EURASIP Journal on
Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

PluralisMAC: a generic multi-MAC framework for heterogeneous, multiservice wireless networks, applied to smart containers

Pieter De Mil*, Peter Ruckebusch, Jeroen Hoebeke, Ingrid Moerman and Piet Demeester

Abstract

Developing energy-efficient MAC protocols for lightweight wireless systems has been a challenging task for decades because of the specific requirements of various applications and the varying environments in which wireless systems are deployed. Many MAC protocols for wireless networks have been proposed, often custom-made for a specific application. It is clear that one MAC does not fit all the requirements. So, how should a MAC layer deal with an application that has several modes (each with different requirements) or with the deployment of another application during the lifetime of the system? Especially in a mobile wireless system, like Smart Monitoring of Containers, we cannot know in advance the application state (empty container versus stuffed container). Dynamic switching between different energy-efficient MAC strategies is needed. Our architecture, called PluralisMAC, contains a generic multi-MAC framework and a generic neighbour monitoring and filtering framework. To validate the real-world feasibility of our architecture, we have implemented it in TinyOS and have done experiments on the TMote Sky nodes in the w-iLab.t testbed. Experimental results show that dynamic switching between MAC strategies is possible with minimal receive chain overhead, while meeting the various application requirements (reliability and low-energy consumption).

Keywords: wireless networks, MAC, multi-MAC, neighbour management, framework, dynamic switching, testbed, smart container monitoring

1. Introduction

Developing energy-efficient MAC protocols for lightweight wireless systems has been a challenging task for decades because of the specific requirements of various applications and the varying environments in which wireless systems are deployed. Many MAC protocols for wireless networks have been proposed, often custom-made for a specific application. However, we have noticed two evolutions in wireless (sensor) networks. First, there is a shift from simple monitoring applications towards multiple applications (or an application with several modes) in the same network. Second, there is a shift from homogeneous devices towards heterogeneous devices (e.g. mobile battery-powered devices with limited processing power and a minimum required lifetime of 2 years, fixed mains-powered devices with many communication interfaces, or battery-powered portable devices which are easy to recharge on a daily or weekly basis). So, how should a MAC layer deal with an application that has several modes (each with different requirements) or with the deployment of another application during the lifetime of the system?

This is exactly the problem we have encountered during the design and implementation of a solution of a smart container monitoring system. The architecture is a complex system of mobile and lightweight devices with stringent energy consumption requirements and varying requirements for the MAC layer because of the peculiarities of the monitoring application.

To the best of the authors' knowledge, we have not found a MAC protocol that can meet all these requirements. Although hybrid MAC protocols exist, these are not sufficient in mobile systems, like the smart container monitoring system. This smart container use case, which will be described in Section 2, has inspired us to create

* Correspondence: pieter.demil@intec.ugent.be
Department of Information Technology (INTEC), Ghent University - IBBT, Gaston Crommenlaan 8 Bus 201, 9050 Ghent, Belgium

a multi-MAC framework and a neighbour management framework (NMF). These frameworks are generic, so they can be deployed in a broader context and deal with other use cases (e.g. building automation, smart cities, etc.) as well that have conflicting requirements that cannot be addressed by monolithic or inflexible solutions.

Instead of designing one monolithic MAC protocol, we will switch between MAC strategies (the so-called maclets) based on the application state and requirements. In Section 3, we will show why one MAC does not fit for all our requirements. Section 4 gives an overview of the PluralisMAC design goals and our architecture is described in Section 5. We will offer common functionalities of a MAC protocol in shared primitives. This way, maclets do not have to implement these themselves. We have implemented this in TinyOS and have done experiments on the w-iLab.t testbed (Section 6). The results of our experiments are presented in Section 7. In Section 8, an overview of related work is given. Lastly, we conclude by a summary in Section 9.

2. An inspiring use case: smart containers

One of the major tasks of supply chain management is to follow goods, stored in containers, from origin to final destination. A huge number of containers are stored in container port terminals, empty or loaded, or on freight ships and tracking their location could increase business efficiency. During the transport of goods, it is important to have an accurate view on the condition of goods (in particular for goods with a limited lifetime, such as food, or goods which needs to be stored at controlled conditions) and on the trajectory.

Smart container monitoring is therefore required. This means that the containers will be equipped with a device that will periodically generate data that need to be sent to a central ICT system in the cloud. However, in order to be accepted by the sector, such a smart container monitoring system must take into account several requirements.

- Lightweight system with a low investment cost.
- Easy to install the system on a container thereby lowering installation costs and avoiding the need for skilled labour.
- Extended battery lifetime: GPRS/UMTS communication is highly energy consuming. Therefore, by realizing energy-efficient communication amongst the monitoring devices instead of sending their information separately, information from several devices could be collected and sent in bulk over a single GPRS/UMTS interface. This avoids many separated GPRS/UMTS connections and thus high energy consumption for each device.
- Increased connectivity for all containers: A 3D stacked container environment is a hostile environment for wireless communication: the containers provide physical, visual and radio shielding. By letting devices communicate with and via each other (multi-hop), containers that are placed at the centre of a big block may still be able to reach the outside world through all the other metal containers without a costly and more powerful transmitter. Our real-life tests indicate that one container can communicate with an adjacent container and the container adjacent to that container (in both directions).
- Ultra-low battery consumption to reduce battery replacements and increase the lifetime of the device: By applying intelligent sleep schemes and communication strategies according to the application requirements, energy consumption can strongly be reduced.
- Allowing a gradual roll-out: Communication between containers can be useful as indicated above. However, such a solution should also be usable even when deployed only in a limited number of containers.
- Additional devices to increase efficiency: next to the devices mounted on the container, additional devices mounted on fixed infrastructure or on the carrier or portable devices are required that are able to interact with the container devices in order to improve communication, reduce energy consumption and enhance process efficiency.

Within the MoCo project [1], in which the authors participate, a smart container system architecture is being developed and implemented. This includes a small, cheap, lightweight monitoring device that will make intelligent use of different wireless technologies (UMTS/GPRS and IEEE 802.15.4 sensor technology) in order to enable optimized and energy-efficient container monitoring. The goal is to realize a cheap generic and energy-efficient connectivity solution including IEEE 802.15.4 sensor technology for communication between 3D stacked containers and making use of tailored and highly energy-efficient protocols.

In the following sections, we will first identify the importance of container monitoring by briefly describing related work. Next, we will highlight the most important aspects and characteristics of our architecture and identify the challenges for the design of an efficient MAC layer for the IEEE 802.15.4 radio.

