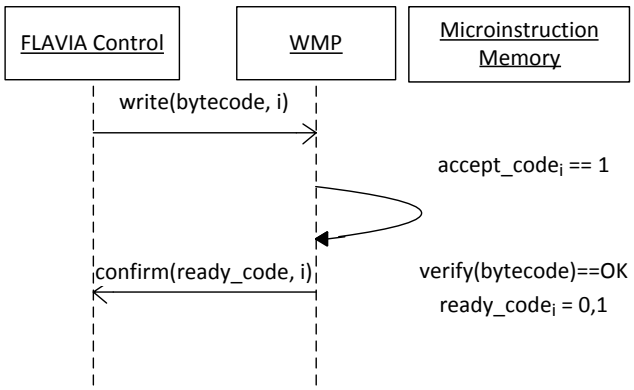


Figure 6: MSC verifying operation

4 802.11 Architecture: Service & Function Modules

In this section we describe two essential architectural containers that enable to develop and extend the MAC functionalities of the 802.11 node, the Service Scheduler and the Function Container. As well, we comment on the set of modules, compound of services and functions, which are developed within this project extending the modularity and flexibility of the 802.11 MAC.

4.1 Containers

Motivated by the necessity to provide an integrated middleware to easily develop new services and functionalities operating at the MAC layer, we design two components of the FLAVIA architecture that are liable for managing the scheduling of new services (*Service Scheduler*) and the registration of enhanced functions (*Function Container*) that permit to easily extend the basic functionalities provided by the 802.11 DCF. In the following sections, we describe the architecture of these two containers, detailing their static interfaces that permit accessing their services and the interactions with other entities that may occur during their execution.

4.1.1 Service Scheduler

The Service Scheduler is the architectural entity that allows the instantiation and the execution of new services, like monitoring, advanced data transport, misbehaviour detection and power saving. Specifically, this entity schedules the synchronous or asynchronous execution of the functions registered by any FLAVIA service during its initialization phase.

The registration and the execution of the main services procedures (i.e., the main entry point of the service's flow control) represent therefore the core functionalities provided by the Application Programming Interface of this component, since they enable the modular and flexible configuration of new services within the FLAVIA framework, focusing only on the development of the main service functionalities. Indeed, the Service Scheduler permits to divide the configuration and consistency control of the service from its execution, thus simplifying considerably their development.

Furthermore, this approach enables the maintenance and the optimization of any single component, which contributes to the implementation and execution of a FLAVIA service, removing the need of a costly redesign of the entire system for the support of more sophisticated functionalities.

As illustrated in Figure 7, the *Service Scheduler* is composed of three main components: the *Controller*, which provides the interfaces for the registration, the configuration and the consistency control of a new service, the *Container*, which

stores the services that have been registered for their utilization and composition, and the *Scheduler*, which implements all the functionalities necessary to schedule the execution of the services according to a particular policy (e.g., based on the priority, the estimated execution time, etc.).

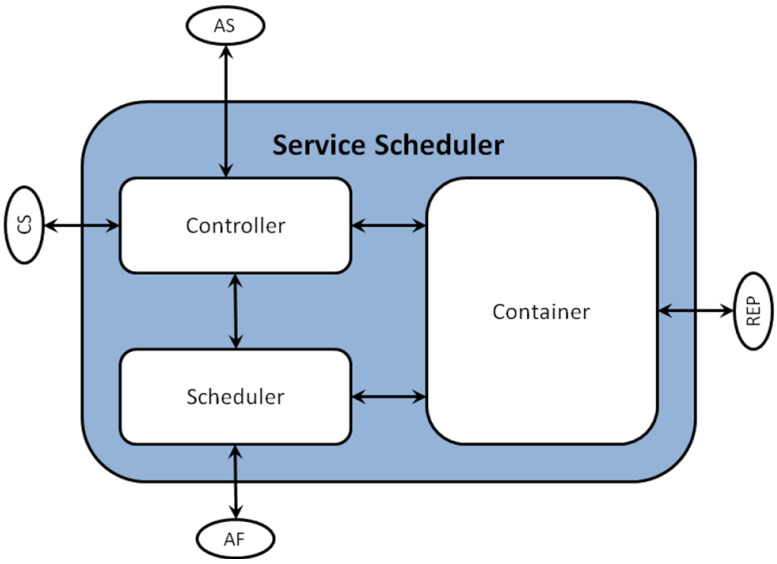


Figure 7: Service Scheduler architecture

Figure 8 illustrates the typical instantiation and execution of a new service within the FLAVIA architecture. The module implementing a FLAVIA service invokes the registration function provided by the Service Scheduler (i.e., *flavia_register_service_tsf_sync()* and *flavia_register_service()* called by services that require, or not, the synchronization with the Time Synchronization Function, respectively), passing as arguments the handler of the main service function that must be executed synchronously or asynchronously and a list of parameters used for the configuration of the service, e.g.: the scheduling period (interval between two consecutive executions of the service function), the events, whose occurrence cause the start and the termination of the service execution, the priority of the service, and the parameters used to set the internal configuration of the service.

Upon the registration of the service function, the *Controller* invokes the Consistency Manager to verify the correct configuration of the service (correctness of the parameters provided during the registration) and of the entire system (existence of the services and functions necessary to execute the loaded service). In case of a successful registration, the Controller replies with a positive acknowledgment to the module that is loading the service. Then, this module invokes the execution of the main service function (i.e., the entry point of the service defined by the service designer), according to the scheduling policy provided as argument at the registration.

Finally, at the occurrence of the termination condition configured at the registration, the Scheduler stops the execution of the main service function and informs the calling

module of the service termination, in order to perform all the operations necessary to correctly terminate/unload the service (flush out the memory, reconfiguration of the hardware, termination of the scheduled transmissions, etc.). Note that the termination phase might never occur.

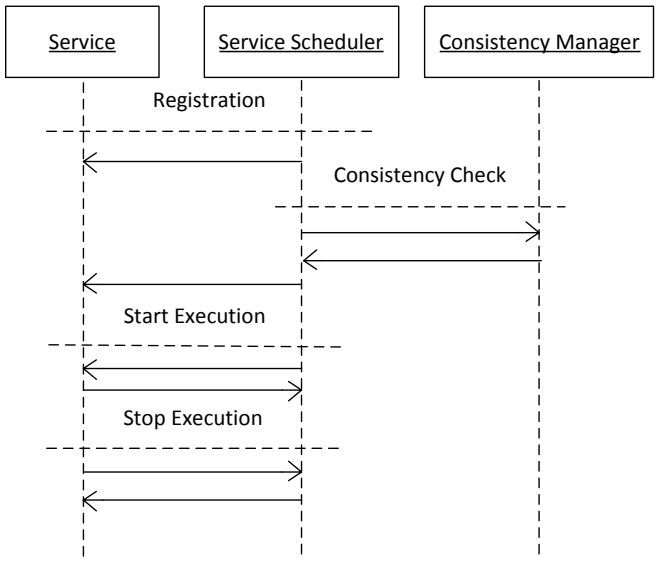


Figure 8: Sequence diagram of a service instantiation through the Service Scheduler

The *Scheduler* component coordinates the execution of all services registered through the FLAVIA framework. More specifically, this element implements the data structures and procedures that handle the concurrent execution of the main service functions according to the policy configured at the registration. Indeed, the *Scheduler* guarantees that the main service function of synchronous services is called within the configured time constraints in a real-time fashion. Further, it prevents the occurrence of race conditions due to the concurrency among the services (e.g.: deadlocks, starvation, misconfiguration of the hardware resources), by coordinating the access to the set of available resources used by the services.

4.1.2 Function Container

The *Function Container* represents the entity of the FLAVIA architecture that handles the set of functional resources, which extends the basic functionalities of standard contention-based MAC protocols. Note that the functions use the set of commands defined and implemented by the underlying hardware. Therefore, the set of functions designed and implemented through the FLAVIA architecture must be based on the features supported by the hardware.

The *Function Container*, whose architecture is depicted in Figure 9, is composed of two main elements: the *Controller*, which provides the interfaces for the configuration of new functions and their registration on the occurrence of events generated by the

MAC protocol (implemented using the corresponding software hooks), and the *Container*, which stores all the functions for their utilization and composition in order to implement enhanced services.

In addition, the *Controller* implements the procedures required to execute the functions at the occurrence of the corresponding events.

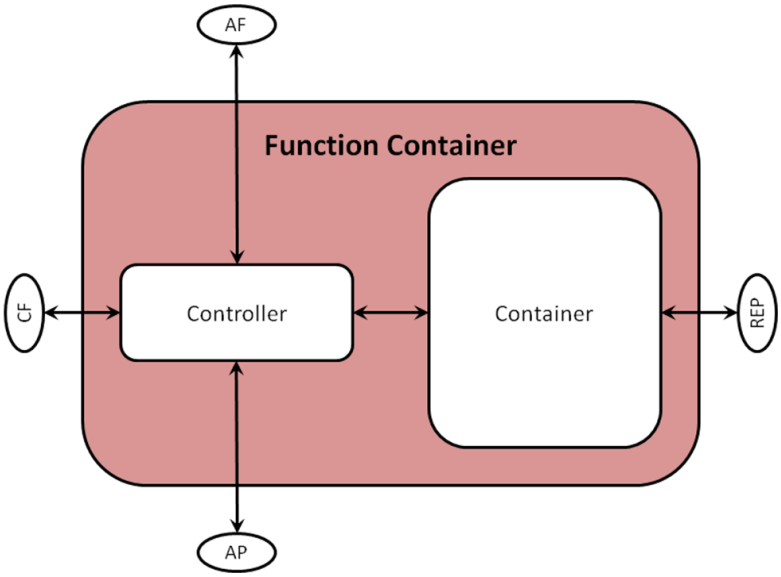


Figure 9: Function Container architecture

Table 4 summarizes the list of events that occur during the execution of the 802.11 MAC protocol and the corresponding FLAVIA hooks on which can be registered new functions to extend the functionalities of the basic MAC protocol.

Event	Hook	Description
Frame Reception	ieee80211_rx	This event occurs when a frame is received by the MAC protocol of a wireless interface.
Frame Queuing	ieee80211_tx	This event is raised after the frame has been created, just before to be sent to the queues of the underlying hardware.
BSS Association	ieee80211_associated	When an 802.11 entity (e.g. a STA) gets associated to a BSS, the BSS association event is generated.
BSS Disassociation	ieee80211_disassociated	When an 802.11 entity (e.g. a STA) receives a disassociation frame, the BSS disassociation event is generated.
Probe Request Reception	ieee80211_req_rx	This event occurs within the MAC protocol at the reception of an

		802.11 Probe Request.
Probe Request Transmission	ieee80211_req_tx	This event is generated after the creation of a probe request frame, just before its delivery to the driver for the successive transmission.
Probe Response Reception	ieee80211_res_rx	This event occurs within the MAC protocol at the reception of an 802.11 Probe Response.
Probe Response Transmission	ieee80211_res_tx	This event is generated after the creation of a probe response frame, just before its delivery to the driver for the successive transmission.
Beacon Reception	ieee80211_beacon_rx	The reception of a beacon frame triggers the Beacon Reception event.
Beacon Creation	ieee80211_beacon_set	After the creation of a beacon frame, just before its delivery to the driver for the successive transmission, the Beacon Creation event is raised.
Beacon Creation IE	ieee80211_beacon_set_ie	This event is generated when the MAC protocol begins the creation of the Information Elements, which are used to advertise auxiliary functionalities supported by the BSS

Table 4: Events generated by the 802.11 MAC protocol and their corresponding hooks

Upon the occurrence of an *event*, the *Controller* of the Function Container triggers the execution of all *functions* that have been registered on the corresponding *hook*, according to the execution policy and the function priority defined at the registration.

Illustrative Example

To better clarify the execution process of a function defined within the FLAVIA architecture, let us refer to the scenario depicted in Figure 10, which shows the sequence diagram of the tasks performed when a new frame is received by the MAC protocol. In this example, we assume that two FLAVIA services, Monitoring and SPS, have been loaded to extend the basic functionalities of the 802.11 DCF. Furthermore, we assume that the two services register the functions *parse()* and *stats()* on the occurrence of the events FRAME_RECEPTION and SPS_PROBE, respectively.

When a new frame is received, the MAC protocol triggers the FRAME_RECEPTION event that is captured by the Controller of the Function Container. The Controller inquires the Container to get the list of functions that have been registered on the corresponding hook (i.e., *ieee80211_rx()*). In the figure, the Container returns the identifier *Mon.parse()*, namely the frame parsing function registered by the Monitoring

service. Then, the Container executes the function *Mon.parse()* that performs the parsing of the received frame, detecting that is a probe packet generated by the SPS service. Therefore, the function *Mon.parse()* triggers the custom event SPS_PROBE on which the SPS service has previously registered the function *SPS.stats()*, which according to the information contained in the probe packet updates the information about the link quality. Note that the execution of *SPS.stats()* is performed by the Function Container after the occurrence of the SPS Probe event, similarly to the execution of *Mon.parse()*.

The events are represented with capital letters, whereas the invocation of the function registered on the corresponding event is denoted with the calls Mon.parse() and SPS.stats().

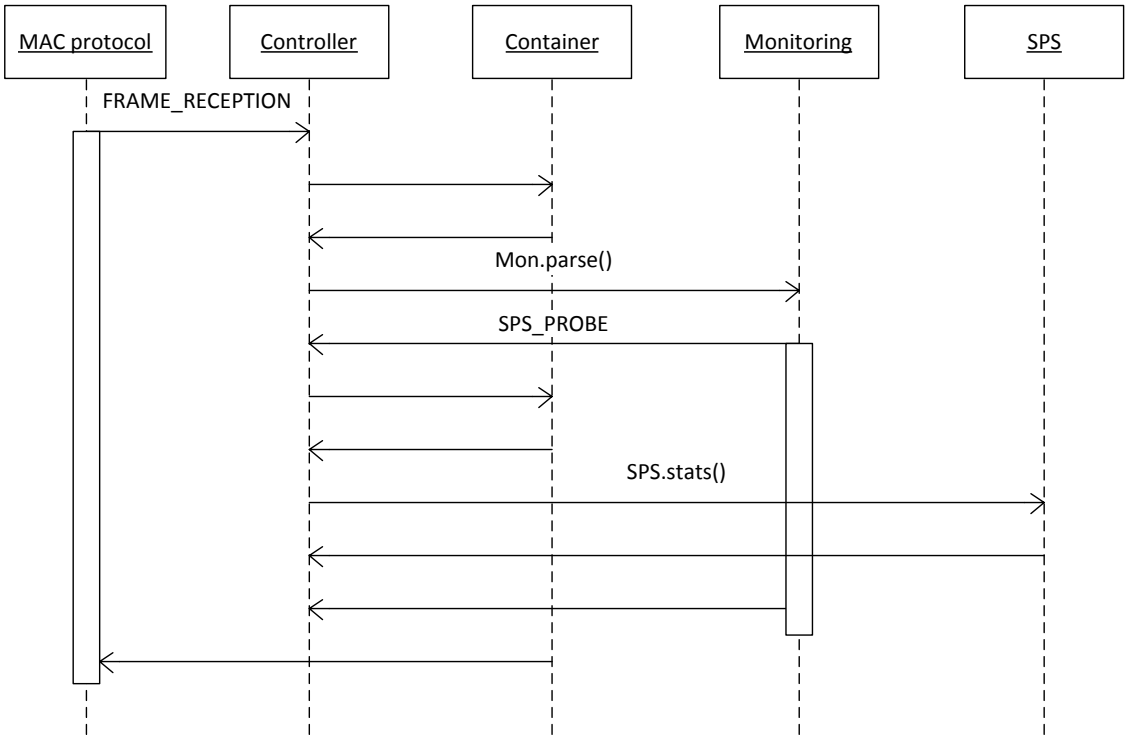


Figure 10: Tasks performed upon the occurrence of the event FRAME_RECEPTION

4.2 Services

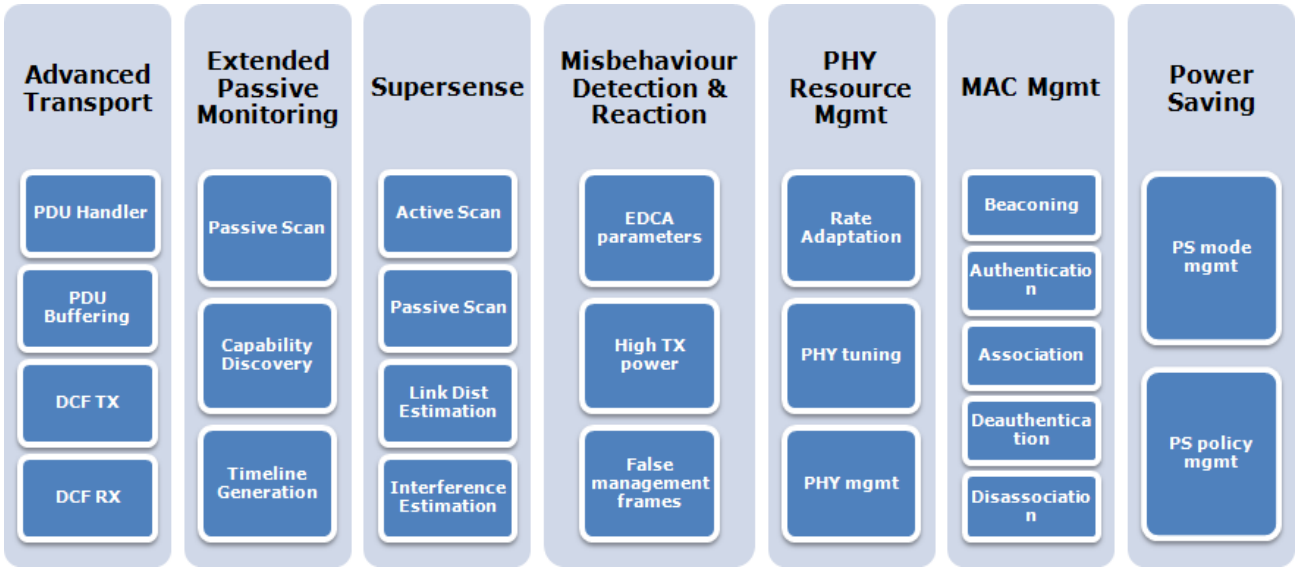
In this section we present the representative set of service modules, and their characteristic services, that an 802.11 node provides to achieve the basic functionality, as well as some new features inline with FLAVIA vision. Table 5 summarizes this set of modules with their corresponding acronym and a short description.