2.1. Related work on container monitoring

Most of the systems that are used today for monitoring containers use RFID tags to monitor the goods inside

the container or to monitor the container itself. The range of these RFID tags is very limited and it is not possible to access this information over a longer distance. Only recently, researchers have started to use wireless sensor networks for monitoring containers. In [2], an architecture of low-power wireless sensor networks for container tracking and monitoring applications is described. They have proposed three devices: (1) internal monitors (sensor nodes), (2) container monitor (has a WSN interface, GSM and GPS) and (3) Prime Monitor (infrastructure node). IBM's Secure Trade Line [3] proposes a safe and secure solution for monitoring containers. For wide range communication, GSM/GPRS or satellites are used; ZigBee provides short-range communication inside the container. Each container needs to be equipped with a device that either has a GSM or satellite receiver, making the device cost expensive. Networking between the containers is not considered (in [4], it is stated that in IBM's solution, a meshed Bluetooth network can be established if one or more containers do not have a working satellite uplink). The Monitoring and Security of Containers (MASC) system [4] introduces a similar container monitoring system that collects data from sensors inside the container. These sensors are wired connected to an outside antenna that directly reports to a base station. No mesh networking between the containers is used. The communication between the container and the remote server is push-based (because the MASC units are battery powered). The EPC Sensor Network [5] introduces a combined global standard infrastructure for WSNs and RFID systems based on the EPCglobal Architectural Framework [6] to support data sharing between partners. The EPC Sensor Network tries to add WSN support to the initial goals of the framework. However, it is not fully specified how this should be done.

All of these solutions above do not fully investigate the communication issues, but mainly focus on providing services. The communication part is handled by standardized components, i.e. ZigBee or Bluetooth, and mostly focuses on direct communication between a container and an external network. We believe that not all the containers can contact the external network using their own WAN interface because of the hostile environment for wireless communication they operate in.

Recently, some research group started to investigate communication between containers. In [7], multi-hop routing is proposed in order to cope with communication problems caused by multipath propagation. The Intelligent Container project by University of Bremen uses a combination of RFID and sensors within one container [8]. Communication between the containers is executed using different mobile networks, such as WLAN, GPRS or UMTS, depending on availability. The goal is to monitor quality changes that could occur during transport. In [9,10], a hierarchical architecture is proposed: (1) an internal container network for communication using a combination of RFID tags and sensor motes; (2) an external network between containers that uses more powerful gateways, using an IEEE 802.11 interface or a GPRS modem. In this architecture, each container needs to be equipped with such a gateway. This research group is also the first one who has conducted a real-life experiment in a 3 × 3 stacked container configuration. They also found that energy consumption is critical for the success of such a wireless system.

Currently, mainly WLAN is used for this purpose. However, this solution is not power efficient. By using a wireless sensor network (e.g. IEEE 802.15.4 based) and designing protocols tailored to the specific characteristics of the container environment and applications, a longer lifetime and lower costs can be achieved. This is exactly the goal of the MoCo project and its architecture that is being implemented.

Finally, also the following two patents are worth mentioning. In [11], a method for tracking containers using a low-rate wireless personal area network is proposed. The devices will transmit the GPS-recorded, environmental parameter data and sensory reading data to a control station of the cargo vessel through a local area network on the cargo vessel. In our architecture, we do not use a local area network on the cargo vessel. In [12], a method for optimizing power consumption of container tracking devices through mesh networks is proposed. Tracking devices form a mesh network to the edge server installed on the vessel. Again, in our architecture, no infrastructure is required on the vessel.

2.2. MoCo smart container system architecture

In Figure 1, a high-level overview of the complete MoCo architecture, with the different components and connections, is shown. The main device is called the MoCo Device. This is a lightweight battery-powered device that will be mounted on a container and will become an unbreakable part of that container. In addition, it is connected to a door sensor capable of monitoring opening and closing of container doors. Apart from a GPRS/UMTS radio, for direct connectivity to the central ICT system, and GPS, it also has an 802.15.4 radio for short-range, low-power communication with neighbouring MoCo Devices or with a MoCo Router or MoCo Reader. A MoCo Router is an optional component that is very similar to a MoCo Device, but is part of the fixed infrastructure (e.g. the container terminal, a logistics site, etc.), is mostly mains-powered and can have additional network technologies for connectivity to the central ICT system such as Ethernet or Wi-Fi. A MoCo

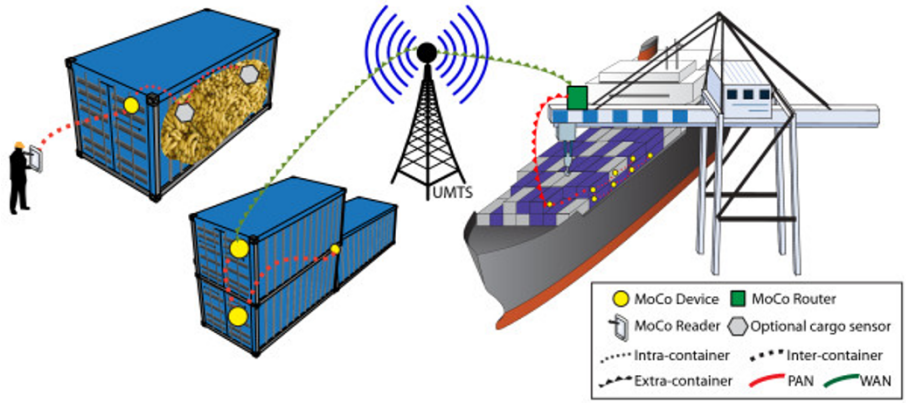

Figure 1 High-level overview of the smart container architecture.

Reader is a portable device, very similar to the MoCo Device, but with a user interface. Finally, the cargo of the container can be equipped with additional sensors for monitoring the status of the cargo.

The MoCo Device will periodically report status information (e.g. position, battery status, etc.) to the central ICT system, will inform about critical events (e.g. opening of door after sealing) and, optionally, will send data collected by sensors attached to the cargo consisting of sensitive and high-value products (e.g. temperature, toxic gasses, etc.). The sensors attached to the cargo will communicate with the MoCo Device over 802.15.4. This is the so-called intra-container communication. In order for a MoCo Device to transmit its status to the central ICT system, several possibilities exist. The MoCo Device can directly send the data by turning on and using its UMTS/GPRS radio or forward it over 802.15.4 to a nearby MoCo Router thereby saving energy. This is called extra-container communication. Alternatively, in order to save battery, the MoCo Device can also transmit the information to neighbouring MoCo Devices, sharing UMTS/GPRS connectivity. The communication between 3D stacked containers is called inter-container communication. Finally, a MoCo Reader can directly interact with a MoCo Device. All communication types are summarized in Table 1, together with some high-level challenges.

2.3. The process

In this section, we will briefly summarize the container monitoring process as determined in the MoCo project. When a container is stored empty, its MoCo Device will daily report about its position and battery level according to a fixed Universal Time Coordinated (UTC) time (e.g. 12:00 UTC time). This way, neighbouring containers will all transmit around the same time and communication can be more easily optimized. This status is called REST mode. When the container is stuffed, its doors are closed and a MoCo Reader will send a trigger, putting the MoCo Device in Secured Transport Mode (STP mode). From now on, the MoCo Device will send 2-hourly status updates and reports of optional sensors attached to the cargo, again according to UTC time. In addition, it will immediately report about any critical event that occurs. Finally, there is an optional mode called Secured Vessel Mode (SVE mode). By sending a trigger from a MoCo Router to a MoCo Device upon loading the container in the vessel, the MoCo Device can go to energy-saving mode, where the 802.15.4 radio is disabled, an internal clock is scheduled and no status updates are sent until the device wakes up again.

2.4. Analysis of the process from a MAC layer point of view

The above process results in very specific communication patterns. When in REST mode, neighbouring

Table 1 Container communication types and their goals and challenges

Communication type	Interface	Goal	Challenges
Intra-container	PAN	Communication between MoCo device and optional cargo sensors in the container	Reliable, energy efficient, self-organizing, secure
Inter-container	PAN	Communication between 3D stacked containers + between container and a portable MoCo reader	Reliable, energy efficient, self-organizing, secure
Extra-container	WAN (PAN)	Communication between the inter-container network and the external world using gateway functionality (+ between container and MoCo Router, using the PAN)	Optimal (best gateway), reliable, energy efficient, self-organizing, secure and evoking a minimum on roaming costs

containers will not transmit anything almost the whole day. Then, once a day, they will jointly transmit their status information to the central ICT system, preferably via a MoCo Router if present, else by sharing UMTS/GPRS or using their own UMTS/GPRS radio. A similar situation occurs in STP mode, but with a higher frequency. In addition, at all times it should be possible to receive a trigger over 802.15.4 and, in STP mode, to send a critical event. This is illustrated in Figure 2. At all times, the main goal is to save as much energy as possible while guaranteeing communication. From a MAC layer point of view (802.15.4-based MAC), this means that in the non-transmitting period the radio should sleep as much as possible, while remaining capable of receiving triggers or sending a critical event. In the transmitting period, data packets should be sent as efficiently and reliably as possible, resulting in completely different requirements on the MAC layer and thus requiring different MAC strategies and scheduling.

From the above description, it is clear that to realize this use case, we have to deal with a complex system of lightweight and mobile devices with stringent energy requirements and varying requirements for the MAC layer. From a research point of view, this imposes many challenges to the design of an efficient MAC protocol for the 802.15.4 radio. In the remainder of this article, we will identify the problems in designing an efficient MAC protocol for this use case and present a generic approach that fulfils our needs and that can also be deployed in a broader context.

3. One MAC does not fit for all the goals

The smart container monitoring process described in Section 2 has revealed the need for a generic and flexible MAC layer that can deal with these changing requirements over time. It is quite clear that a single MAC protocol cannot fulfil all the requirements imposed by our use case. For example, in [13], real-life performance evaluation of WSN protocol combinations has shown that, depending on the traffic pattern and the packet interval, a given combination of MAC and routing is better under certain circumstances and can outperform another combination. The fact that so many MAC protocols [14] exist is also a clear indication that one MAC does not fit for all kinds of applications, network conditions or hardware profiles.

Bachir et al. [15] give an excellent overview of the MAC essentials: a contention-based protocol, like CSMA, does not rely on a central entity and is robust to node mobility, which makes it intuitively a good candidate for networks with mobility and dynamicity, but suffers from degraded performance when the traffic load increases. In contrast, TDMA schemes have some shortcomings resulting from their dependency on network topology and strict time synchronization. This would require large overheads in the smart container use case.

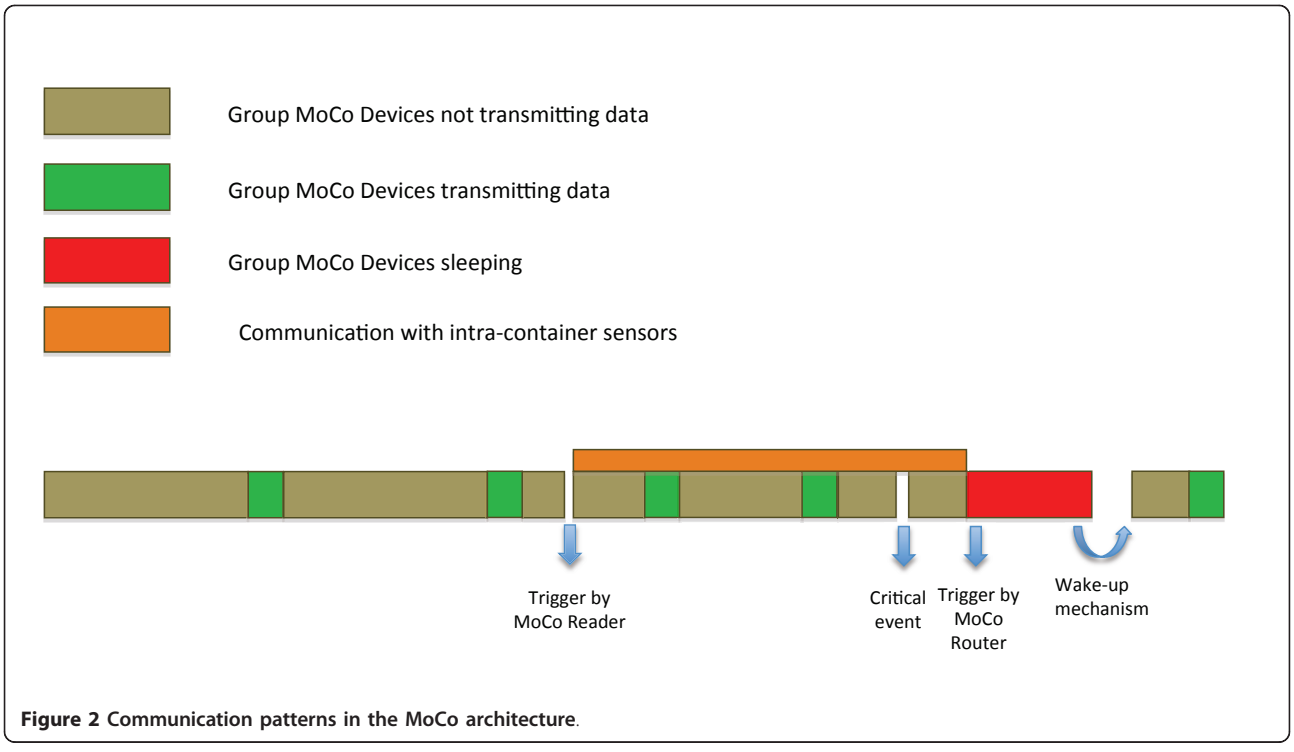

Figure 2 Communication patterns in the MoCo architecture.

Unsurprisingly, there exist some protocols like Arisha [16] where the sink assigns slots to all the nodes or PEDAMACS [17] where the sink gathers information during the setup phase and calculates a global scheduling. In the latter, the traffic pattern is always convergecast. In our use case, we have identified other traffic patterns, so once again: one MAC does not fit all our goals.