Module	Short name	Description
Advanced Transport	ATRAN	It provides extended operations for sending and receiving the MPDU across the driver and the wireless processor, and for managing multiple virtual queues.
Extended Passive Monitoring	MONI	It analyses the available radio spectrum, collecting information on the quality of each link, in order to choose the best available one for transmission.
PHY Resource Management	PHYR	It allows the configuration and query of basic PHY layer parameters, as well as enabling the rate adaptation service.
SuperSense	SPS	SPS coordinates active and passive monitoring activities among several devices, in order to minimize the interference and select the best network configuration.
Misbehaviour Detection and Reaction	MDR	The MDR module detects and handles the misconfiguration of 802.11 parameters, avoiding possible selfish behaviours of the nodes.
MAC Management	MGMT	It performs the basic set of management operations, depending on the node operation, such as: beaconing, authentication and association.
Power Saving	PS	The power saving service enables the configuration of different power save modes and policies, according to user and application requirements, including tuning on/off the radio.

Table 5: FLAVIA 802.11 modules

The modules are composed of different services that implement, extend and improve certain functionality. At the same time, the services invoke functions, such as *frame_forging()* or *listen_channel()*, which send the appropriate commands to the hardware. Part of these services will interact directly with the upper layers, as in the case of advanced transport or power saving, whereas other services configure MAC/PHY and collect statistics. Additional services can be added for dealing with technology-specific capabilities of different platforms.

Figure 11 gives a more detailed perspective of the different modules proposed and the corresponding services envisioned for each of them. In the following, we describe the operation of the new modules and services, and of those which have experienced a substantial change with respect to the previous deliverable. For legacy modules, such as MAC Management, for the sake of clarity, the details can be found in the previous



deliverable 4.1.1 [2].

Figure 11: 802.11 service modules overview

4.2.1 Advanced Data Transport

Data transport with parameterized QoS service allows traffic with special MAC layer transmission requirements to be accommodated. For example, multimedia traffic, such as a VoIP or Video traffic, usually requires low and stable end-to-end delay and low packet loss probability. The service is provided both in single-hop and multi-hop networks. The FLAVIA framework allows to define different functionalities to be composed for the creating of a given data transport service (as illustrated in Appendix A for the definition of a multi-hop data transport service).

An important aspect to be considered for the data transport definition is the packet classification and queuing process. For common contention-based systems we follow the queue scheme depicted in Figure 12. All the packets have to be ultimately transmitted in the air according to the MAC rules. The MAC transmission rate (which depends on many factors including network load, channel quality, retransmissions, etc.), corresponds to the rate at which the packets are drained in the so called "air-queue", i.e. the queue from which the MAC protocol takes the packets to be passed to the trans-receiver.

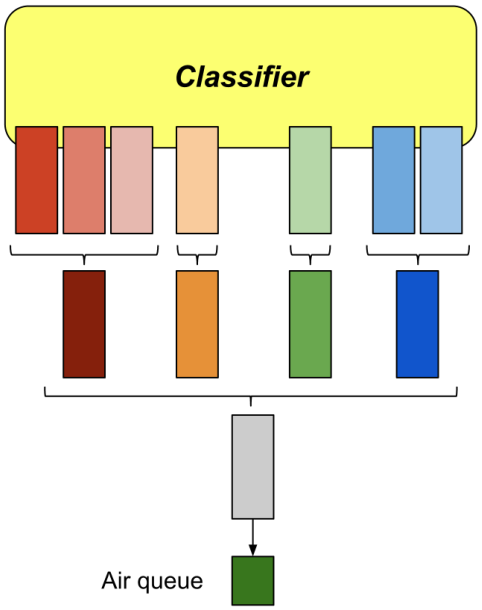


Figure 12: Queue scheme of a contention-based system

On top of the air queue, a hierarchical tree of queues may be defined for prioritizing a given type of traffic and controlling different traffic flows. In order to perform this packet sorting, a classifier must be introduced.

In FLAVIA, thanks to the multi-thread capability of the WMP, multiple air queues (generally linked to different MAC machines and corresponding to multiple logical wireless interfaces) can be available as depicted in Figure 13. A data transport service can therefore be defined by exposing N air queues to the upper layers. A straightforward exploitation of this capability is the definition of the EDCA queue structure of the IEEE 802.11e protocol [11]. Air queues can be used by independent MAC machines or by a multi-queue MAC machine as indicated in Figure 13.

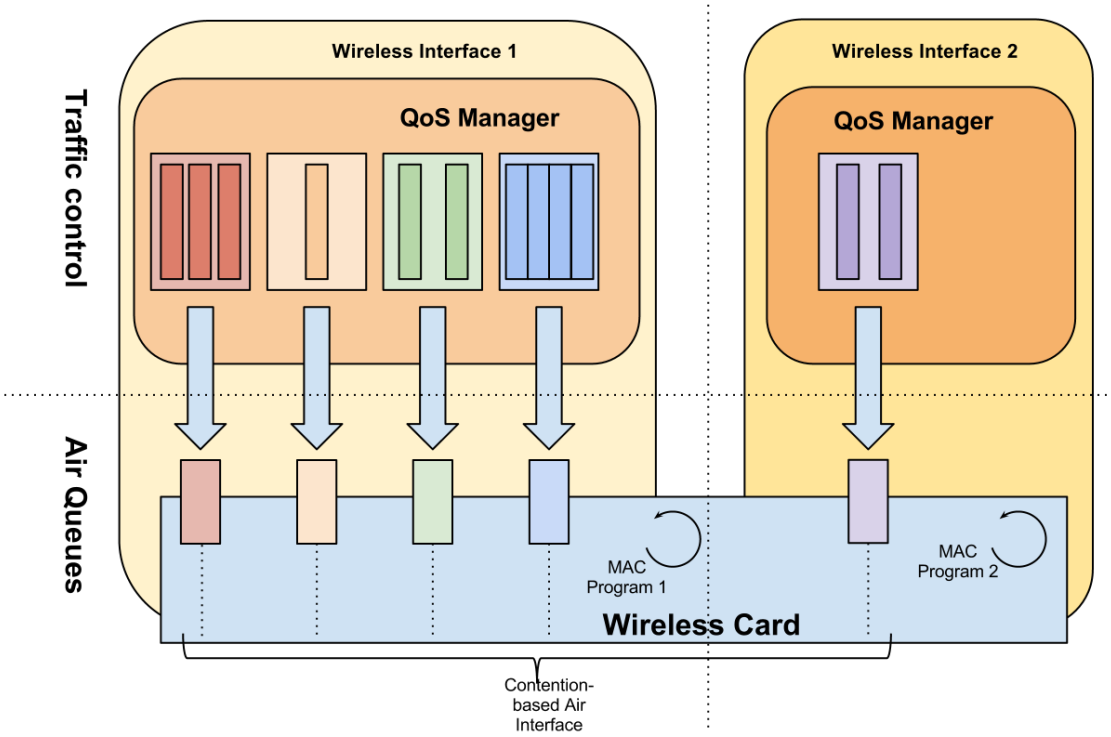


Figure 13: FLAVIA contention-based traffic architecture

Packet Scheduling Service

An important module of the data transport is the packet scheduling service, since it allows to define the queuing policy for each interface and to bind the upper queues to the physical air queues. This module provides functionalities for creating/destroying the queues, binding filters to the queues for accepting packets matching specific conditions, or linking the queues to the air queues.

The basic element of the scheduling service is the queue that stores the packets while they wait for being processed. Different kind of queues can be defined, exposing each of them the following set of functionalities: module initialization, packet enqueue, packet dequeue and packet drop. The diverse queues are organized in a tree hierarchy to perform traffic classification. Each queue has a unique identifier and can be inserted in the tree or removed from it. In order to manage the queues, the service allows tree traversing. Besides, storing a packet in a specific queue is achieved by means of the filters defined for each of the tree leafs. Filters can be attached to or removed from a queue, being also configurable the parameters of the filter.

Interaction with the air-queue

Each queue hierarchy is connected to the air queue by a direct connection. The air queue can control the rate at which packets are sent from the upper queues. To accomplish this, the queue tree must support the functionalities of stopping packet

dequeue or resuming it.

The direct connection between the tree and the air queue assures that: i) the air queue does not lose packets, relaying the control of dropping packets to the packet scheduling service; ii) the draining rate of the air queue is controlled by the specific air interface of the MAC.

4.2.2 Extended Passive Monitoring

The FLAVIA Monitoring (MONI) module provides a set of passive monitoring services able to measure several parameters related to radio channel conditions, capabilities of neighbouring nodes and also provide estimation of MAC 802.11 parameters based on measurements. Each node performs PHY/MAC layer measurements within the time-scale of microseconds, based on all types of 802.11 frames (data, management, and control). The passive measurements are performed along with the normal activity of the wireless card and reported periodically to the Information Base. The MONI module supports multiple network interfaces per node. The results of MONI measurements are utilized mainly by the following modules:

- Misbehaviour Detection and Reaction.
- Consistency Manager.

The results are made available to all FLAVIA framework modules and user space applications through the Information Base module.

The MONI module works on a frame level – this means that all frames sent and received by each network interface can be examined by the MONI module functions. This imposes high requirements on the MONI module on the effectiveness of the frame analysis (i.e., limited computational power available at the nodes should be taken into account).

The measurement functions require access to the header of each frame and to frame timing information, to discover the following parameters per each neighbouring station interface:

- Sender/receiver MAC address
- Operation mode
- Service Set Identifier (SSID)
- Channel
- Supported rates
- Frame type
- Preamble type
- Priority of received frame

- Frame length
- Timestamp of RX/TX frame
- Correctness of received frame
- RX power and RX noise
- Duration field

Based on the above listed parameters the following parameters are obtained:

- Number of active nodes in the neighbourhood
- Number of received frames
- Number of transmitted frames
- Frame Error Rate (FER)
- Bit Error Rate (BER)
- Per AC and overall uplink delay
- L1, L2, and L3 throughput
- Percentage of channel occupancy
- The approximate remaining L1/L2/L3 link capacity
- Number of retransmissions
- NAV
- Backoff
- IFS

The MONI component consists of following sub-modules, depicted in Figure 14:

- Passive scan.
- Capability discovery.
- Timeline generation to allow the extraction of EDCA parameters (cf. Section 4.2.3).

MONI implements the following interfaces:

- To the Service Scheduler, that is used to load and unload the service.
- To the Information Base, utilized to store monitoring results.

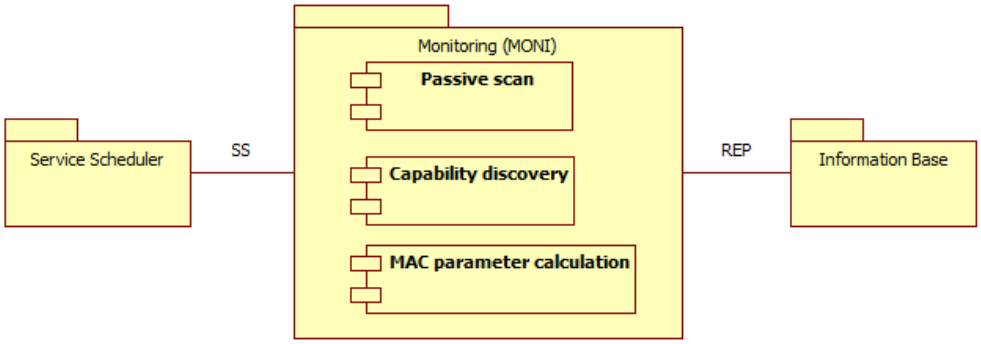


Figure 14: MONI module components and interfaces

The MONI modules are described in detail in D4.2 [3]. In Figure 15, the message exchange diagram corresponding to the MONI service operation is presented.

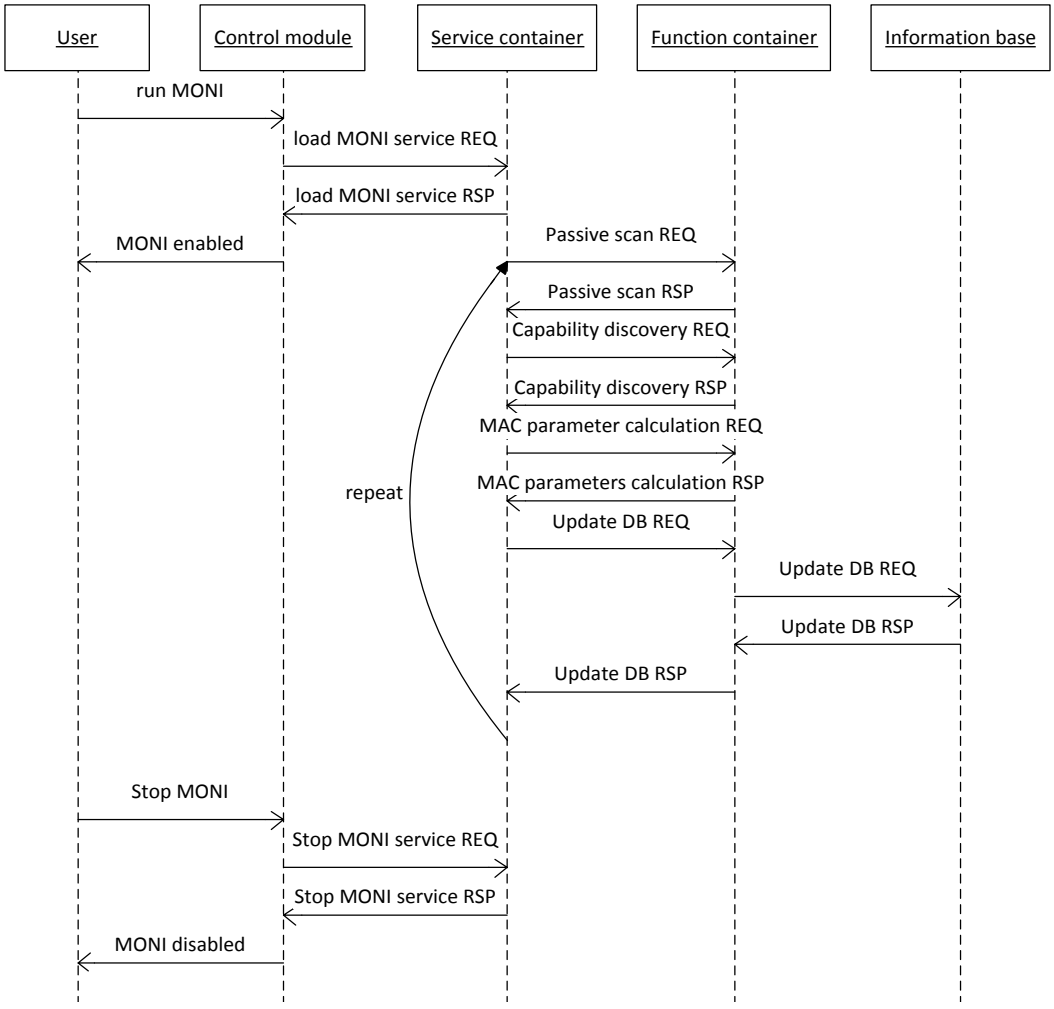


Figure 15: MONI module operation message sequence chart

4.2.3 Misbehaviour Detection and Reaction

The Misbehaviour Detection and Reaction (MDR) module is responsible for handling the misconfiguration of 802.11 parameters. As described in Annex A of D4.2 [3], its operation is based on network measurements obtained from the monitoring module (MONI) and stored in the Information Base (IB), as well as detailed timestamps obtained from the Wireless MAC Processor (WMP). Based on these measurements, it detects misbehaving nodes and applies methods to encourage such nodes to cooperate. These methods are then applied by configuring the medium access control function in WMP.

In this section we provide a revision of the MDR architecture proposed in Annex A of D4.2 [3]. First, we describe the overall module architecture, its interoperability with other modules including message sequence charts. Second, we describe the operation of MDR in four cases:

- Incorrect setting of EDCA parameters.
- Setting the TX power above the allowed limits.
- Sending false management frames.
- Reacting to misbehaviour.

MDR interoperates with two modules. In order to detect misbehaviour it obtains from IB measurements of the wireless channel made by MONI and detailed timestamp information from WMP. To appropriately react to misbehaviour it modifies the medium access function in WMP. Figure 16 presents a general overview of the data exchanged between the abovementioned modules (the arrows indicate the flow of data).

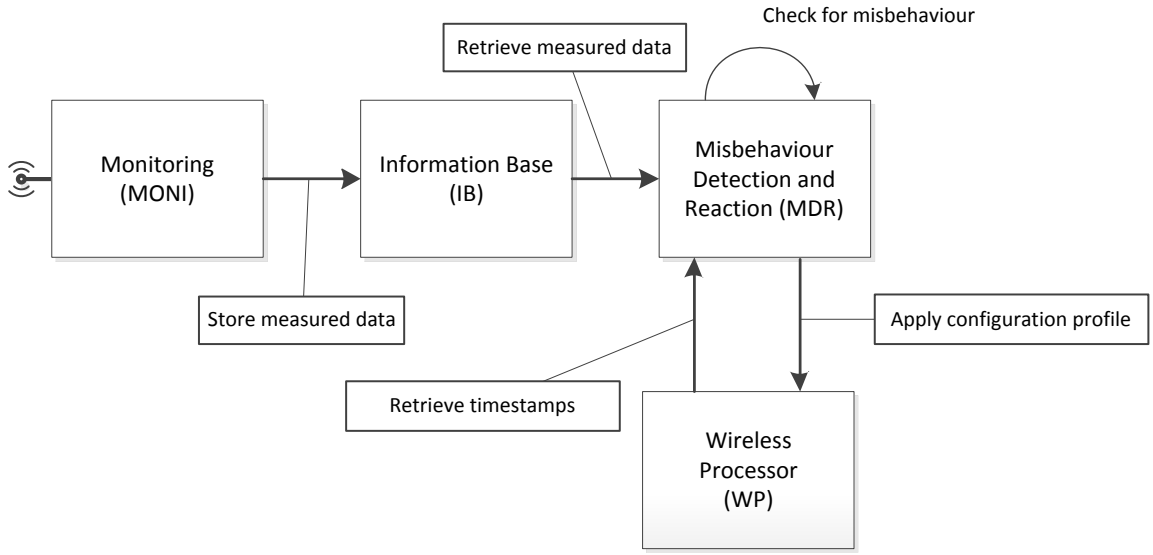


Figure 16: Overview of MDR operation

Figure 17 presents the message exchange diagram corresponding to the operation of the MDR service, while Figure 18 depicts the components and interfaces of the module.

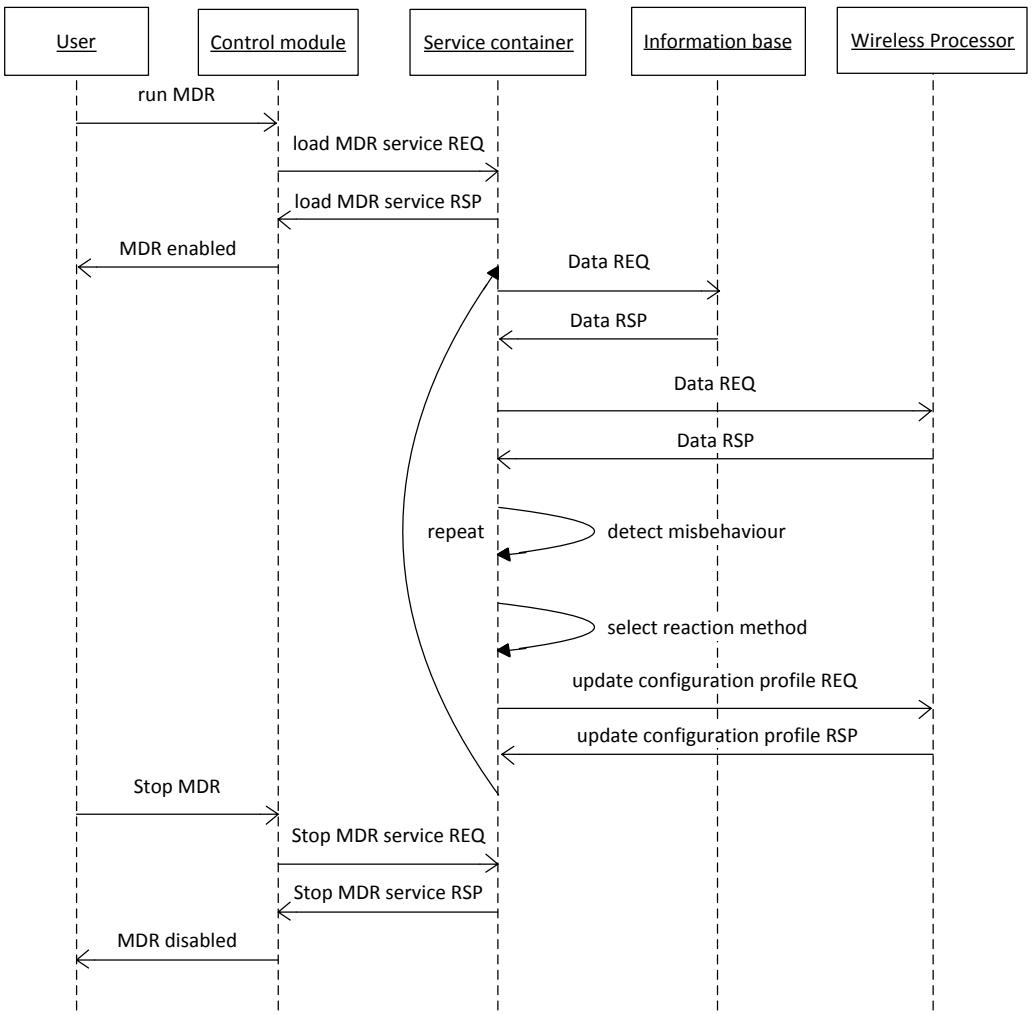


Figure 17: Message sequence chart of the MDR module

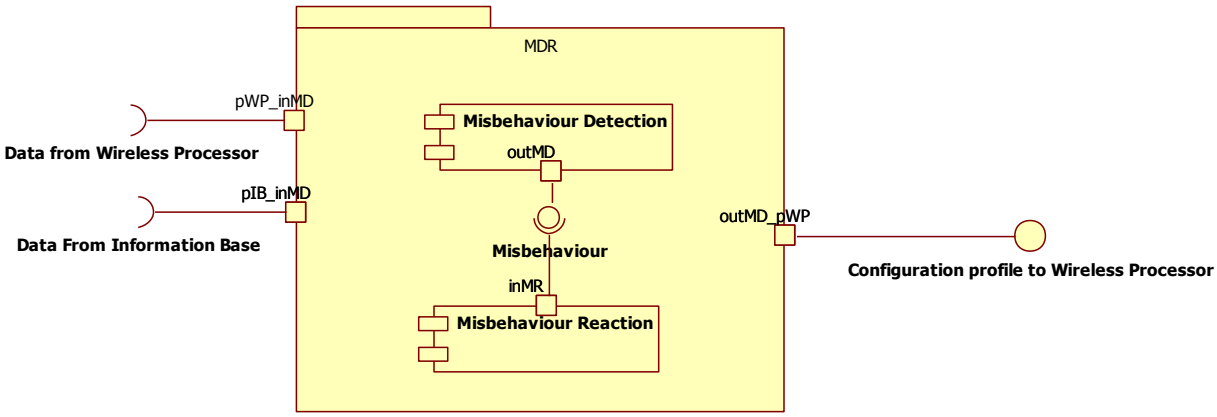


Figure 18: MDR module components and interfaces

EDCA parameter misbehaviour

Based on the data obtained from the IB and WMP, MDR is able to evaluate the IEEE 802.11 EDCA MAC parameters employed by an observed node. This is done by creating a detailed timeline of events and then extracting relevant information. The evaluation of certain MAC parameters (e.g., NAV and TXOPLimit) is a straightforward comparison with the standard values. However, the obtaining the values of AIFS, CWmin and CWmax require an analysis of the distribution of the inter-frame space between consecutive transmissions by a given stations, to allow us calculate these values (Figure 19). The correct setting of the CW values will be detected based on inter-frame space distribution with the use of any of the following methods: chi-square test, mean test, and entropy test. The number of employed CW detection methods can be extended. These methods have configurable parameters, which determine the number of false positives. The methods can also be configured to measure either actual or only consecutive backoff [14] as well as take into account all backoff values or only those for which the frames had their *retry* bit set to 0 [15].

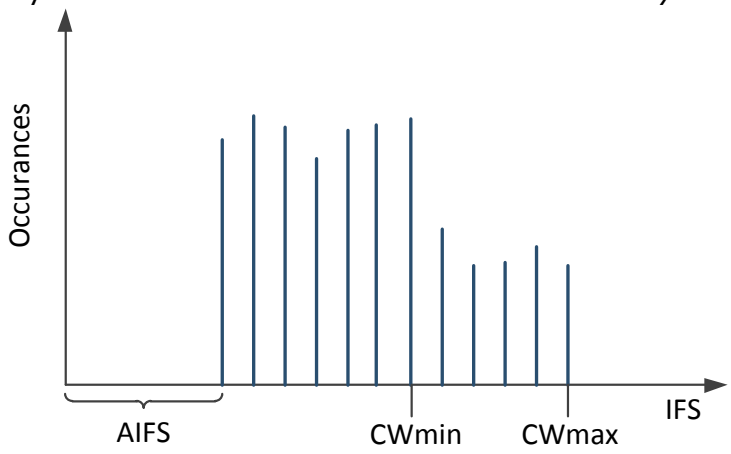


Figure 19: Distribution of IFS

High TX power

Not all the cases of operation with abnormally increased transmission power can be detected. In this implementation of the MDR module only cases when the increased TX power influences the performance of the network are considered. Such an approach excludes a set of problems where mobile stations are placed very far from the access point and use increased TX power and a highly directive antenna.

In this set of problems, the value of the EIRP (*Equivalent Isotropic Radiated Power*) can violate regulatory limits; still, from the network point of view, the station can exhibit no misbehaviour. Detection of the abnormally increased TX power is divided into two stages. In the first stage, the average RX power of the station under study is compared to maximum allowable TX power value, which is set according to the regulations applicable in the particular country. It is defined as a maximum RX power

value obtainable for a reference 10-meter long link consisting of transmitter operating with maximum allowable TX power and a reference receiver equipped with a half wave dipole antenna. The second stage covers less obvious cases where the average RX power stays within the limit of allowable values. This stage differs depending on station operation mode. When the station under study operates in single rate mode, the difference in the retransmission rate of frames exchanged between the local station and the station under study is examined. Otherwise, when the examined station uses a rate adaptation algorithm, misconfigured TX power is detected based on differences in data rate usage statistics recorded during communication from the local station to the station under study, and in the reverse direction. Additional measures may need to be taken into account if the two nodes use antennas with different gains or the channel is not perfectly symmetric (such as indoors). Furthermore, due to the nature of the detection algorithm, in most cases stations with misconfigured TX power can be detected only during active data exchange.

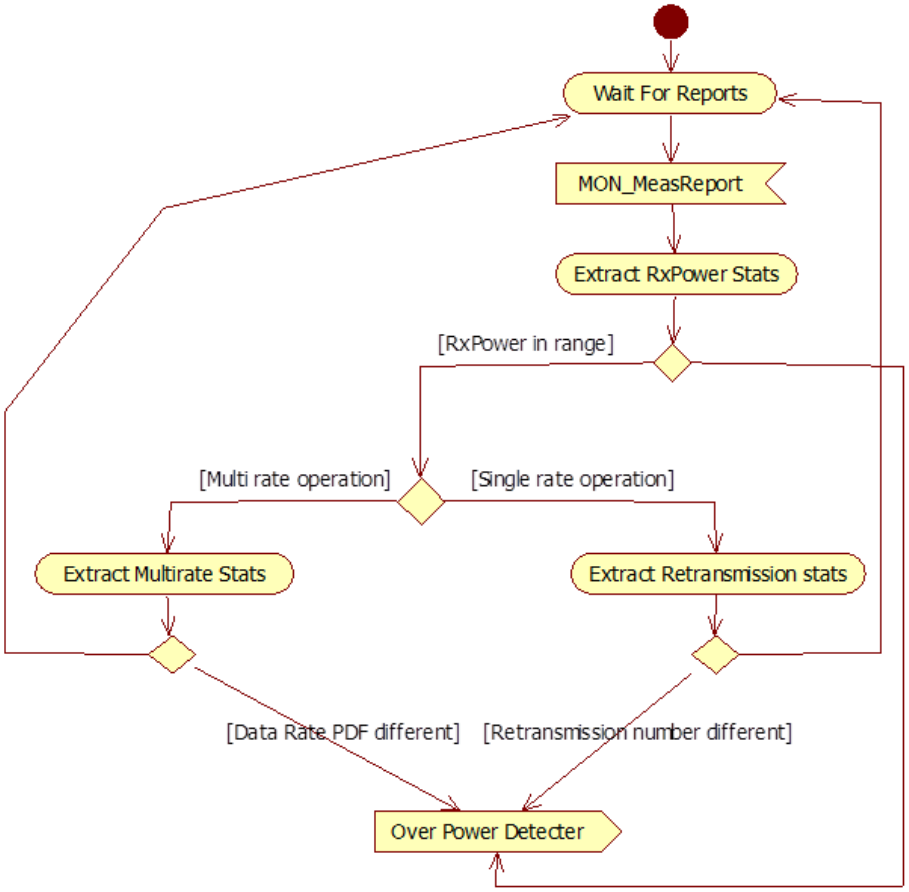


Figure 20: High TX power detection algorithm

False management frames

Detecting a high number (flood) of false management frames is based on a straightforward comparison of the number of their occurrences with predefined thresholds set for a given time frame. The following frame types will be checked for flooding attacks: RTS/CTS, beacon, authentication and deauthentication. Additionally, MAC spoofing may be detected by comparing the measured SNR value of consecutive frames transmitted from a given MAC with a predefined limit.

Reacting to misbehaviour

The reaction part of the MDR service will apply one of the following three methods to encourage correct behaviour: dropping acknowledgement (ACK) frames [16][17], selective frame jamming [18], and applying a game-theoretic strategy of adjusting the CW values to achieve an efficient operating point [19]. All these methods send appropriate configurations to the WMP to change its behaviour. The methods are applied when misbehaviour is detected and are suppressed when the misbehaviour ceases.

4.2.4 SuperSense

The virtualization and flexibility features of the FLAVIA architecture foster the development of SuperSense (SPS), an innovative monitoring service that dynamically analyses the available wireless spectrum using both passive and active techniques to estimate the best network configuration. SPS analyses continuously the available wireless channels to select the set of parameters that provides the best network performance.

The monitoring activity is performed concurrently to the data TX using two virtual interfaces operating over a single physical interface. The virtualization module is liable for scheduling the activities of the different virtual interfaces, representing the two operation modes, in order to fairly distribute the radio resources. In particular, the time spent for data transmission and active monitoring tasks is scheduled according to a time division mechanism implemented using a preemptive weighted round robin policy.