4. Design goals of PluralisMAC

The presented use case clearly needs a novel approach for designing a MAC. Often a MAC protocol is designed specifically for one application. From our use case, we are strengthened in our belief that it is time to move away from a dependence on classic monolithic MAC protocols. Therefore, we have designed PluralisMAC. This is a generic multi-MAC framework for heterogeneous, multiservice wireless networks. Our container monitoring use case has inspired the design of this framework, but we want to emphasize that it should be clear that it can be used in a much broader context where mobile and lightweight devices are involved. PluralisMAC has the following design goals:

- Provide extensible MAC primitives: Most MAC protocols share a common set of singular MAC primitives (e.g. DATA, ACK, BEACON, RTS, CTS, preamble, etc.) used for transmitting data and control frames. These primitives have to be provided by the framework in order to improve reusability, to reduce the memory footprint and to reduce the chances of bugs. Composed primitives (e.g. RTS-CTS-DATA-ACK, preamble-DATA, etc.) are a combination of singular MAC primitives. This way MAC protocols can be more easily designed, since they do not have to implement the primitives themselves. Of course, the set of MAC primitives has to be extensible in order to be future proof.
- Support dynamic switching of MAC strategies: We have expressed the need for multiple MAC strategies (even on one node) previously. PluralisMAC must work well with many WSN deployments under dynamic contexts. PluralisMAC should allow fast development and dynamic addition (plug-in) of MAC solutions for (future) use cases and new environments. Therefore, we have introduced a new concept, maclets. A maclet is the short name for a MAC strategy. A maclet can choose MAC primitives and PHY settings (e.g. the sleep scheme, transmit power, etc.). Our framework allows that the application expresses certain requirements (e.g. maximum hop-by-hop latency of the messages) for which PluralisMAC will activate the maclet that provides these application requirements.
- Provide generic neighbour monitoring and neighbour filtering: Neighbour management is a crucial task of many MAC protocols in order to increase efficiency and reliability. Neighbourhood control can be used to filter a good subset of neighbours based on network conditions (e.g. Link Quality Indicator or Received Signal Strength Indication–RSSI), neighbour information (e.g. remaining energy, network ID, etc.), topology constraints and/or application requirements. Often the same information is retrieved and processed. Providing generic neighbour monitoring and filtering at an architectural level can improve reusability. Moreover, because the same information is only processed once we can even reduce processing delay.
- Backward compatible and future proof: It is important that existing MAC implementations can communicate with the maclets, and that it will be easy to implement future MAC protocols. For example, the IEEE 802.15.4-2006 is backward compatible to the 2003 revision. Each revision typically adds new functionalities or primitives. Note that over-the-air programming is out of scope of this article. Protocols for reprogramming wireless nodes via the PAN interface are discussed in [18-20]. In our use case, firmware upgrades of MoCo Devices and MoCo Routers can be done via the WAN connection.
- Provide a research platform for MAC designers: By providing a set of shared primitives and a generic neighbour framework, different MAC designs can be compared with each other. This will yield a fairer comparison of MAC protocols because the same implementation of primitives for a specific implementation of the radio driver is used.

5. PluralisMAC architecture

A schematic overview of the frameworks that together form PluralisMAC is shown in Figure 3. Each framework provides a key functionality of a MAC protocol. The medium access logic (e.g. when to listen, how to send, the data transfer model, etc.) is captured in the multi-MAC framework. Neighbour monitoring and filtering is done in the NMF. The context information module enables a cross-layer information exchange between the layers of the stack. The higher layers (e.g. routing, application), the wrapper, the hardware abstraction layer (HAL) and the radio driver are briefly discussed in Section 6. By splitting up an otherwise monolithic MAC, we get a modular and generic design.

5.1. Multi-MAC framework

The multi-MAC framework is decomposed in several modules, as shown in Figure 4. The multi-maclet framework contains the modules that (bottom-up) (1)

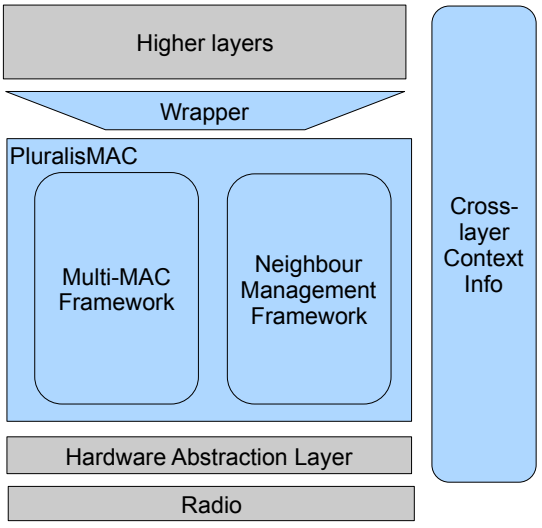

Figure 3 Overview of PluralisMAC in the communication stack.

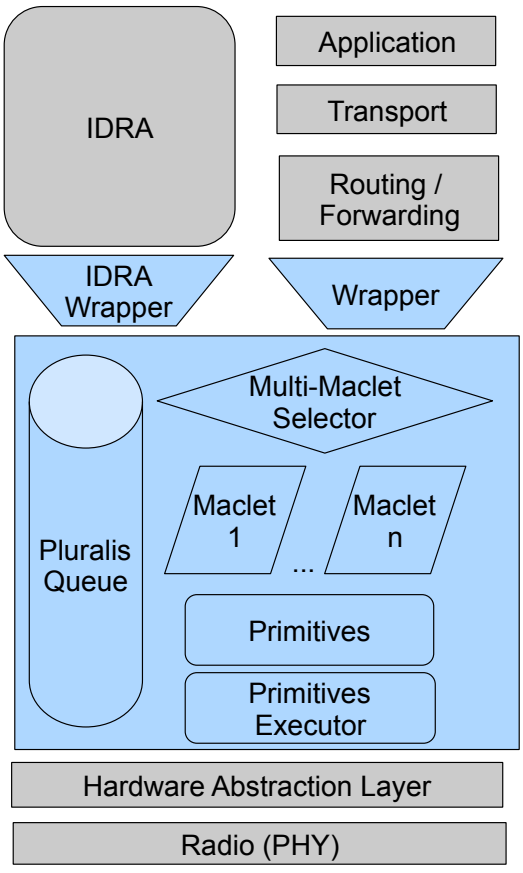

Figure 4 Detailed view of the multi-MAC framework.

implement and execute the shared primitive (primitives executor), (2) contain the medium access protocol logic in terms of chosen primitives (maclets) and (3) the coordination and management of the maclets (multi-maclet selector). The PluralisQueue stores the incoming and outgoing packets, with associated metadata.

5.1.1. Primitives executor

The primitives executor contains the implementation of the various MAC primitives (e.g. data frames, control frames, etc.) and offers also the supported PHY primitives of the HAL (e.g. turn off the radio, wake up, set transmit power, set channel, etc.). Since each MAC protocol uses (some of) these primitives anyway, it makes sense to implement them once instead of multiple times. This way, each primitive implementation will be less prone to errors.

We provide two types of MAC primitives: singular MAC primitives and composed MAC primitives. A singular MAC primitive is a combination of an addressing scheme (unicast or broadcast), a message type (data or control packet) and options (e.g. ack request, clear channel assessment before sending, etc.). Table 2 lists the set of singular MAC primitives. Note that any MAC designer can add a new MAC primitive, but this basic set should be sufficient for most MAC designs. With this set of singular MAC primitives, composed MAC primitives can be created. In one implementation of Low-Power Listening, the MAC repeats the data message during the time of the receiver's sleep interval with a certain inter packet delay. Once the receiver turns on its radio, it will receive the packet. This is an example of a composed primitive we have implemented. The packet train can be stopped before the end of the receiver's maximum sleep interval if an ACK is received. Another composed primitive is useful in receiver-initiated MAC designs. A node will only send data to the next hop if it receives a beacon message from that next hop first. While we provide default timings for these composed primitives, a MAC designer could easily adapt these timings (because it is a composition of singular MAC primitives). This can be done either during design-time, or at run-time with an extra protocol to ensure interoperability.