This module sets and manages the total duration of a SPS period and the specific length of the operation modes by introducing a new data structure, the super-frame. The duty-cycle of the super-frame, representing the alternation of transmission and monitoring phases along with the time assigned to each activity, is broadcast using a new Information Element (IE) contained in the beacon. As illustrated in Figure 21, the IE contains two main variables, namely m and e, which are used by all devices to compute the overall duration of the super-frame, T_{SPS}, and the time spent to perform the active monitoring, T_{MON}, according to the following equation:

$$T_{SPS} = s \cdot 2^{e}$$

$$T_{MON} = s \cdot m$$

ID	Length	e	m	Flags
1	1	1	1	1

Figure 21: SPS Information Element

Figure 22 illustrates the super-frame defined by the SPS service. Every super-frame always starts with an active monitoring period followed by a transmission period, in which all nodes that belong to the same BSS operate using the same medium access mechanisms (either CSMA/CA or TDMA) to transmit their data traffic. During an active monitoring frame, only one node is allowed to send probes on the wireless channel in order to estimate actively the quality of the wireless links established with nearby nodes and the interference that might be generated by external sources.

Note that the SPS service requires that all devices are capable of supporting its functionalities. To this end, SPS nodes communicate the service activation to the rest of nodes. Therefore, all the nodes should disable the SPS service, if they detect the presence of nodes that do not support SPS.

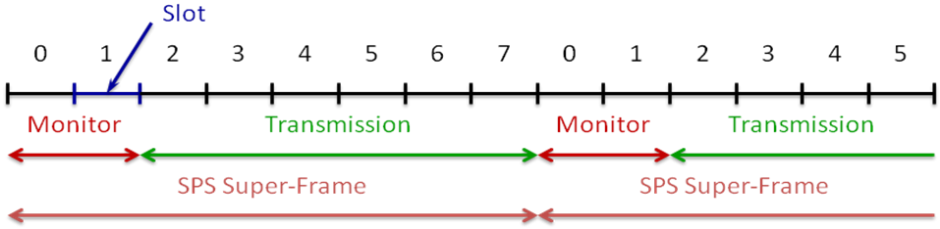


Figure 22: SPS super-frame

SPS collects several statistics used to assess the optimal network configuration or used by other services to optimize their internal configuration according to the channel conditions. For each available wireless channel and data rate transmission, SPS measures: (i) frame loss and delivery rate (also known as link quality); (ii) temporal and spatial frame reception correlations; (iii) expected frame delivery probability based on RSSI measurements.

The passive monitoring activity, explained in Section 4.2.2, is performed continuously and thus simultaneously to other activities like data transmission.

Figure 23 presents a comprehensive message exchange diagram corresponding to the SPS service operation, by indicating the messages and interfaces regarding the communication between the different modules. Besides, Figure 23 provides the reader

with a schematic view about how the service is loaded and set up and, within the framework, how the service is stopped and unsubscribed.

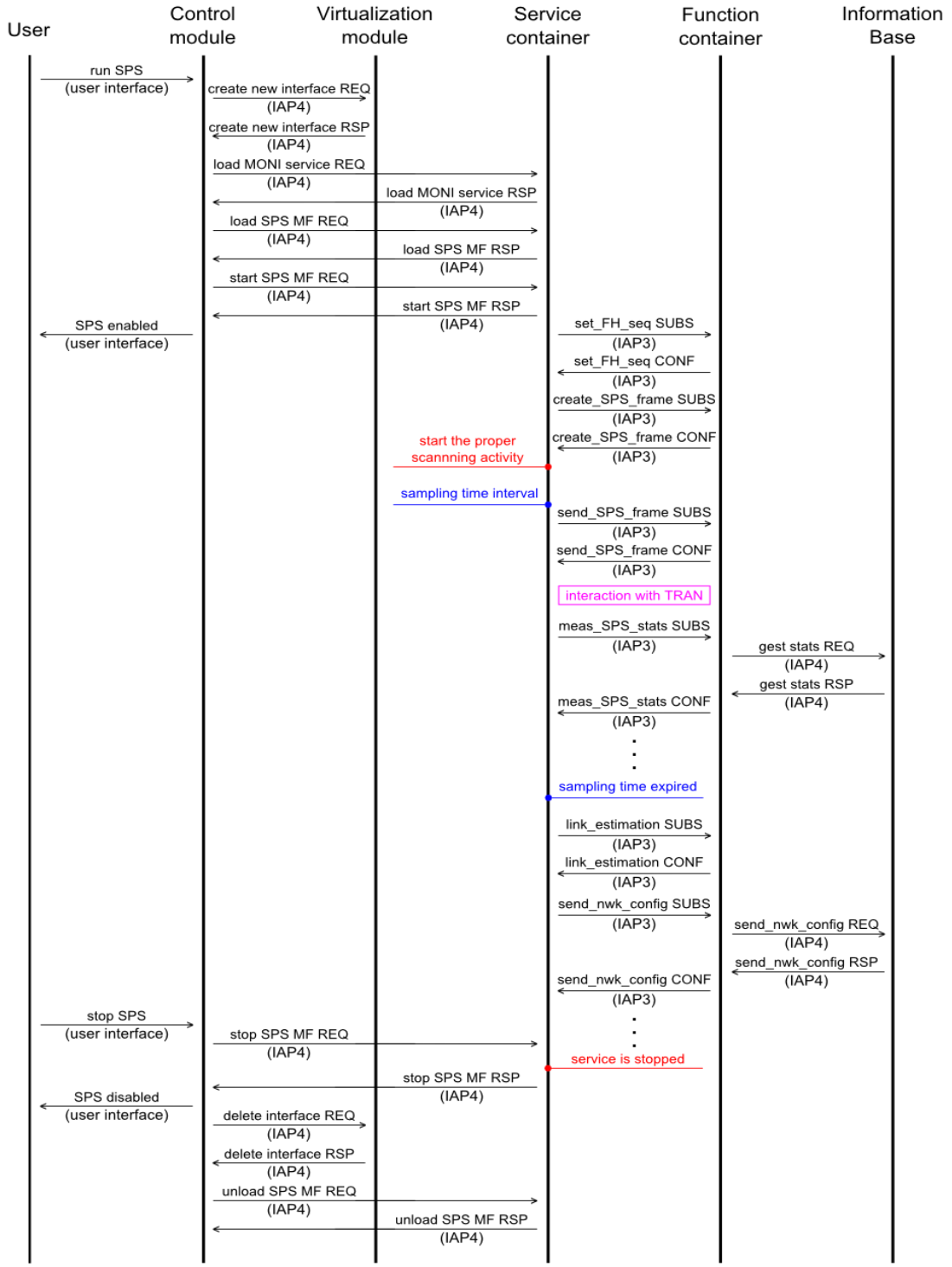


Figure 23: SPS message exchange

4.2.5 Power Saving

PS service module enables various power saving algorithms to be easily implemented through specifying helpful functional blocks and their interactions with other services, functions or the Information Base. The development of power saving mechanisms can also benefit from a modular and flexible architecture. FLAVIA offers a framework whereby tasks such as monitoring, frame forging, or sleep/awake transitions are exposed as online functional blocks that are provided with a common shared data space.

We envision two categories of potential PS mechanisms that can be implemented:

- Sleep/awake mechanisms: These mechanisms schedule intervals when a wireless interface dozes, being in a low-power state, and intervals when it becomes active by idling, transmitting or receiving.
- PHY/MAC adaptation mechanisms: These mechanisms adapt different PHY parameters (e.g., transmission power, modulation coding scheme), and/or MAC parameters (e.g., EDCA parameters, user association policies).

We propose a PS mechanism, named *NoA/ASPP*, a sleep/awake mechanism. Adaptive Single Presence Period (ASPP) [12] is a novel power saving algorithm to adaptively control the Notice of Absence (NoA) protocol specified by the WiFi Alliance. NoA has been proposed in order to provide energy savings to all devices (including AP-alike agents) in a WiFi-Direct network, a peer-to-peer wireless communication technology specified by the WiFi Alliance.

The wireless stack is the reference framework where to incorporate this PS module. The services that compose this module are liable for handling and loading the developed PS algorithms, respectively, PS mode management and PS mode policy.

Finally, our PS mechanism requires triggering sleep/awake events. This action is ultimately performed by hardware through setting the proper hardware registers accordingly. For that purpose, we specify a primitive to command the hardware to perform certain atomic and hardware-specific tasks, e.g., change its state to a low power state (sleep).

- *drv_flavia_ps_notify()*: This is a notification primitive and requires drivers to handle it. Thus, we push all the "intelligence" to the upper layer, designing this way a hardware-agnostic power saving framework.

In order to support sleep/awake transitions, typically required by power saving algorithms, it is still needed that drivers/firmware support sleep/awake events (issued by the previously mentioned primitives of the PS service).

4.2.6 Rate Adaptation

An important functionality FLAVIA architecture incorporates is the H-RCA rate adaptation mechanism [13]. H-RCA is another relevant example of how new functionality can be incorporated in the wireless stack thanks to the flexibility offered by FLAVIA, in order to enhance the performance of practical WLAN deployments. Rate control constitutes a fundamental building block of current devices that seeks to select the appropriate transmission rate, such that reliable communication is possible even under suboptimal propagation conditions.

The available rate adaptation solutions only rely on SNR and basic packet loss ratio (PLR) statistics, and often make inappropriate rate selection decision due to their inability to distinguish between channel error and collision induced losses. Thus, the solution adopted in FLAVIA relies on a packet-pair sampling technique to decide the most appropriate modulation scheme under current conditions, which is able to minimize the average MAC delay and provide higher and stable throughput.

The H-RCA operation consists of the following steps:

- The supported rates set is retrieved from the Information Base and sorted in increasing order.
- Rates r_i for which the PLR in given channel conditions is higher than the PLR for a higher rate r_j are identified and these rates are excluded from the rate-set.
- Data transport service is requested to configure the transmission opportunity (TXOP) parameter to permit the observation of packets solely susceptible to loss through channel noise and distinguish transmission failures that occur due to collisions.
- For each rate, compute a critical PLR value, the rate-lowering threshold, above which a lower rate would give higher throughput.
- Employ Bayesian inference on transmission statistics passed by the data transport service to determine if the PLR of the current rate is above a rate-lowering threshold.
- Set the rate increase frequency such that the opportunity-cost of sampling a higher rate is kept below 5%.
- Increase the rate when N successful transmissions not necessarily consecutive are observed at the current rate, being N larger than the successful transmission threshold (STh).

Table 6 summarizes the TXOP and STh parameters for the 802.11a PHY corresponding to each of the rates in the available set, while Figure 24 overviews the H-RCA operation.

Rate (Mbps)	TXOP (for 1000 B packets)	STh
6	0.0030s	361
12	0.0016s	589
18	0.0011s	779
24	0.0009s	893
36	0.0007s	1140
48	0.0006s	1349
54	0.0005s	NA

Table 6: 802.11a TXOP Parametrization

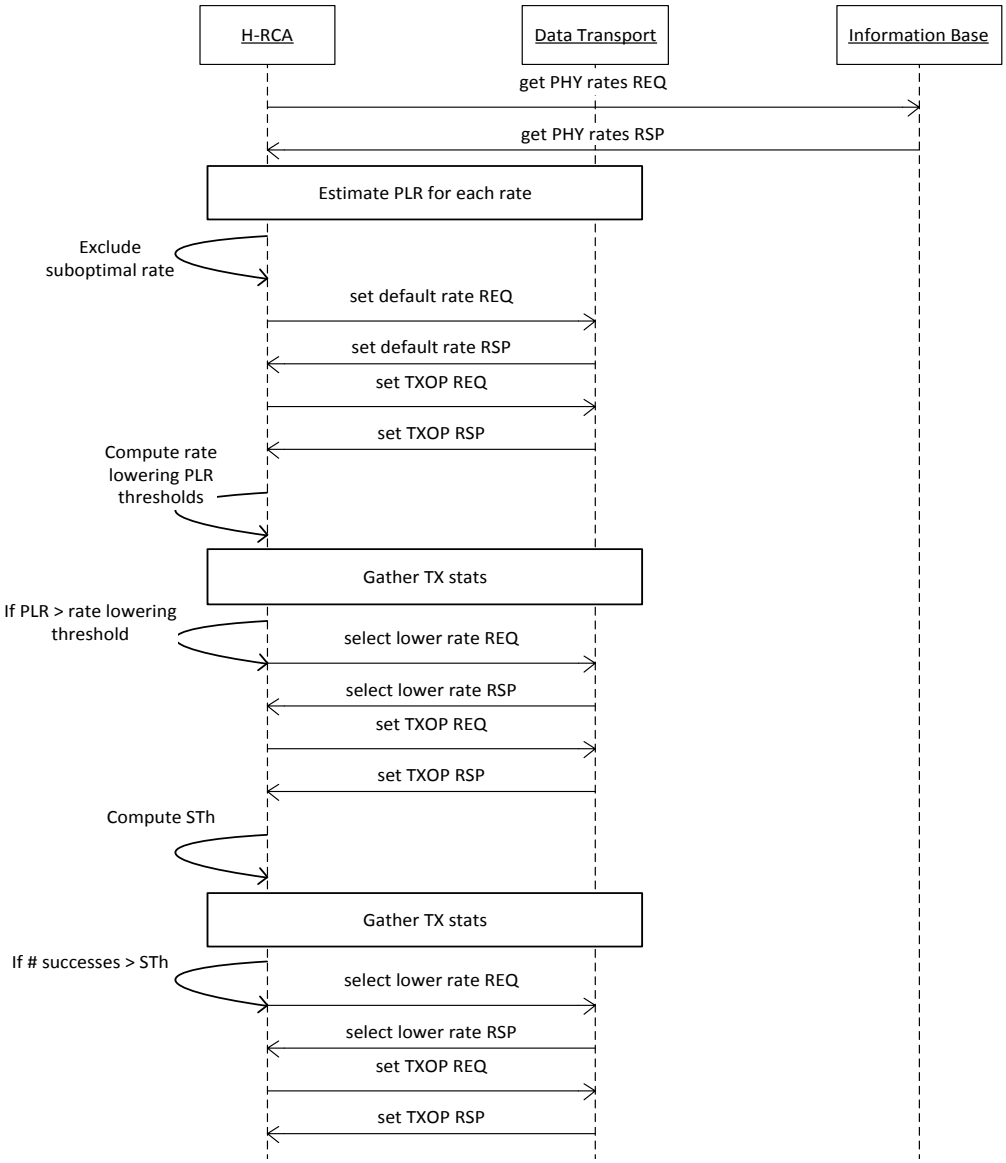


Figure 24: H-RCA operation

4.3 Functions

In this section Table 7 collects the set of functions utilized by the different 802.11 services and the description of their functionality.