The primitives executor also uses the interface offered by the HAL. This includes turning on/off the radio and setting the channel, the transmit power, back off intervals for CCA and the node identifier.

5.1.2. Maclets

By providing the primitives, the implementation of a MAC design is simplified and therefore we call it maclets (by analogy with "applet"). In a maclet, a MAC designer chooses the primitives that have to be used on a per-packet basis (if the primitive they want is not

Table 2 Singular MAC primitives

Singular primitives	Addressing	Options
Data	Unicast or broadcast	Ack request, clear channel assessment, retries, channel, transmit power
ACK (acknowledgment)	Unicast	Channel, transmit power
Beacon	Broadcast	Channel, transmit power
RTS (request to send)	Unicast or broadcast	Channel, transmit power
CTS (clear to send)	Unicasts or broadcast	Channel, transmit power

available, it is possible to implement it in the primitives executor). Either a static design is implemented or a dynamic design that can be influenced by the application state (e.g. maximum required hop-by-hop latency), or other context information (e.g. type of power source). A maclet must add metadata to the packet, so that the primitives executor knows what to do. The maclet still determines when to start the execution of a primitive.

A maclet can be created that will try (we say *try*, because it is hard to guarantee Quality of Service in a wireless system) to ensure for example a maximum hop-by-hop latency (e.g. 1000 ms). This could be asynchronous or synchronous. The latter is possible if the MAC designer implements a time synchronization protocol. It is possible to set timestamps in outgoing packets when the transmission has already started, and to set timestamps for incoming packets. These timestamps are added to the metadata.

Each maclet could do neighbour discovery, or a shared neighbour discovery maclet can be created. Typically, neighbour discovery is done during the start-up phase of a network, or if the number of available neighbours drops below a certain threshold. Maintaining up-to-date neighbour information can be done during the execution of any maclet, and is handled by the NMF.

5.1.3. MultiMacSelector

Since one MAC protocol does not fit all requirements, it is possible to offer multiple maclets. Each maclet knows for what kind of applications (e.g. low latency, high reliability, etc.), network conditions (e.g. a dense network) and hardware profiles (e.g. battery powered, energy harvester, etc.) it offers the most optimal medium access solution. First, the maclets will send a registration message containing this information. Next, the best maclet to process a given packet from a higher layer will be selected. This can be based on context information (if available) or a fixed schedule can be chosen. Each maclet can be started, paused, restarted or stopped. When a maclet is paused or stopped, it must cancel any ongoing transmission and the MAC designer can choose to put the radio in sleep mode or receive mode. Since most sensor nodes have only one radio, only one maclet can be activated at the time. If more radios are present, it is possible to assign a maclet to each radio (this will require a simple extension of the multi-maclet selector).

5.1.4. PluralisQueue

It is essential that we can buffer incoming and outgoing messages with associated metadata (listed in Table 3). We have a queue that can be accessed by any PluralisMAC module (and the radio driver) to get a free entry, or to release an entry. The requesting module fills in his module identifier, so that every other module knows who the creator of that entry is. If a module is passed the pointer to this entry, it will check if the "nextOwner" identifier is the module's own identifier and set its identifier in the "owner" field. Other modules are not allowed to change the message or associated metadata if they are currently not the owner. This mechanism is inspired by Berkeley's OpenWSN architecture [21].

5.2. Neighbour management framework

The NMF adds monitoring and filtering capabilities to PluralisMAC. It is common for many MAC protocols to do some processing on packets to gather neighbour statistics like average RSSI, inter packet delay or various other MAC parameters. Therefore, to improve reusability we can extract this common building block and provide it as generic feature. We also provide the possibility to add generic packet filters like a link level duplicate filter and neighbour filters like a RSSI threshold filter. The design of NMF is depicted in Figure 5 and consists of four major building blocks. These will be discussed in detail in the following sections.

5.2.1. Neighbour controller

The neighbour controller provides a generic interface between the NMF modules and the higher/lower layer for processing packets. Using generic interfaces makes NMF portable and also has the advantage that a developer can choose to bypass the NMF completely. The NMF also provides cross-layer interfaces to access the common neighbour repository. This interface can be used by all MAC protocols (maclets) and higher layer protocols at different moments during packet processing. A protocol has to add functions for its specific owner id in order to process and filter packets. These functions are added to the Neighbour Monitor and Neighbour Filter Engine. More details about those

Table 3 Metadata that can be filled in by the PluralisMAC modules and the radio driver

moduleID_t	creator	The identifier of the module that requested a free entry in the queue
moduleID_t	owner	The identifier of the module that is busy processing the entry
moduleID_t	nextOwner	The identifier of the next module that must process this entry
uint8_t	handle	A handle that is used if more than one queue is present in the stack (e.g. IDRA queue) so that we can match entries
uint8_t	Length	Length of the payload. Must be updated if a MAC header is added
Bool	useTransmitPower	If set to FALSE, a default transmit power will be used. Otherwise "TransmitPower" will be used
transmitPower_t	transmitPower	The transmit power for sending packets
Bool	useChannel	If set to FALSE, a default channel will be used. Otherwise, "Channel" will be used
channel_t	channel	The channel on which the packet has to be sent
Bool	useCCA	If set to FALSE, CCA is not used. Otherwise, CCAmode and backoffInformation is used for the clear channel assessment
ccaMode_t	CCAmode	The CCA mode (cfr. IEEE 802.15.4 standard)
backoff_t	backoffInformation	Contains initial backoff and congestion backoff
uint64_t	rxTimestamp	The timestamp set when a packet is received
sleepWakeupSchedule_t	sleepWakeupSchedule	The timings for the sleep/wakeup schedule
int8_t	rssi	Received Signal Strength Indicator
uint8_t	lqi	Link Quality Indicator
Boolean	crc	If set to FALSE, the crc was not correct
uint8_t	macletID	The identifier of the maclet that must process this packet
uint8_t	primitiveID	The chosen primitive that needs to be executed for this packet
uint16_t	nextHop	The next hop address determined by the routing/forwarding layer

functions are given in the following sections. When the neighbour controller processes a packet, it first executes the Neighbour Monitor before passing it to the Neighbour Filter Engine. This way, filters can use information that is already gathered during the processing of packets in the Neighbour Monitor. The Neighbour Controller interface is shown in Table 4.

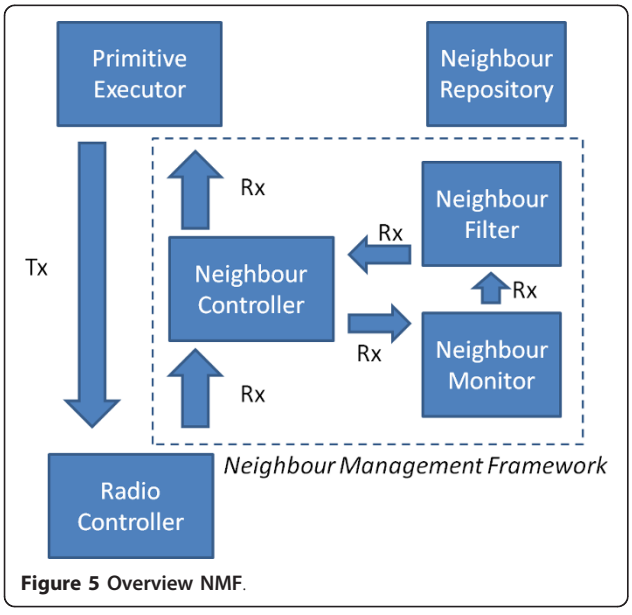

Figure 5 Overview NMF.

5.2.2. Neighbour repository

The Neighbour Repository is designed so that we can have multiple access levels for neighbours. The different access levels define the modules that can view, change and remove neighbour information. The access levels are organized in a hierarchical manner so that every access level exactly has one predecessor and one successor. With the use of access levels we can create different views on the neighbour repository. This feature makes it possible to shield certain neighbours from higher layers. This can be useful when considering the case where a MAC protocol has an always on neighbour discovery period and routing beacons are aggregated with MAC beacons. This could yield to a situation where a routing protocol chooses a neighbour as next hop for a certain destination but the MAC protocol does not create a communication scheme with this neighbour, for instance a TDMA protocol does not assign a slot for this neighbour. When we are able to shield those neighbours from the higher layers, a MAC protocol can still collect information without influencing the behaviour of higher layer modules.

Currently, we have three different access levels: Monitored, Enabled and Activated. A neighbour is always added in the *monitored* state by the Neighbour Controller. If a neighbour passes the neighbour filter checks, the Neighbour Controller changes its state to *enabled*. Note that a filter can also enable a neighbour by default (this will be the common case for most MAC protocols).

Table 4 Neighbour controller interface

Function	FilterResult_t processPacket(uint16_t neighbourAddress, uint8_t ownerID, uint8_t length, void*packet)	
Parameters	uint16_t neighbourAddress	Mac address of the neighbour
	uint8_t ownerID	Owner of the packet. This can be a MAC protocol or a higher layer module. A protocol has to add collectors, aggregators and filters for its specific ownerID in order to process packets
	uint8_t length	Length of the packet pointer
	void* packet	Pointer to the packet or a part of the packet. Because a packet is only processed by specific owners, the packet formatting is always known in advance
Return	FilterResult_t	Returns the result of the filters executed for the ownerID. If no filters are executed, the result will be SUCCESS

A MAC protocol *activates* a neighbour when it has negotiated a communication strategy (e.g. assigned a slot for it in the previous example).

5.2.3. Neighbour monitor

The Neighbour Monitor is in essence an empty shell filled up with functions from now on referred to as collectors and aggregators. The collectors and aggregators are provided by the different protocols of the system, identified by a protocol owner id. The Neighbour Monitor only executes collectors and aggregators for a specific owner id when it processes packets. Collectors retrieve directly accessible parameters from packets or packet metadata like RSSI, timestamps or address information and store it in the neighbour repository. Aggregators combine information retrieved by the collectors in to an aggregated parameter like average RSSI or 'running average over n packets' RSSI and also add it to the neighbour repository.

Although only the collectors and aggregators for a specific owner are executed, they can however use the information stored in the neighbour repository by other collectors and aggregators. Special care has to be taken regarding the execution order of the collectors and aggregators. In the RX- (TX-) chain, a protocol can only use information collected or aggregated by lower (higher) layer modules. This information is available for the developer at design time.

5.2.4. Neighbour filter engine

The Neighbour Filter Engine executes filter rules provided by the protocols of our system. Again only filter rules for a specific owner id are executed. The filter owner (designer) is responsible for combining the different filter rules and aggregating the different results into one single return value. If the return value of the filter engine equals to *success* then the neighbour is activated and the packet is processed further. Otherwise the packet is dropped. This behaviour can easily be adapted however.

Link level duplicate detection is a good example for a filter provided by a MAC protocol. First of all, the MAC protocol designer provides a collector that retrieves the packet id and stores it in the neighbour repository. Secondly, an aggregator adds the previous to last packet id to the neighbour repository. Finally, a filter can check on those two packet ids to determine if the current packet is a duplicate of the previously received packet (of course more packet ids can be stored with another aggregator if one wants to make another duplicate detection filter). In another example, collectors are used for storing RSSI and a packet counter, and an aggregator is used to determine the average RSSI in order to build an RSSI threshold filter.

6. PluralisMAC realization

One of the major goals of PluralisMAC is the ability to evaluate different MAC protocols in realistic use cases and realistic conditions. Our use case presented in Section 2, for which we build a proof-of-concept, is a perfect candidate. In this section, we will give a brief explanation of the different tools we have used to test PluralisMAC and to build our prototype system.

6.1. w-iLab.t testbed

In order to emulate realistic conditions, we make intensive use of the facilities of the w-iLab.t testbed [22]. Figure 6 gives a general overview of the w-iLab.t testbed deployed at our office building. The testbed consists of 200 nodes spread over three floors of a 17.5 × 90 m^2 office building. The architecture of the testbed is based on the widely used MoteLab testbed concept from Harvard University. The deployed devices under test (DUT) are Tmote Sky motes. The intermediate nodes (iNode) are Alix 3C3 devices running Linux. All the iNodes are

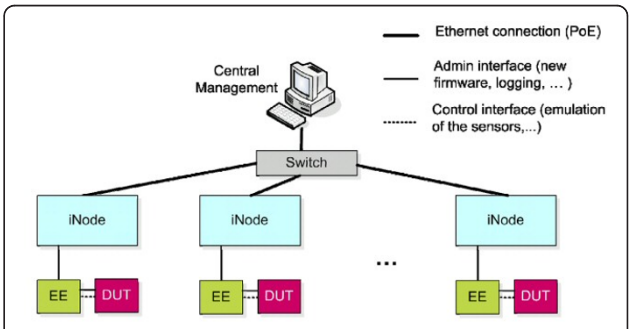

Figure 6 Overview testbed.

connected to the management backbone using Ethernet switches. Finally, the Environment Emulator (EE) sits in-between the iNode and the sensor node. The following features are used from w-iLab.t.

- Central management server: The server is responsible for managing experiments and logging test data received during experiments from the DUT into a MySQL database. The server provides a web interface for creating and executing experiments. The web interface can also be used to control certain features of the EE.
- EE: The EE can disconnect the USB power from the DUT, and power it with its own regulating voltage source. This enables the EE to emulate the real behaviour of a battery depleting. The current used by the DUT can then be measured with a sample frequency of 10 kHz. Using this approach, we can easily determine the exact power consumption during an experiment. The EE has some General Purpose digital Input/Output pins connected to the DUT. This allows for real-life, real-time digital sensor/actuator emulation. We use it in our prototype system to emulate a reliable clock source for the MoCo Device. The EE is synchronized with the central management server using NTP. In a real-life scenario, a MoCo Device would be equipped with a reliable clock on their UMTS or GPS chip.
- iNode: We use the iNode to run a java program that emulates the UMTS connection with the cloud. The java program communicates with the mote over USB using the TinyOS SerialForwarder.
- DUT: The Tmote Sky executes the MoCo Device software.