Function	Description
flavia_service_scheduler_init()	It initializes the Service Scheduler creating a single thread workqueue to manage the services
flavia_service_scheduler_exit()	It unloads the service scheduler, deleting the auxiliary structures like the workqueue
flavia_register_service(service_handler, config_params)	It registers the service into the Service Scheduler
flavia_register_service_tsf_sync(service_handler, config_params)	It registers a service, which must be synchronized with the TSF, into the Service Scheduler
flavia_remove_service()	It stops the pending timers and deletes the corresponding work from the workqueue
flavia_ss_timer_function()	It adds the work implementing flavia_service_hook_container on the workqueue used by the Service Scheduler to handle services
flavia_service_hook_container()	It invokes the function registered by the service and reschedules the timer that executes flavia_ss_timer_function
flavia_service_hook_container_tsf_sync()	Like the previous one, but requiring synchronization with the TSF module
flavia_function_handler_init()	It performs consistency checks and initializes the internal data structures to fulfil the management task
flavia_function_handler_exit()	It removes the functions registered on all hooks and deletes the structures to free their memory space
flavia_register_function()	It registers a new function on a hook in the Function Container
flavia_remove_function()	It deletes a function registered on a hook when a service is removed
flavia_function_hook_container()	It invokes all functions defined on a specific hook when triggered to be executed
frame_forging()	It forges the creation of a solicited frame
fetch_defer_params()	It loads the transmission parameters, i.e., SIFS, BO, …

defer()	It defers the transmission of a frame when sensing the medium busy incrementing the BO counter
check_frame()	It checks the type of the incoming frame
drop_frame()	It drops the frame if there is any error or the maximum retry limit was reached
PDU_enqueue(MPDU)	It adds to the transmission queue a new packet pending to be transmitted
PDU_dequeue()	It dequeues from the transmission queue a packet ready to be transmitted
PDU_schedule(MPDU)	It schedules the transmission of a packet
flavia_sps_init()	It initializes SuperSense module
flavia_sps_init_tsf_polling()	It starts polling
flavia_sps_exit_tsf_polling()	It ends polling
flavia_sps_set_superframe_ie()	It configures the Information Element to be included in Beacons that defines the SPS superframe
flavia_sps_start_tsf_polling()	It starts polling once the station is associated
flavia_sps_tsf_superframe()	It computes the length of the superframe, according to the Information Element in the Beacon frames
flavia_tx_to_mon_mode()	It toggles operation from TX/RX to active monitoring
flavia_enable_active_probe()	It enables active monitoring
flavia_mon_to_tx_mode()	It toggles operation from active monitoring to TX/RX
flavia_disable_active_probe()	It disables active monitoring
flavia_active_probe_work()	It enables active probing for a device to transmit during active monitoring phase
flavia_send_active_probe()	It transmits monitoring frames
flavia_sps_exit()	It removes the registered functions and unloads the service
flavia_fem_int()	It initializes the Extended Monitoring module
flavia_fem_exit()	It unloads the Extended Monitoring service, freeing resources
listen_channel(channel,channel_dwell time)	It senses the channel for a channel_dwell time, that is the amount of time spent on each channel
create_probe_frame(parameters)	It creates a probe request/response frame with the corresponding parameters

send_probe_frame(destination interface)	It is responsible for managing the transmission of a probe frame
set_freq_hopping_sequence(sequence)	It configures the sequence for the frequency hopping when scanning the different channels
set_network_config(channel, modulation, parameters)	It sets the configuration for a node when behaving as an AP
create_dist_estimation_frame(timestamp, id)	It creates an ad-hoc distance estimation frame
send_dist_estimation_frame(destination interface)	It sends an ad-hoc distance estimation frame
channel_quality_estimation(channel, channel stats)	It computes the estimation of the channel quality based on statistical data
measure_collision_stats(statistical metric,statistical data)	It estimates the collision of the channel given the statistical data
obtain_measurements()	It gets the measurements of the network being monitored
estimate_CW(MAC address)	It estimates the CW from the statistical data
compare_MAC_parameters_with_std(MAC address)	It compares the observed 802.11 MAC parameters with the standard ones
send_configuration()	It sends the proper configuration to a node that is misbehaving
get_phy_possible(param)	It queries the IB for supported range of TX powers
get_phy_param_change_timescale(param)	It queries the time-granularity of the hardware to change PHY parameters, or if per-packet
transmit(data, params, values)	It sets the parameters per-transmission
get_phy(param)	Get current/last used value of parameter
set_phy(param, value)	Set current/next used value of parameter
enable_autorate(type)	Start/stop an automatic rate adaptation scheme
disable_autorate()	Disable auto rate adaptation
create_mgmt_frame(subtype)	It handles the transmission of a mgmt. frame according to the parameters
recv_mgmt_frame(frame)	It handles the reception of mgmt. frame performing the corresponding operation
create_beacon(SSID,interval)	It forges the transmission of a beacon announcing the specific SSID and the interval of transmission
check_fcs(frame)	It checks the FCS of an incoming frame

computeFCS(MAC_header, frame_body)	It computes the FCS for a frame to be transmitted
ps_policy()	It specifies the power saving policies, including beacon reception period and scheduling event trigger, and stores the context of the running mechanism
ps_management()	It provides management logic to support the PS mechanisms being implemented
ps_frame_queue(AID, frame)	It enables the AP to buffer the frames for STAs in the PS mode. If the queue is full it discards the frame
ps_frame_dequeue(AID)	It removes the frame from the AP buffer into the transmission queue upon the reception of a triggering message
create_PS_frame(subtype)	It creates a PS-type frame to trigger the send of buffered frames in the AP
check_STA_PS_status(AID)	Used by an AP station to check the status of the rest of the stations
event_trigger_scheduling()	It triggers the sending of a PS-triggering frame

Table 7: FLAVIA 802.11 functions

5 802.11 Architecture: Control & Management

The Control and Management modules, envisioned for an 802.11 node, are essential for the correct operation of the FLAVIA architecture. On one side, the Consistency Manager is the control entity liable for assuring the consistency and correct access to the information stored within the Information Base module, which can be accessed by the different operational modules. On the other side, the Virtualization module enables the creation of multiple virtual devices over a single PHY interface and manages the resources shared among these different virtual MAC entities that may be running. In what follows we detail the aforementioned modules.

5.1 Consistency Manager

The FLAVIA Consistency Manager (CM) is responsible for intra- and inter-node configuration and parameter detection. In addition, the analysis and resolution of potential or existing configuration inconsistencies are ones of its major functionalities. From the architecture point of view, the CM module is divided into two basic components: Intra-CM and Inter-CM, as shown in the Figure 25 presenting the component diagram of the CM. The CM basically relies on data stored in the Information Base. The most important data for the CM are recorded in the Discovered Capabilities Data Base. It offers two basic interfaces: Internal Consistency Check and Remote Consistency for internal and external operational modes, respectively.

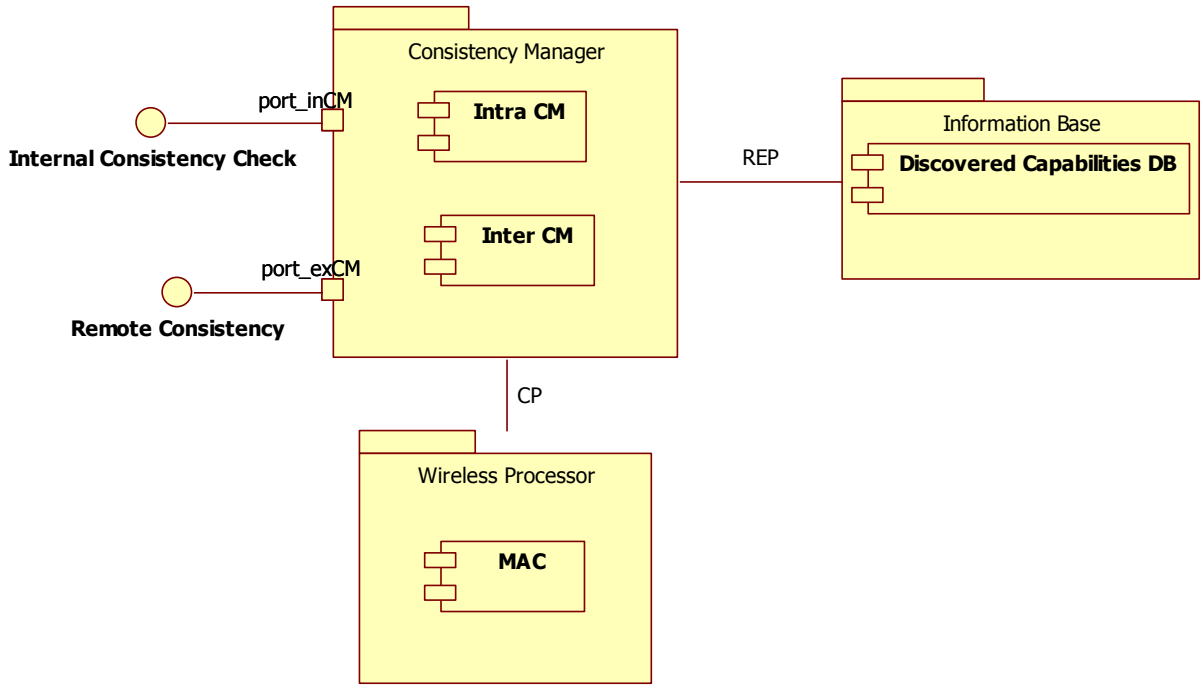


Figure 25: Component diagram of the CM

In what follows, we present the most typical use cases of the Consistency Manager:

1. Parameter Configuration Change.
2. Service Consistency Check.
3. Remote Consistency Change.

The Parameter Configuration Change use case covers the most common local action of the CM. At this point, the CM introduces an additional level of modules/services differentiation, which adds the following configurable features:

- Allow or deny a specified module/service to request a specific parameter value change.
- Assign priorities to modules to perform specific actions.
- Decide which request is allowed to take precedence over another.

The Intra-CM in case of detection of consistency violation can take the following actions: force the service generation violation to abort its request, return an error code or automatically enforce a correction based on the request. The Inter-CM component upon detection of remote inconsistency is capable of performing remote consistency change procedure, however remote node resolves the request accordingly to parameter configuration change scheme.

The Parameter Configuration Change use case is presented in Figure 26.

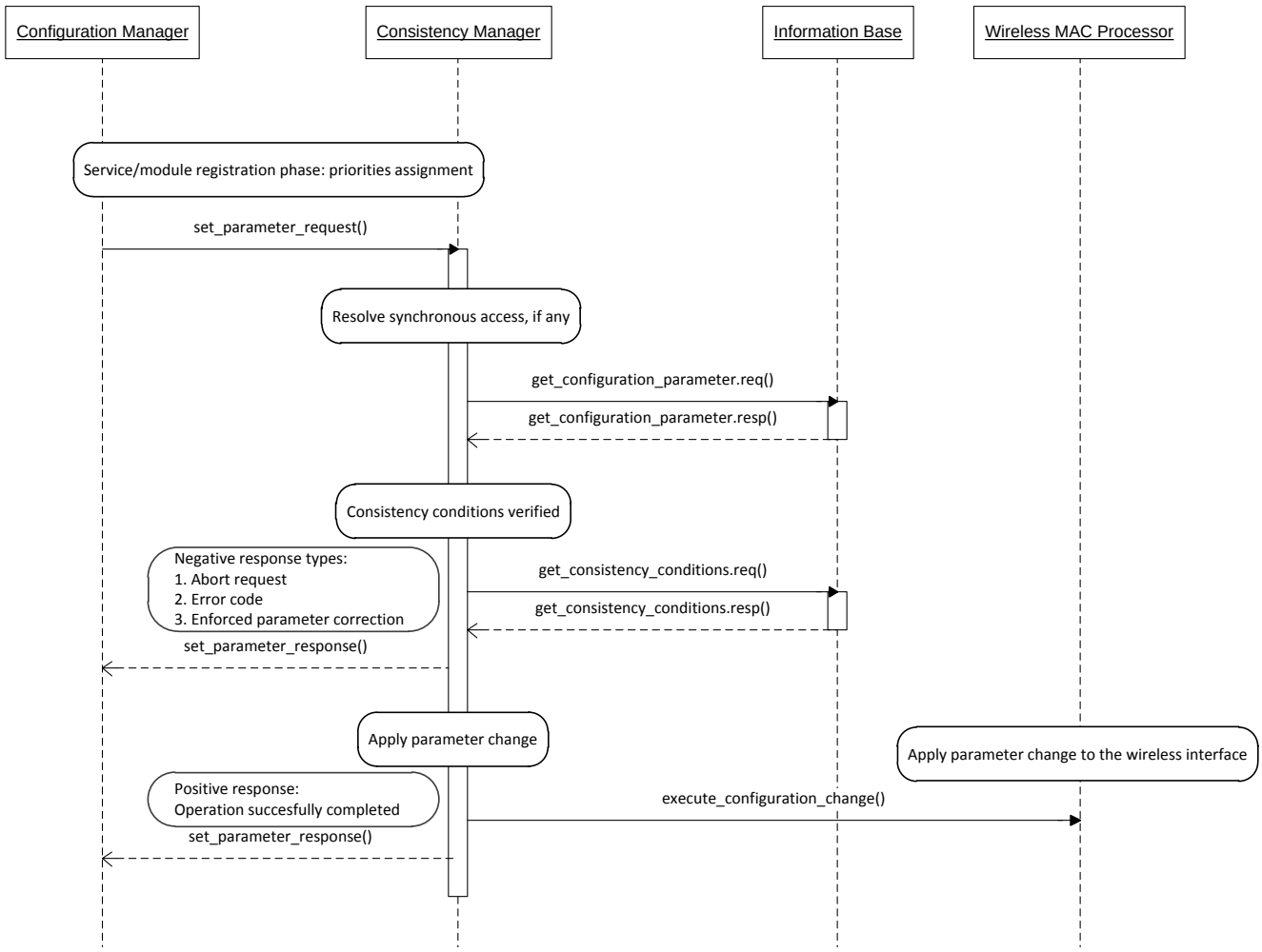


Figure 26: Parameter Configuration Change use case

The Service Consistency Check use case is triggered by the Service Scheduler in order to verify the correct configuration of the service (especially the verification of the correctness of the parameters provided during the registration) and of the entire system (checking the availability of the services and functions necessary to execute the loaded service). The Service Consistency Check, also considered as the Intra-CM operational mode, is depicted in Figure 27.

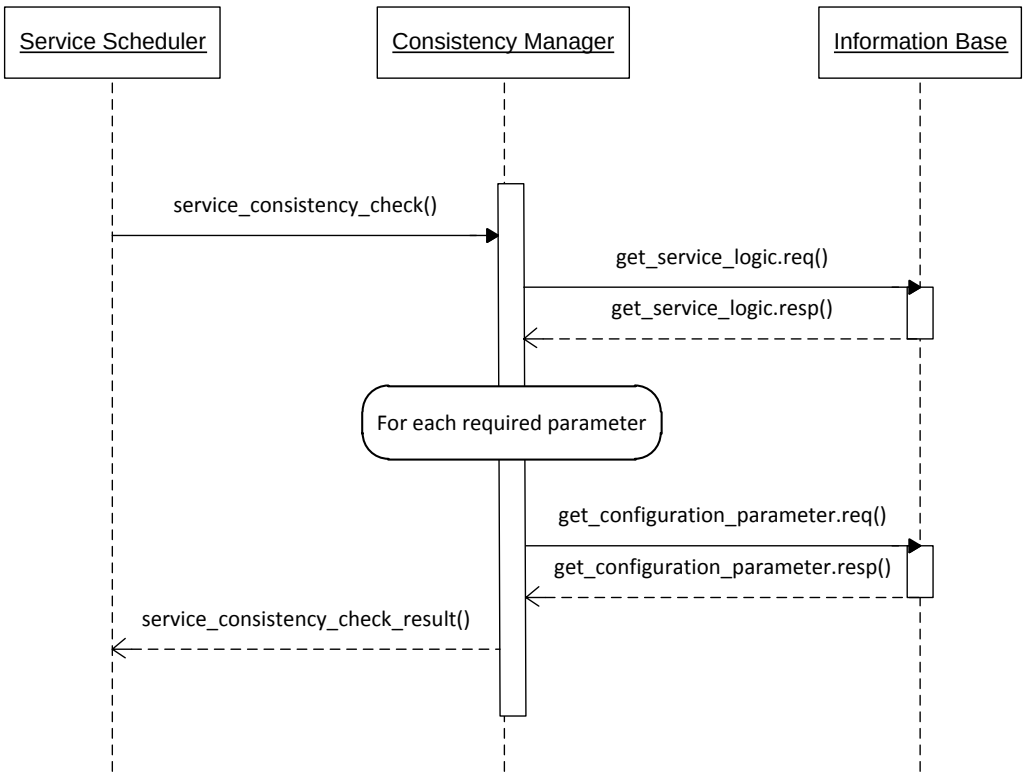


Figure 27: Service Consistency Check use case

This last use case of the CM covers the situation when a remote inconsistency is detected by means of Capabilities Discovery service of the Monitoring module. Consequently, the CM performs an operation of the remote parameter change. When a remote node receives the *set_remote_parameter.request,* it performs a similar procedure as in the case of the local Parameter Configuration Change. The Remote Parameter Change use case is shown in Figure 28.

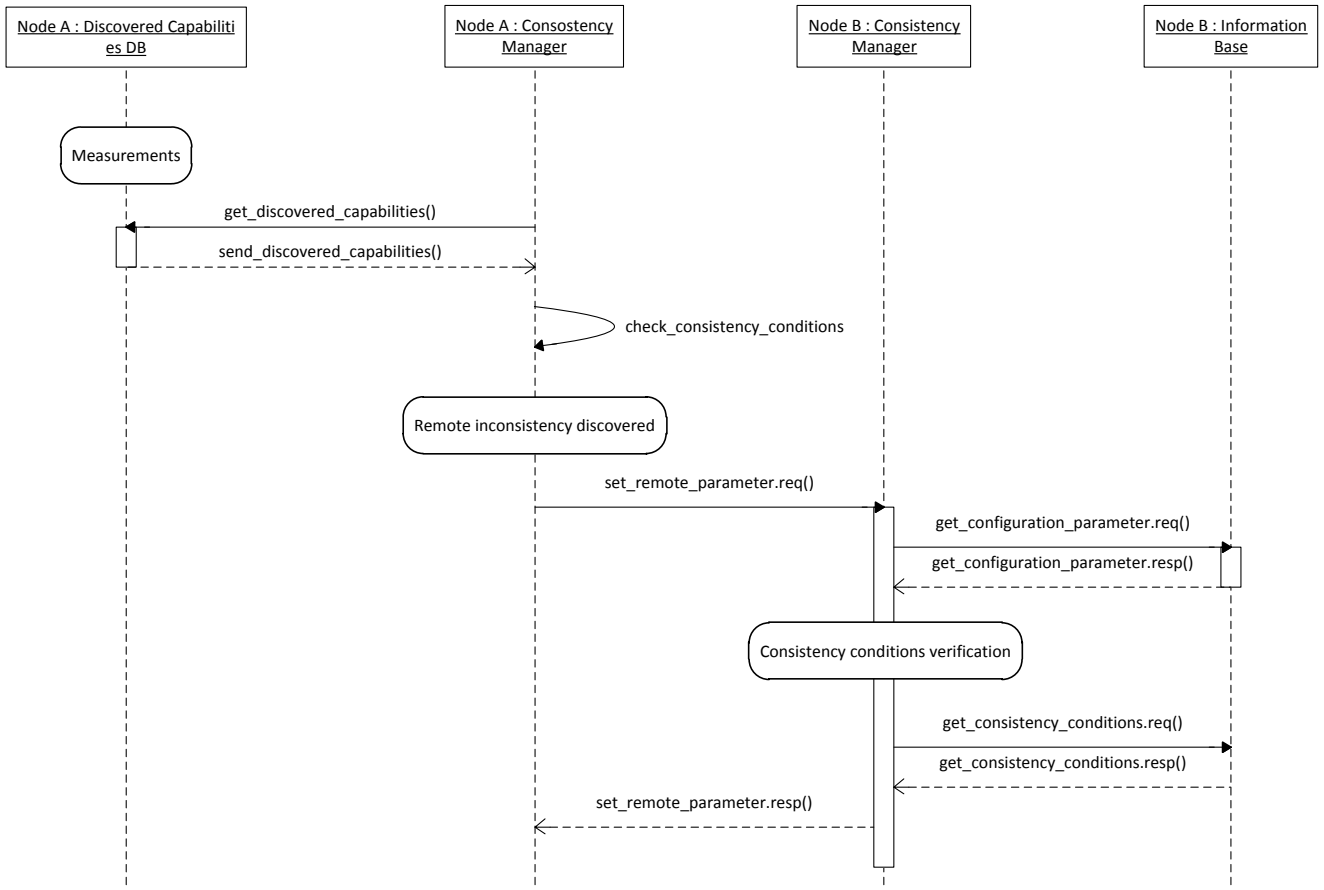


Figure 28: Remote Parameter Change use case

5.2 Information Base

The Information Base (IB) as defined for the overall FLAVIA architecture can be applied directly to the contention-based system architecture. In this section, first we briefly report the IB architecture as specified in D222. Second, we outline the information base support for multiple wireless interfaces and the memory management architecture.

The IB is divided into three subcomponents:

- The low level *data collector* that acts as a monitoring module interacting with the wireless processor, on which different data aggregation and filtering operations can be defined.
- The *data gateway* manages multiple accesses on the system state parameters and works in conjunction with the Consistency Manager.
- The *functional data manager* works on the service/function database in order to track and save the modules available in the system.

5.2.1 Data Collector

For this subcomponent, we envision exploiting the *Hardware Abstraction Interface* (HAI) for defining a minimum set of hardware parameters and signals that are stored as low-level data. Different polling or event-based data reading schemes can be defined, by taking into account the constraints imposed by the hardware features (such as the minimum polling time, the hardware measurement quantization, etc.).

5.2.2 Data Gateway

The data gateway enables the data sharing among FLAVIA modules and is responsible for managing the possible conflicts arose when multiple FLAVIA modules operate on the same system state data.

This data can be bound to a specific wireless interface (*wif*), as depicted in Figure 29, which can be real or virtual, or have a system-wide visibility to allow inter-interface storing. Thus, the data gateway acts on each data repository independently.

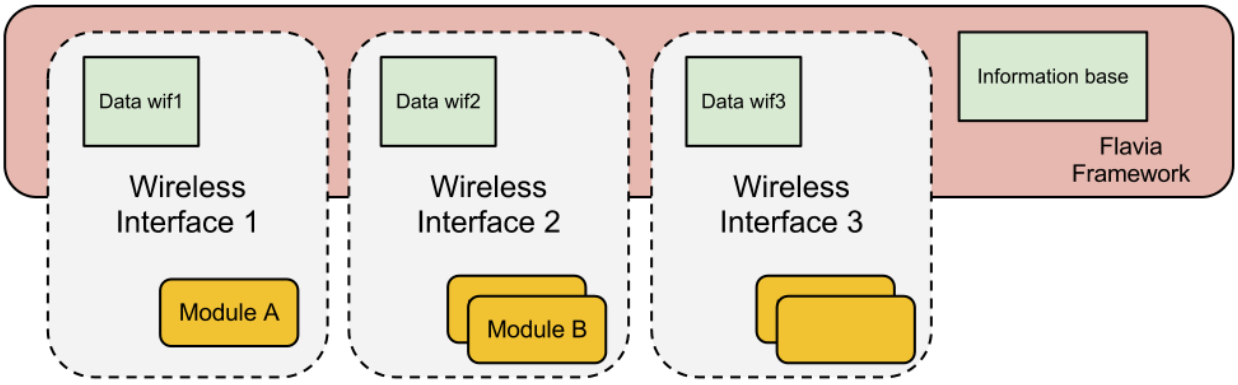


Figure 29: Data Gateway

The data gateway assumes that all the modules know all the data structures representing the system state. Moreover, each data field is mapped to a different Unique Identifier (UID), which is notified to all the operating modules.

The defined interface, *wif*, supports the following operations:

Reading operations

- *get_data(UID, wif)*: data - read a data value
- *on_change_listener(UID, change_listener, wif)*: outcome - register a change listener related to a given data field.

Single writer protection

- *create_data(UID, data value, wif)*: MID - initialize a data field.
- *change_data(MID, data value,wif)*: set a new value for the data field identified by the MID.
- *clear_data(MID, wif)*: outcome - remove a data field.

Multiple writers with challenge protection

- *set_value(UID, data value, wif)*: outcome - set a data field.
- *delete_data(UID, wif)*: outcome - remove a data field.
- *protect_data(UID, consistency_verification, wif)*: outcome register a consistency verification operation.

5.2.3 Functional Data Manager

The Functional Data Manager organizes and interrogates the database containing the information about the system functional resources. At this design stage, we envision storing a minimum set of fields for each functional resource:

- *Resource name*: the name of the service, function or command to be invocated;
- *Resource interface*: the list of basic or advanced data to be passed to the functional resource;
- *Resource state*: the resource availability state that indicates if the resource can be immediately invocated (running) or if it has to be loaded (unloaded), initialized (loaded), or started (initialized);
- *Resource dependencies*: the list of other resources called by the current one;
- *Resource advanced data*: the aggregation of basic data in data structures used by the current resource;
- *Resource consistency conditions*: the list of consistency tests related to the system state data affected by the current resource.

5.2.4 Memory management

A sensitive issue in the Information Base is the management of the memory. This section specifies how the IB deals with the stored values and how the modules must interact with the IB. Three operations are described to overcome with this issue: the *write*, the *read* and the *update* operations. All these operations define a clear distinction between the memory of the IB and the memory of the module. As a result, the delete process gets simplified and potential issues (such as, memory leaks or concurrency problems) are avoided.

The Figure 30 presents the *write* operation. A Module A passes the Value A to the IB using the set operation. The IB allocates memory to store that value and the related metadata. If the operation is completed with success a positive response will be returned.

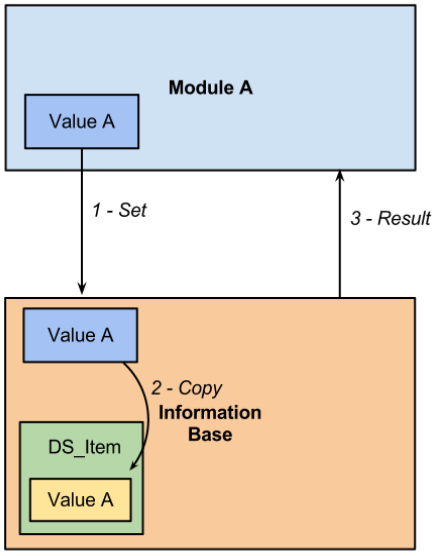


Figure 30: Write operation of a value

The *read* operation is depicted in Figure 31. The Module X needs to access a value previously stored, thus it allocates the correct memory space to store the value and inquire about it to the IB using the appropriate key. The IB copies the requested value, if available, in the allocated space and returns it to the requesting module.

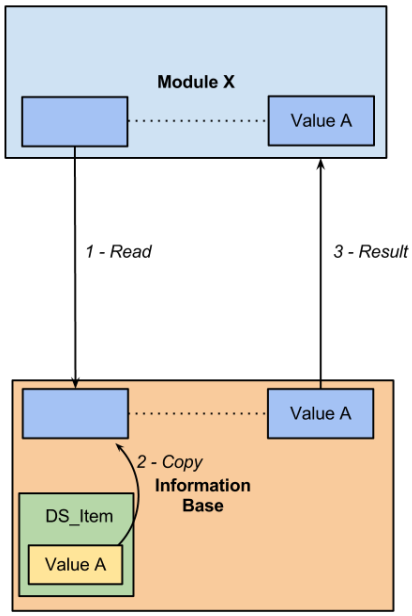


Figure 31: Read operation of a value

Figure 32 describes the *update* operation. Now, a module changes a stored value by providing it to the IB. If the memory required for the new value is equal to that used for the old the value, then the value is substituted. On the contrary, if the value exceeds the memory size, then the memory cell is released and a new allocation for the new value occurs.

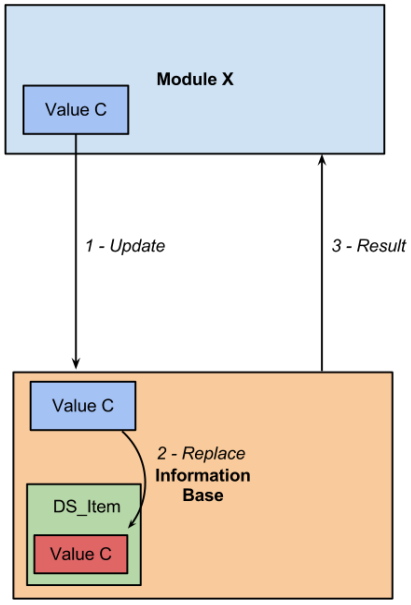


Figure 32: Update operation of a value

5.3 Virtualization

The virtualization module creates and manages the life cycle of the virtual devices over a single PHY interface. As previously explained in the D4.1.1 [2], a critical aspect of the virtualization is the actual sharing of the real physical resources based on different techniques. In the case of CDMA and FDMA, these modes are controlled by the MAC layer itself, in a close-to-hardware fashion; though, in case of TDMA scheme (applicable to the IEEE 802.11a/b/g/n technologies), the virtualization has a role in controlling the time slice allocation that maps virtual and logical devices on top of the physical hardware.

Meanwhile, each virtual device is usually managed through the usual and expected wireless API (typically, on Linux-based system, the mac80211 frameworks offers a range of standard API to control the behaviour of each IEEE802.11 device in terms of status, frequency). The standard wireless API offers logical device management (i.e., a single card could be used as Access Point and client at the same moment, on the same channel), but this should be not be confused with virtualization, where each virtual card can run on a different channel, using TDMA as sharing technology.

Therefore, the virtualization module needs a dedicated module, which defines the following set of functions:

- **Creation of virtual interfaces**, with optional range of parameters. In case of 802.11 and TDMA technology scheme, specification of the time slices, the type of constraints (min/max of time slice allowance) on them, and also if the constraint is hard or soft. This range of parameters allows flexibility of self-management, as the system is able to take decision on periodization based on this info.

  create_vif(name,[phy=physical_device],[duration],[min],[max],[hard|soft])

- **Destruction of virtual interfaces.** It enables proper destruction of the interface, equal to physical removal of a device using plug'n'play. This insures coherence of the entire system (no logical instance running on non-existing virtual interfaces for example).

  delete_vif(name)

- **Management of execution parameters** (time slices of the TDMA access for example), where some virtual interfaces can be weighted more than others according to external parameters, according to the very same set of parameters defined at the very first item of the present list.

  change_vif(name, duration],[min],[max],[hard|soft]).

- **Migration of virtual interfaces**: this functionality is designed to re-map existing virtual interfaces to existing or new interfaces, in order to insure redundancy, safety mechanisms. Important nodes can stay online. This can be used also when destroying a virtual interface.

  migrate_vif(name,newphy)

If the optional parameters are not given, the virtualization control module insures best effort operations (i.e., duration is based on the remaining free time, the physical interface is the default interface, soft constraints, no min or max share allocation).

The operation of this module is straightforward. After internal coherence control through the Consistency Manager, the commands are propagated through the different elements of the FLAVIA architecture through the various IAP previously defined.

The typical life cycle is defined with the message exchange diagram depicted in Figure 33.

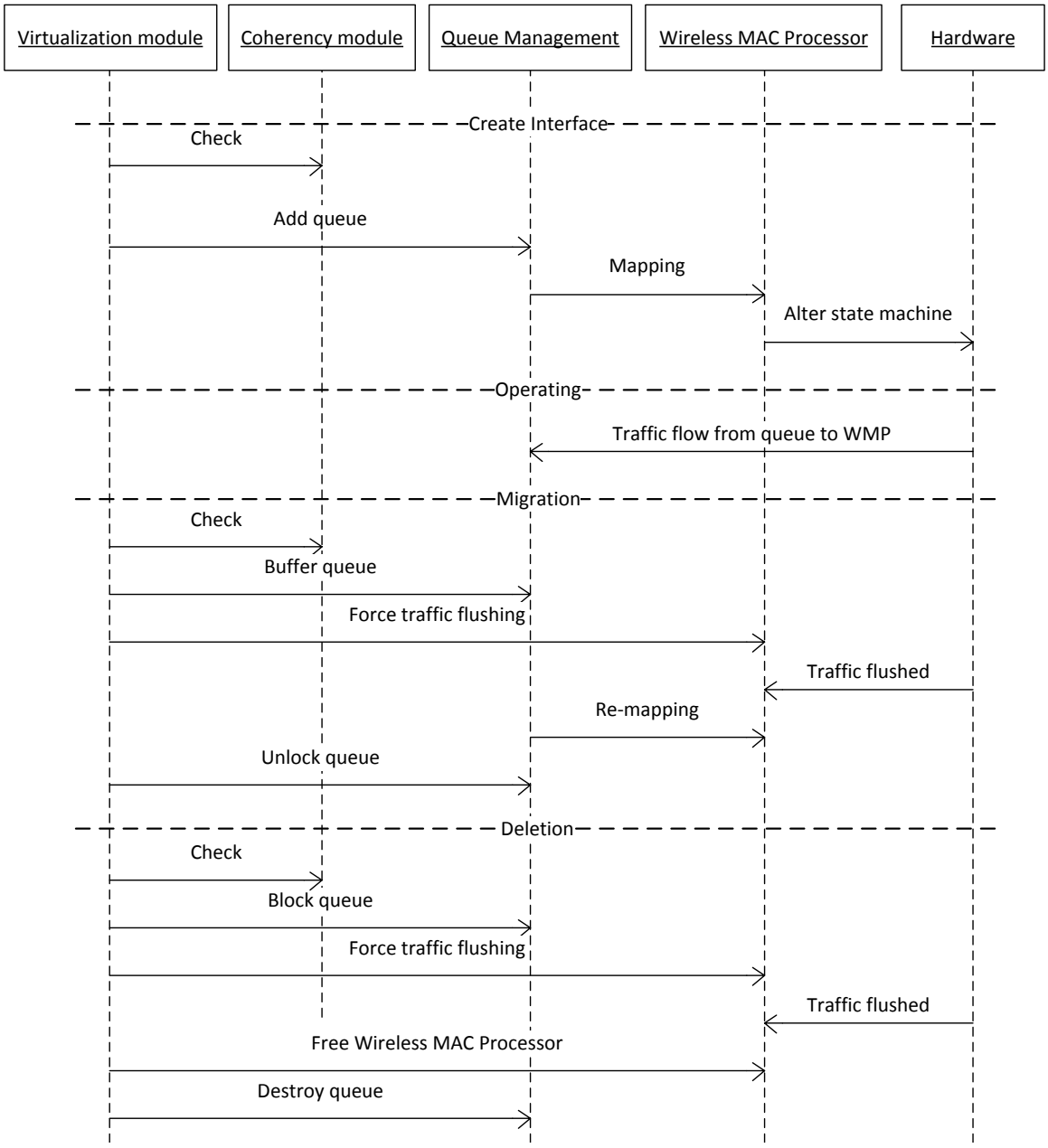


Figure 33: Life cycle of the virtualization module

6 Interface specification

The Interface Access Points (IAPs) describe general interfaces offering an abstract framework so that the different FLAVIA entities communicate with each other utilizing the functionalities provided by the architecture. Following the work carried in the previous deliverable, D4.1.1 [2], we define the interfaces by means of primitives, keeping a high degree of flexibility. The primitives are operations requested to an entity and require a set of parameters to be executed. Note that in this deliverable we specify the framework for these interfaces, which will be detailed after the implementation experiences in D4.3.