6.2. IDRA

Beside MAC protocols, we also needed an application and other network protocols. For these, we make use of the IDRA framework [23]. IDRA is a network architecture and application platform very suitable for developing L3 protocols and higher layers. IDRA is developed for TinyOS 2.x and written in nesC. It provides generic building blocks like queue management, packet decoding or aggregation on a system level for network protocols and applications. PluralisMAC is designed to work independent of the IDRA framework but for test purposes we have made an integrated solution and provided wrapper layers between PluralisMAC and IDRA. We reused the IDRA version of the Collection Tree Protocol. We also added a simple gateway discovery protocol and an application. As gateway discovery was not the focus of this article we have chosen a fixed device acting as gateway. Other devices will listen for announcements from this device. The application manages the timers for the different stages in the MoCo process, i.e. transmitting and not transmitting. It generates traffic to the gateway and notifies the MAC and network protocols of the application state and requirement changes through the context information module. The application on the gateway forwards the received application data over USB to the java application on the iNode.

7. Experimental validation

With this experimental validation we want to show that PluralisMAC is capable of executing a standard MAC protocol without introducing a large overhead. The overhead introduced by our system will be measured in different circumstances. The focus of this article was validating an experimental MAC framework in a proof-of-concept scenario. For this reason, we have limited ourselves to a simple scenario with a limited number of nodes. With this evaluation we want to show that

1. It is possible to implement a stable MAC protocol in PluralisMAC.
2. PluralisMAC can work in an energy efficient manner.
3. The delay introduced by our system is acceptable.
4. PluralisMAC is capable of switching MAC protocols.

With this evaluation we do not want to prove that the implemented MAC protocol is the ideal protocol for our use case. In fact the version of LPL that we have implemented in PluralisMAC is a very simple one. No preamble sampling is used and the on duration could be shorter. Moreover, we do not send short preambles but repeat the packet for the length of the LPL sleep interval.

7.1. Proof-of-concept scenario

Based on the use case description in Section 2 we have selected a proof-of-concept scenario that we will use to validate our framework. We consider a network with an increasing number of MoCo Devices. One of these MoCo Devices will announce itself as the gateway (or sink, the used gateway selection algorithm is out of scope) every 2 h. This timing is set according to a UTC time base, which can be obtained via the WAN interface the first time the device is installed. Each MoCo Device knows that data collection is done around UTC 0:00, 2:00, 4:00, etc., in STP mode, and once a day (UTC 0:00) in REST mode. For our proof-of-concept scenario, we focus on the STP mode. We have divided the 2-h timeslot in three periods: first, a long period where the application tolerates a hop-by-hop latency of 30 s (ultra-low duty cycle phase), second, a short (8 s) period in

which the MoCo Devices wait for sink announcements (always-on setup phase), and finally a data collection phase of 52 s, in which the application tolerates a hop-by-hop latency of 1 s. This will repeat every 2 h in STP mode. In the ultra-low duty cycle phase, a MoCo Device could need to send a critical event. In reality, this will occur very rarely, because first the MoCo Device will try sending the critical event using its own WAN interface. Only if that fails, then the MoCo Device will ask a neighbouring node to send the critical event using its WAN interface.

During the setup phase, the selected sink will send sink notifications. We take into account a guard time of 1 s (which is five times more than needed according to our calculations) at the start of this phase to compensate the clock drift during 2 h. All the devices can adjust their clock either during the connection with the cloud or because they receive sink notifications with accurate timestamps. After this period, the neighbours either heard the sink notifications, so they know the gateway, or they did not hear a sink notification and will try to use their own WAN interface. In the former case, the MoCo Device will send its monitored data (five packets) to the sink in the data collection period. In the latter case, the node can switch to the ultra-low duty cycle period already.

7.2. Experiment setup

We used five nodes in our experiment: one fixed MoCo Router acting as sink device and four regular MoCo Devices. We chose this specific setup because the main focus is on validating the framework and not on testing protocols. By limiting the number of nodes we could examine the test data more carefully.

During an experiment we gradually increase the number of nodes, going from one regular node and one fixed sink to four regular devices. This way we can measure the influence of increasing density on the workload. This is also the reason why we choose to use a fixed sink device.

In Table 5, we list the real-time periods and the time periods used in our experiments. We have used other timings because otherwise the experiments would last too long. We have only reduced the ultra-low duty cycle phase in our experiments. The report interval has been rescaled to 180 s. During this period, we have 120 s of not transmitting state where we maintain an ultra-low duty cycle of 30 s with an on duration of 100 ms. The other 60 s of transmitting state are divided in an 8 s always on setup period and a 52 s data collection period. The total experiment duration is 2160 s or 36 min. The experiment is divided into four stages with increasing node numbers going from two nodes in stage 1 to five nodes in stage 4. In every stage, three report intervals are executed that leads to 540 s per stage.

Table 5 Proof of concept time periods and their counterparts in our experiments on the testbed

Period	Real length	Testbed length
Ultra-low duty cycle	119 min	120 s
Setup	8 s	8 s
Data collection	52 s	52 s

In the beginning of each report interval, nodes are synchronized by the EE with an interrupt on one of the GPIO pins. This emulates the behaviour of a stable oscillator that would be integrated on the MoCo Device. The synchronization is not perfect but even a good oscillator cannot provide a perfect synchronization on its own. We measured a variance of ± 500 ms. This can be due to several reasons. Note that the trigger for the interrupt generator originates from the w-iLab.t web interface and has to be processed by the central management server, send over ethernet to the iNode and from the iNode to the EE over USB where the request is processed again.

7.3. Process description

The report interval always begins with an always-on setup period. During this period, all devices first listen for 1 s to compensate for the variance in synchronization (guard time). Then the sink sends six sink notifications, one every second. On hearing a sink notification, a regular node checks if the packet is not a duplicate. This is necessary because the sink notifications are broadcasted and propagated in the network. The packet is dropped when a duplicate is detected; otherwise two metrics are used to consider the sender of the packet as best next hop to the sink:

1. Minimum average RSSI threshold: this information is gathered by the NMF. By using the NMF, we automatically include RSSI information from duplicate packets received from this neighbour.
2. Minimum number of hops to the sink.

After 8 s every node should have found a sink device and starts the data collection period. In the data collection period, an LPL sleep interval of 1 s is maintained. This information is automatically deducted from the application requirements and stored in the context information module. Using the context information module makes it possible to reuse our MAC protocol for other applications with different requirements. During this period, the regular devices send six packets to the sink device. Because we use LPL, we actually keep repeating the same packet for the duration of the LPL sleep interval or until the packet is

acknowledged. The regular nodes have 52 s to transmit their data.

After the data collection period, the transmitting state is finished and the nodes fall back to an ultra-low power state. In this state, we maintain an LPL sleep interval of 30 s. This value is mainly deducted from the use case where the main goal is to save energy. We still need to maintain a duty cycle because we have to be able to receive triggers from MoCo Routers or MoCo Readers. For test purposes, we let a random node send one packet to the sink in this period.