A complete notation of a primitive over an interface will begin with the interface identifier, followed by a short operation notation ended with the primitive type:

<Module Name>_<IAP ID>_<Operational Name>_<Primitive Type>

Table 8 shows the identifiers corresponding to each type of interface, according to the possible interactions among the different FLAVIA components, whereas Figure 34 depicts an overview of these interactions.

ID	Interface	Description
IAP1	WMP – MAC Interface	Control and data transfer between the wireless MAC processor and MAC layer
IAP2	Inter–entity Interface	Interactions between services in different entities (e.g., among mobile stations or between the access point and mobile stations)
IAP3	Services – Functions Interface	Functionality required by services from functions
IAP4	Control & Mgmt. Interface	Configuration and management
IAP5	Inter Services Interface	Services interactions within the same entity
IAP6	Application Interface	Communication between FLAVIA services and upper layers (Control and data)

Table 8: Interface Access Points Identifiers

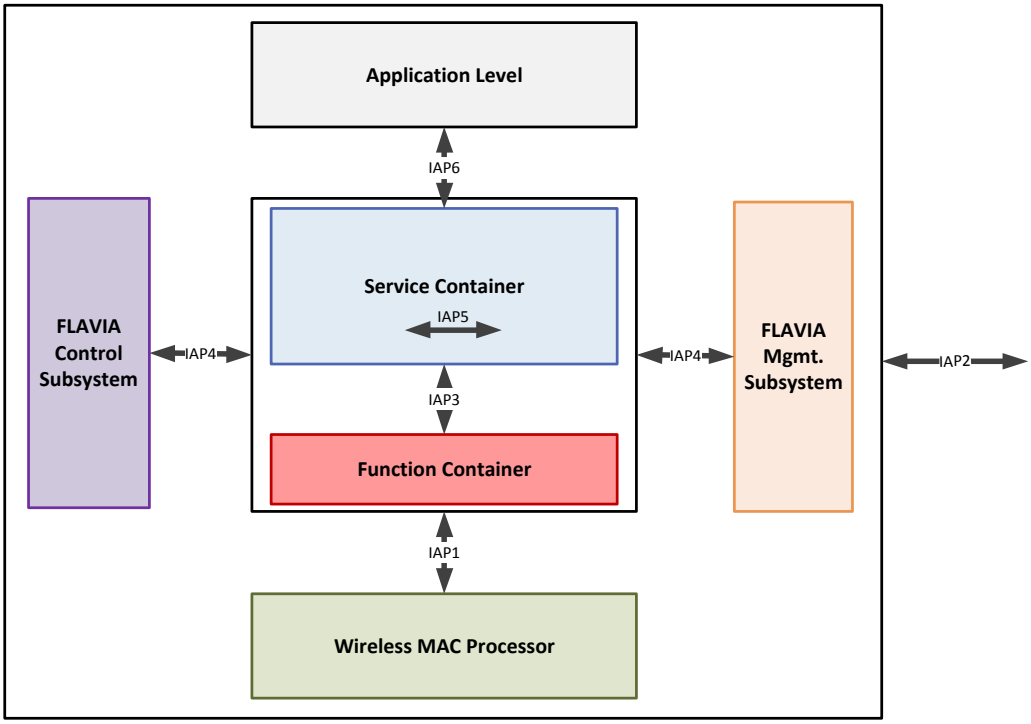


Figure 34: FLAVIA entities interaction

6.1 Intra-node

6.1.1 Application Interface (IAP6)

IAP6 permits the application layer to exploit the underlying FLAVIA services, accessing to the functionalities implemented by the lower layers, e.g.: advanced data transport or monitoring. For example, a user may desire to configure the type of the power saving policy.

6.1.2 Inter-services Interface (IAP5)

IAP5 enables the communication among different FLAVIA services, i.e., exchanging configuration parameters and current status or forwarding network performance statistics. A use case for this interface is the monitoring service that collects the network information (channel quality, interference, etc.) that can be used by other services, such as SPS, to select the best network configuration, or by MDR in order to detect misbehaving nodes.

Another example, in the case of sending a frame, services such as Monitoring or MAC

Management will need to interact with Data Transport service. Data Transport service does not define a specific primitive for each service, but a common one (*send_frame(type)*), so that a general definition is accomplished. This generalization facilitates adding new services within a future without the need of modifying the existing framework, since there will be no need of defining additional primitives; subsequently, being aligned with the FLAVIA vision in terms of flexibility.

6.1.3 Control & Management Interface (IAP4)

IAP4 provides a common interface to the 802.11 service modules to interact with the control and management subsystems so they can be reconfigured according to specific events. This interface is used to set and polled information stored in the Information Base, guaranteeing always the system consistency. This information might be relative to the admissible set of configurable parameters or the current status of variables. This interface also requests the access to common resources handled by the virtualization manager.

An example to illustrate this case is the activation of power management that triggers the modification of the configuration other running services, like the data transport and the monitoring services, which should stop their execution during periods of inactivity.

6.1.4 Services - Functions Interface (IAP3)

Services can invoke functions contained in the Function Container through the IAP3 interface. This interface manages the concurrent access of different services to the same function, avoiding race conditions.

6.1.5 WMP - MAC Interface (IAP1)

IAP1 enables the configuration of the WMP, as well as the request of WMP parameters, collected to be used by upper services, such as monitoring, when is reported with the network scanning results.

6.2 Inter-node

In this section we introduce the mechanism considered to implement a generic inter-node communication as required by an 802.11 FLAVIA scenario. Instead of specifying from scratch a protocol, tailored to FLAVIA needs, we rather rely on the Generic Advertisement Service (GAS) from the 802.11u standard [8]. This is a mechanism that provides a flexible interface with some defined primitives, and is readily available in some Linux-based implementations, although is typically considered for a service discovery scenario (therefore it will be extended to support the IAP2 implementation).

6.2.1 Inter-entity Interface (IAP2)

The Inter-entity Interface is responsible for the communication among processes running in different nodes, i.e.: between stations or between an access point and a station. These processes can be operating services or functions. This interface provides different primitives in order to allow the transfer of information among these nodes.

By means of IAP2, a running service of a node can intercommunicate with another instance of the same service operated in a different node, in order to:

- Keep consistency on some essential configuration parameters. An example of this case is the announcement of the value of the configure parameters, e.g., rate adaptation service.
- Trigger a join reaction towards certain events. For example, in case of link quality degradation or possible occurrence of interference.
- Decide or negotiate on possible network capabilities, even prior to the association.
- Disseminate information to assist internode cooperation.

A potential way to implement this communication is given by the standard procedure defined in 802.11u [8], Generic Advertisement Service (GAS). GAS is used as a container for Access Network Query Protocol (ANQP) elements sent between clients and APs.

GAS provides functionality that enables STAs to discover the availability of information related to desired network services, e.g.:

- Information about services such as provided in an IBSS, local access services, available Subscription Service Providers (SSP) and/or SSPNs or other external networks.

GAS uses a generic container to advertise network services' information over an IEEE 802.11 network. Public Action frames are used to transport this information. Its basic operation is depicted in Figure 35.

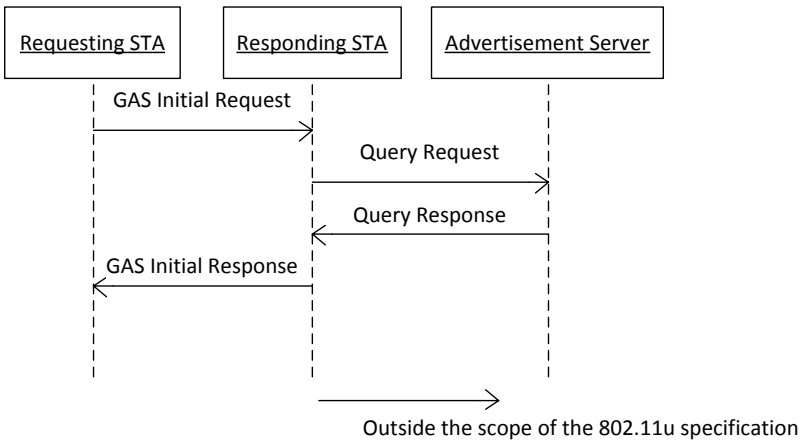


Figure 35: GAS operation

The structure provided by the GAS request and carried within an ANQP element (depicted in Figure 36) are extendable in order to specify query primitives that are FLAVIA-aligned. Thus, we can specify new messages in order to extend the communication among different nodes.

Info ID	Length	Information
2	2	variable

Figure 36: ANQP element format

6.3 Summary of primitives

Table 8 collects the complete set of primitives proposed for an 802.11 FLAVIA node.

Type	Primitives	Description
IAP6	TRAN_IAP6_send_frame_REQ TRAN_IAP6_send_frame_RSP TRAN_IAP6_receive_frame_IND TRAN_IAP6_receive_frame_CONF	The transport service exposes its interface to the upper layers to enable sending packets through the 802.11 MAC layer.
	MONI_IAP6_set_monitype_REQ MONI_IAP6_set_monitype_RSP MONI_IAP6 moni_start_REQ MONI_IAP6_moni_start_RSP MONI_IAP6 moni_stop_REQ MONI_IAP6_moni_stop_RSP MONI_IAP6 estimate_interference_REQ MONI_IAP6 estimate_interference_RSP MONI_IAP6_estimate_link_distances_REQ MONI_IAP6_estimate_link_distances_RSP	The Monitoring service will also expose its interface so that applications can set up the type of scanning to be performed and modify configuration parameters, e.g., the time spent to perform the active monitoring
	PHYR_IAP6_get_phy_REQ PHYR_IAP6_get_phy_RSP PHYR_IAP6_set_phy_REQ PHYR_IAP6_set_phy_RSP	The PHY Resource Management service is expected to interact with the upper layers to return or configure an explicit set of PHY, such as the rate adaptation algorithm to use. This interface allows the configuration and inspection of current values for each of the managed parameters.
	PS_IAP6_set_policy_REQ PS_IAP6_set_policy_RSP PS_IAP6_get_policy_REQ PS_IAP6_get_policy_RSP	The power saving service exposes its interface to the upper layers in order to return or configure the type of power saving policies.
IAP5	TRAN_IAP5_send_frame_REQ TRAN_IAP5_send_frame_RSP TRAN_IAP5_frame_sent_IND TRAN_IAP5_frame_sent_CONF TRAN_IAP5_receive_frame_IND TRAN_IAP5_receive_frame_CONF	The Transport service interacts with all the other services, therefore being an essential service in the FLAVIA architecture. It receives request commands from other services, in order to transmit the types of frames according to the other services' selection.
	PHYR_IAP5_transmit_REQ PHYR_IAP5_transmit_RSP PHYR_IAP5_get_phy_REQ PHYR_IAP5_get_phy_RSP PHYR_IAP5_set_phy_REQ PHYR_IAP5_set_phy_RSP PHYR_IAP5_enable_autorate_REQ PHYR_IAP5_enable_autorate_RSP PHYR_IAP5_disable_autorate_REQ PHYR_IAP5_disable_autorate_RSP PHYR_IAP5_set_phy_IND PHYR_IAP5_set_phy_CONF	The PHY Resource Management service is expected to interact with other FLAVIA services, e.g., Monitoring and Transport service. This service can receive an explicit set of PHY parameters to be utilized for each packet transmission, but may also request transmission statistics from the Transport service when automatic rate adaptation is performed.
	PS_IAP5_dequeue_ps_frame_REQ.	The PS service interacts with the Data

	PS_IAP5_get_psm_status_REQ PS_IAP5_get_psm_status_RSP PS_IAP5_buffer_ps_frame_REQ PS_IAP5_buffer_ps_frame_RSP PS_IAP5_dequeue_ps_frame_REQ PS_IAP5_dequeue_ps_frame_RSP	Transport so that the buffered frames are dequeued or enqueued depending on the state of the stations.
IAP4	IB_IAP4_get_data_REQ IB_IAP4_get_data_RSP IB_IAP4_on_change_listener_REQ IB_IAP4_on_change_listener_RSP IB_IAP4_create_data_REQ IB_IAP4_create_data_RSP IB_IAP4_change_data_REQ IB_IAP4_change_data_RSP IB_IAP4_clear_data_REQ IB_IAP4_clear_data_RSP IB_IAP4_set_value_REQ IB_IAP4_set_value_RSP IB_IAP4_delete_data_REQ IB_IAP4_delete_data_RSP IB_IAP4_protect_data_REQ IB_IAP4_protect_data_RSP IB_IAP4_get_service_logic_REQ IB_IAP4_get_service_logic_RSP	The Information Base (IB) exposes one interface, wig, with several primitives to support the operation and interaction with the rest of FLAVIA modules. Operations such as reading and single/multiple writing.
	CM_ICC_IAP4_set_parameter_REQ CM_ICC_IAP4_set_parameter_RSP CM_RC_IAP4_get_consistency_conditions_REQ CM_RC_IAP4_get_consistency_conditions_RSP	The Consistency Manager presents two interfaces, Internal Consistency Check and Remote Consistency, to perform internal and external operations, respectively.
	WMP_IAP4_write_REQ WMP_IAP4_write_RSP WMP_IAP4_run_REQ WMP_IAP4_run_RSP	The WMP exposes an interface to enable the modification of state machines, specifying the activating conditions and enabling code switching.
IAP2	GAS_IAP2_send_request_IND GAS_IAP2_send_request_CONF GAS_IAP2_get_parameter_REQ GAS_IAP2_get_parameter_RSP	The GAS module enables the generic communication among nodes, exchanging different information or performing capability negotiation.

7 Conclusions

This deliverable completes the specification of the designed architectural framework for the implementation of a contention-based FLAVIA node, based on the feedback received during the revision of WP2 architecture and the module specification.

Based on the evolution of the general framework provided by WP2 and on the architecture for an 802.11 node presented in D4.1.1, we have reviewed and updated the set of service and function modules as well as the interface specification. New services that may be loaded in real-time in a FLAVIA-enable node have been added, e.g.: Advance Data Transport and Misbehaviour Detection, showing that the FLAVIA architecture is not tailored to neither the 802.11 standard nor very simple extensions (these were the focus of the previous deliverable).

In addition, we have detailed the operation of the Service and Function containers liable for scheduling and managing new services and the registration of enhanced functions under the premises of a modular and flexible architecture, focusing on the dynamics of the instantiation of the services and how the FLAVIA functionality supports the their real-time execution. We have described the architecture of those two containers, detailing their static interfaces, which permit to access their services and the interactions with other entities that may occur during their execution.

The Wireless MAC Processor is described extensively in this deliverable from the architectural perspective. Starting from the description carried out in D.2.2.1 and D.2.2.2 for a generic programmable radio system, here we specify the set of events, conditions and actions for contention-based 802.11 systems on the basis of the analysis of DCF lower-MAC operations, which defines the primitives to support the different functionalities. We also present how the WMP interacts with other modules in order to perform its basic operations.

We extend the description for the Control and Manager subsystems, as well as add new entities that were not included in the previous deliverable, as the Consistency Manager. The Consistency Manager avoids potential race conditions about the various pieces of information that will be stored in the shared repository, namely, the Information Base, which stores the configuration and operation parameters and whose operation is described in this document. In the case of the virtualization manager, it is explained how the scheduling process is performed so different MAC instances may access to common hardware resources and present the set of functions that enable this management.

Finally, we also describe the intra- and inter-node communication, emphasizing on this previous one and proposing a mechanism to implement a generic inter-node communication built on GAS, which is specified in the 802.11u standard.

References

[1] IEEE 802.11, *LAN/MAC Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Medium Access Control (MAC)*. Revision of IEEE Std 802.11-1999, 2007.

[2] FLAVIA Project Deliverable D.4.1.1 – *802.11 architecture and interfaces specification*, June 2011, available at http://www.ict-flavia.eu

[3] FLAVIA Project Deliverable D.4.2 – *802.11 modules specification*, November 2011, available at http://www.ict-flavia.eu

[4] FLAVIA Project Deliverable D.2.1.1 - *Report on Scenarios, Services and Requirements*, January 2011, available at http://www.ict-flavia.eu

[5] FLAVIA Project Deliverable D.2.1.2 – *Revision of Report on Scenarios, Services and Requirements*, January 2012, available at http://www.ict-flavia.eu

[6] FLAVIA Project Deliverable D.2.2.1 - *Architecture Specification*, May 2011, available at http://www.ict-flavia.eu

[7] FLAVIA Project Deliverable D.2.2.2 - *Revision of Architecture Specification*, May 2012, available at http://www.ict-flavia.eu

[8] IEEE 802.11u, Amendment to Standard for Information Technology. *LAN/MAC Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY Specifications Amendment 9: Interworking with External Networks*, IEEE Std. 802.11u, Feb 2011, Supplement to IEEE 802.11 Standard.

[9] Gallo, P., Gringoli, F., and Tinnirello, I., *On the Flexibility of the IEEE 802.11 Technology: Challenges and Directions*. Future Network and MobileSummit 2011.

[10] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, *Wireless MAC Processors: Programming MAC Protocols on Commodity Hardware*, IEEE INFOCOM 2012, Orlando (FL), USA, March 25-30, 2012.

[11] IEEE 802.11e, Amendment to Standard for Information Technology. *LAN/MAC Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS)*, IEEE Std. 802.11e, Nov 2005, Supplement to IEEE 802.11 Standard.

[12] D. Camps-Mur, X. Perez-Costa, S. Sallent-Ribes, *Designing energy efficient Access Points with Wi-Fi Direct*, Computer Networks, Volume 55, Issue 13, 15 September 2011.

[13] K. D. Huang, Ken R. Duffy and David Malone, *H-RCA: 802.11 Collision-aware Rate Control*, Hamilton Institute technical report, 2011.

[14] Raya, M., Aad, I., Hubaux, J., and El Fawal, *A. DOMINO: Detecting MAC layer greedy behavior in IEEE 802.11 hotspots*. IEEE Transactions on Mobile Computing, 5:1691–1705, 2006.

[15] Serrano, P., Banchs, A., Targon, V., and Kukielka, J. *Detecting selfish configurations in 802.11 WLANs*. IEEE Communications Letters, 14:142–144, 2010.

[16] Ahn, Y. w., Cheng, A., Baek, J., and Fisher, P., *Detection and punishment of malicious wireless stations in IEEE 802.11e EDCA network*. In Proc. of IEEE Sarnoff Symposium. 2010.

[17] Dangerfield, I., Malone, D., and Leith, D.J*. Incentivising fairness and policing nodes in WiFi*. IEEE Communications Letters, 15:500–502, 2011.

[18] Cagalj, M., Ganeriwal, S., Aad, I., and Hubaux, J.-P., *On Selfish Behavior in CSMA/CA Networks*. In Proc. of IEEE INFOCOM. 2005.

[19] Konorski, J. *A game-theoretic study of CSMA/CA under a backoff attack*. IEEE/ACM Transactions on Networking, 14:1167–1178, 2006.

[20] IEEE 802.11s, Amendment to Standard for Information Technology. *LAN/MAC Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY Specifications Amendment 10: Mesh Interworking*, IEEE Std. 802.11s, Sept 2011, Supplement to IEEE 802.11 Standard.

[21] FLAVIA Project Deliverable D.6.2. - *Novel approaches and solutions for contention-based technology enhancements*, Sep 2011, available at http://www.ict-flavia.eu

APPENDIX A: Service building example

In order to illustrate how services can be built over the proposed architecture, we next focus on the case of an advanced data transport with parameterized QoS service. More specifically, we propose to use the IEEE 802.11s [20] mesh reservation-based channel access method, named called MCCA (Mesh Coordinated Channel Access). The MCCA is an optional access method that allows stations to access the wireless medium at selected times with lower contention that otherwise be possible. These times are called MCCA TXOPs (or MCCAOPs). To obtain MCCAOP (to reserve the channel) the transmitter (MCCAOP owner) and the receiver(s) (MCCAOP responder(s)) exchange special management frames. We will further refer to this procedure as the one-hop reservation procedure (more detailed description of MCCA is presented in D6.2 [21]).

In the multi-hop network case, for each flow with a given QoS requirement, the service establishes MCCA one-hop reservations along the multi-hop a path found by path selection protocol (which should take into account the channel access method used), and then transmits data frames using these reservations.

We first present the high-level, functional architecture required by this advanced service, and then the low-level, MAC scheme required to support it.

A.1 Functional Architecture

The functional architecture of the proposed service is illustrated in Figure 37. Each station use two different channel access methods: i) contention-based, DCF, realized by DCF state machine (in future implementations it can be replaced with EDCA) and ii) reservation-based, MCCA, realized by MCCA state machine. DCF is liable for the transmission of background traffic and management frames while MCCA is responsible for the transmission of QoS-sensitive traffic.

Consider the transmission of a single QoS-sensitive flow. First, all the packets of the flow enter the Classifier module. When the first packet of the flow arrives at the source station, using the information from upper layer headers (i.e.: IP, UDP, SIP), the Classifier module determines the TX parameters (e.g.: packet size, packet inter-arrival time etc.) and the QoS requirements of this new flow, and also assigns a Traffic stream Identifier (TsId) to the flow.

Next, the Classifier module sends this information to the *Multihop Reservation Function*. For each flow, with a given transmission parameters and QoS requirements, the *Multihop Reservation Function* reserves resources for the data transmission along the whole multi-hop path, namely, *multi-hop reservation*. Thanks to special management frames, the *Multihop Reservation Function* of each station along the path creates a Software Queue, where packets of the flow are stored before actual transmission, establishing a one-hop reservation for their TX by means of the *OneHop Reservation Function*. Besides, the *Multihop Reservation Function* forwards the QoS

requirements to the next hop along the path. It should be noted that in a one-hop reservation, only packets of the flow for which this reservation was established can be transmitted.

The MCCA Scheduler module stores all the information about the MCCA one-hop reservations of a station and also the reservations of its neighbouring stations. This module obtains the information via advertisement procedure, i.e.: the information is sent periodically in beacons or management frames. When the MCCAOP is reserved for a particular flow, the MCCA Scheduler module calls the function *push_frame_from_queue(i)* that pulls the first frame from the i-th Software Queue corresponding to the flow to the MCCA FIFO hardware queue. Then the command *mccaop_start(duration)* is called, generating an MCCAOP_START event. The parameter duration specifies the duration of the MCCAOP. Upon the reception of an MCCAOP_START event, the MCCA State machine triggers the transmission of the frame.

MCCA-capable stations shall support the Reservation Allocation Vector (RAV) mechanism, which is provided in addition to physical and virtual carrier sense mechanisms to minimize the probability of collisions within the MCCAOP. The RAV represents a time interval reserved for the MCCAOP by station or any neighbouring of this station (i.e., time interval in which station cannot transmit using contention-based channel access method)

A station is not allowed to start the transmission of a frame using contention-based channel access method if this transmission intersects with the RAV. To implement RAV, we propose to use the following commands:

- When RAV starts, the MCCA Scheduler calls the command *rav_start(duration)* that generates the event RAV_START. The duration argument indicates the duration of RAV.
- At the end of RAV, the MCCA Scheduler calls the command *rav_end(next_rav)* that triggers the event RAV_END. It also indicates the *next_rav* time, that is, when the next RAV will start. A station is not allowed to start a transmission when the RAV is set (*rav==busy*) or when the transmission will overlap with the next RAV.

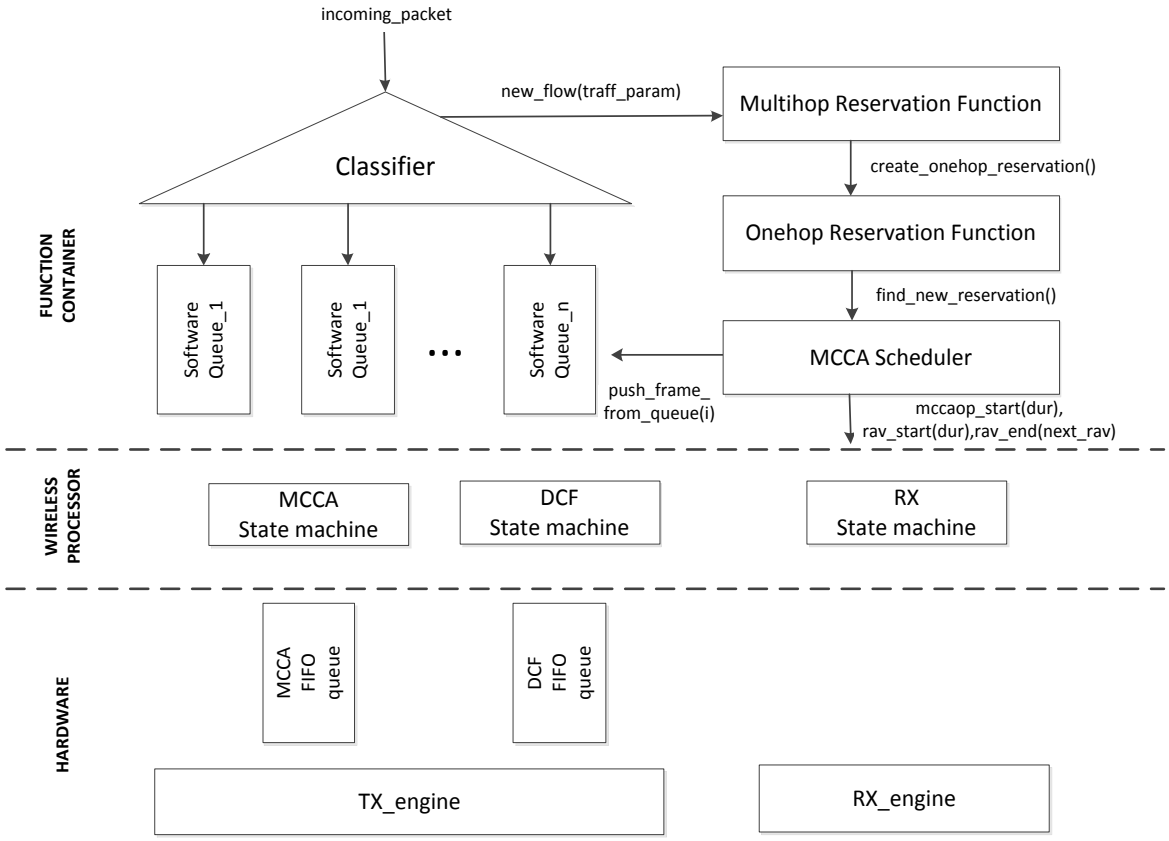


Figure 37: Functional architecture of Data Transport with Parameterized QoS service

A.2 MAC Programs

DCF state machine

Figure 38 shows the state machine for the DCF queue (dotted lines illustrate the difference with respect to the original state machine).

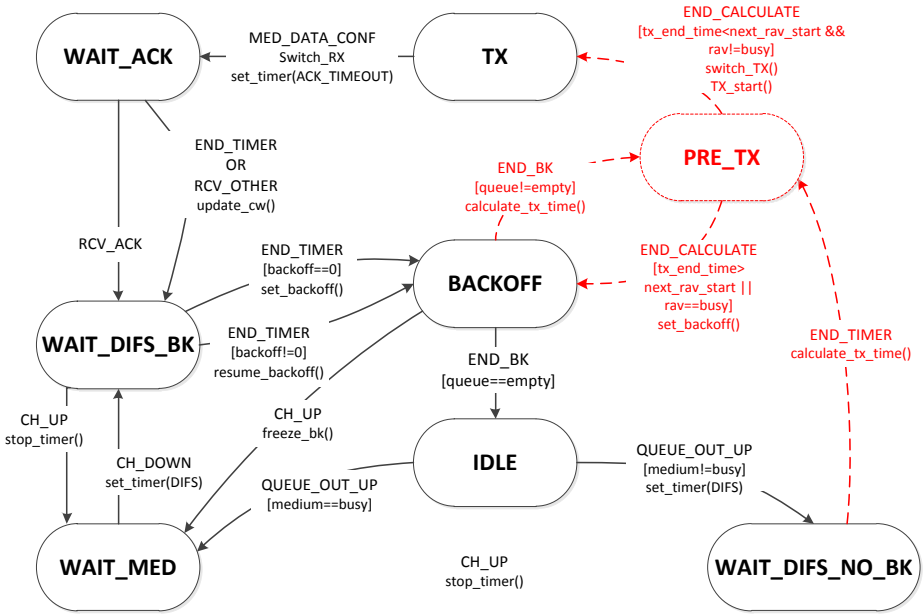


Figure 38: DCF state machine

To implement the RAV mechanism we add a new state, PRE_TX. When the station is ready to transmit it will calculate the frame transmission time, by means of the *calculate_tx_time()* command, and move to the PRE_TX state. If the station time to finish its transmission (TX_END_TIME) is less than the next RAV start and RAV is not set now (*rav!=busy*), then the station can start its transmission and move to TX state**.** Otherwise, the station will defer its transmission and compute a new backoff value, moving to BACKOFF state.

RX state machine

Figure 39 illustrates the RX state machine. The changes to the RX state machine are similar to the ones carried out in the DCF state machine. In this case, we define an additional state, PRE_TX_REPLY. Thus, a station is not allowed to start the transmission of an ACK frame if the ACK transmission overlaps with the RAV.

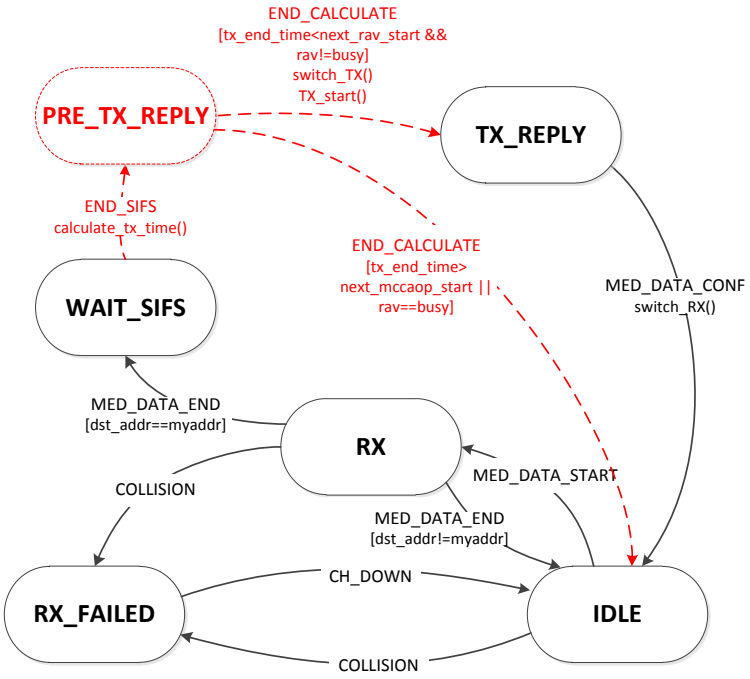


Figure 39: RX state machine

MCCA state machine

The simplified version of the MCCA state machine is presented in Figure 40. In this case, when an MCCA_START event occurs the station should wait a PIFS and then start the transmission if the queue is not empty. Once the station receives an ACK frame or an ACK timeout expired, it should clear the MCCA hardware queue so that the next packet arrives to the empty queue.

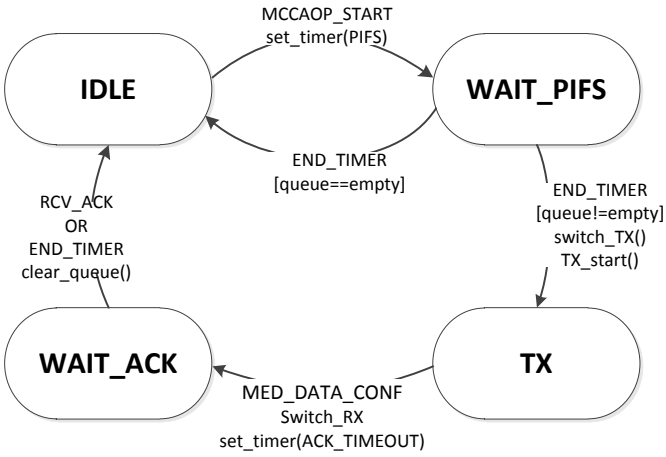


Figure 40: MCCA state machine