7.4. Results

7.4.1. Delay measurements

Table 6 gives an overview of the results. We focussed on delay introduced by our system. We measured the delay by generating interrupts on the EE. Because all tests are performed on one node we do not have to compensate for the synchronization variance. For the receive chain, we measured an average of 3.8 ms delay for unicast packets and 4.6 ms delay for broadcast packets. This is mainly due to the fact that duplicate detection has to do more processing for broadcasted packets. The NMF introduces an average delay of 1.8 ms. In the transmit chain, we measure the delay between a call to send in PluralisMAC and the send done event from PluralisMAC. This is also dependent on the MAC protocol behaviour, especially when using different duty cycles for LPL. Therefore, we have averaged out the result for every stage in our process. The lowest delay we notice is 72 ms for broadcast packets in the always-on setup period. This is fairly high but it includes a packet conversion between IDRA and PluralisMAC and two memcpy instructions, one from the IDRA queue and one to the radio driver. It also includes the delay introduced by the radio driver. In the data collection period, we notice an average delay of 292.7 ms. This is not bad considering we use an LPL sleep interval of 1000 ms. The average delay in the ultra-low duty cycle period is 29103.5 ms. We give this value for the sake of completeness because this value is only averaged over 12 packets.

7.4.2. Reliability

Table 7 shows the link level reliability. This does not include reliability after retransmissions. The reliability is measured using packet numbers included in every packet. Gaps in between packet numbers at the receiver side indicate packet loss. We notice 100% reliability for broadcast packets in the always-on setup period, for unicast packets in the ultra-low duty cycle period and for unicasts in the data collection period with two nodes. The reliability for unicast packets drops to 92.9% when more nodes content for the medium in the data collection period. Although this seems bad, we only loose 1 packet per 18 packets, representing 1 packet per 3 reports intervals or 1 packet per test stage. If we would use retransmissions, we would obtain better results for the reliability but as reliability is not the focus of this article, we will leave this for future work.

7.4.3. Energy consumption

Table 8 shows the average current consumption for the sink node and the first regular node for each period in function of increasing node density (stages 1 and 4). The measurements are done by the EE attached to the nodes. For the setup period (always-on), we can clearly see that the average current consumption is high and the same for every node. In the data collection period, the current consumption is the same for the sink node, and we see an increase of 8% for the regular node in a network with five nodes. More nodes are sending, so the regular node will need more time before an ACK is received from the sink. In the ultra-low duty cycle period, we can see that the sender (the regular node) has to spend ten times more energy than the listener. The average current consumption for the sink has increased with 18%. We expected to see no increase in current consumption because the sleep and listen scheme of the sink is independent of the number of nodes. This is due to the fact that the EE averages the current measurements over 50 ms, which is not accurate enough for a very low current. The “increase” we measured is thus explained by the fact that 99.67% of the averaged current measurements in the ultra-low duty cycle period fluctuate between minimum 0.26 mA and maximum 0.56 mA (the median is 0.38 mA). A screenshot of the current measurements is shown in Figure 7.

8. Related study

We have proposed a framework that extracts common functionality from MAC protocols (e.g. primitives and

Table 6 Delay measurements of PluralisMAC

Measured process	Measured delay (ms)
Unicast transmit processing delay in ultra-low duty cycle period	29103.5
Unicast receive processing delay	3.8
Unicast transmit processing delay in data collection period	292.7
Broadcast transmit processing delay in always-on setup period	72
Broadcast receive processing delay in always-on setup period	4.6
Processing delay NMF	1.9

Table 7 Reliability measurements of PluralisMAC

Measured process	Reliability (%)	Packets lost per stage
Unicast reliability in ultra-low duty cycle period	100	0
Unicast reliability in data collection period for two nodes	100	0
Unicast reliability in data collection period for greater than two nodes	92.9	1
Broadcast reliability in always-on setup period	100	0

neighbour management). This enables reusability and reduces MAC development effort. The λMAC framework [24] and MultiMAC [25] are good examples of how MAC protocol development can be simplified by providing generic interfaces for common functionalities, hereby reducing MAC protocols to their essential task, namely performing power control (time management) and deciding when to send.

Several modules are available in λMAC. We will compare them with PluralisMAC modules: (1) The packet layer (responsible for the actual sending/receiving of a packet, radio state changes and CCA) is the radio driver in our architecture. (2) The network time layer (responsible for storage and generation of time-synchronization information) has no counterpart in our architecture. Setting timestamps on received packets or adding timestamps in outgoing packets is done in the radio driver (because this is best done below the MAC layer) and the maclets could implement a time-synchronization protocol. We agree that it should be a general service to the entire application stack. A PluralisMAC time management framework is left for future work, because in our use case we obtain time-synchronization information through the WAN interface. (3) The λMAC module (responsible for time management) has no counterpart in our architecture because we do not require allocating time blocks for transmissions. A PluralisMAC maclet designer has the freedom to design his/her MAC protocol with the primitives we offer (or add their own primitives). If the designer knows that, for example, it can send a reply 200 ms after it has received a message received from an energy-harvested node (because the energy-harvested node needs to harvest some "new" energy to turn the radio in receive mode), there is no limitation in PluralisMAC to initiate another primitive that lasts less than 200 ms. (4) The (de-)multiplexer is like our wrapper and multi-mac selector. (5) Transmission layer (contains the unicast, broadcast and other application-level primitives) is like our primitives, except for the fact that we do not require to request time blocks. Furthermore, we have separated singular MAC primitives and composed MAC primitives.

Multi-MAC, an extension of SoftMAC and MetaMAC, is an adaptive MAC framework (or mediating layer on top of MAC protocols) for cognitive radios that automatically selects between multiple MAC protocols. It does not offer common primitives, but reconfigures and/or selects a MAC layer.

Generic neighbour management has already been addressed in literature. The studies of Langendoen et al. [26] and Polastre et al. [27] clearly show the problems that can arise when protocols do not share neighbourhood information. If the selection of a good neighbour differs in the MAC protocol and in the routing protocol (T-Mac and MintRoute [27]), then this will lead to unexpected behaviour like high packet loss or increasing delays. Polastre et al. [27] propose the use of a general neighbour table. This is similar to what we do with the neighbour repository but they do not provide the possibility to add generic functions (collectors and aggregators) to collect neighbour information. Neighbours are added to the neighbour table by asking all the network protocols. How to resolving conflicts is not addressed however. We believe that with our filter engine we have a more generic and flexible approach.

Fonseca et al. [28] proposed a generic 4-bit link estimation that combines information from the physical, MAC and network layer into a 4-bit value. Again this is a good example of the necessity for some sort of generic neighbourhood management but they limit their work to link estimation and provide no flexibility for future extensions. The Link Estimation Exchange Protocol (LEEP) [29] is another example of generic link layer estimation included in TinyOS. LEEP is an example of an active neighbourhood control protocol because it exchanges bidirectional link information among

Table 8 Average current consumption (mA) for the three periods

Period	Two nodes in network (stage 1)		Five nodes in network (stage 4)	
	Sink	Regular node	Sink	Regular node
Setup	18.612	18.784	18.886	18.622
Data collection	2.156	2.210	2.156	2.390
Ultra-low duty cycle	0.443	4.673	0.524	4.903

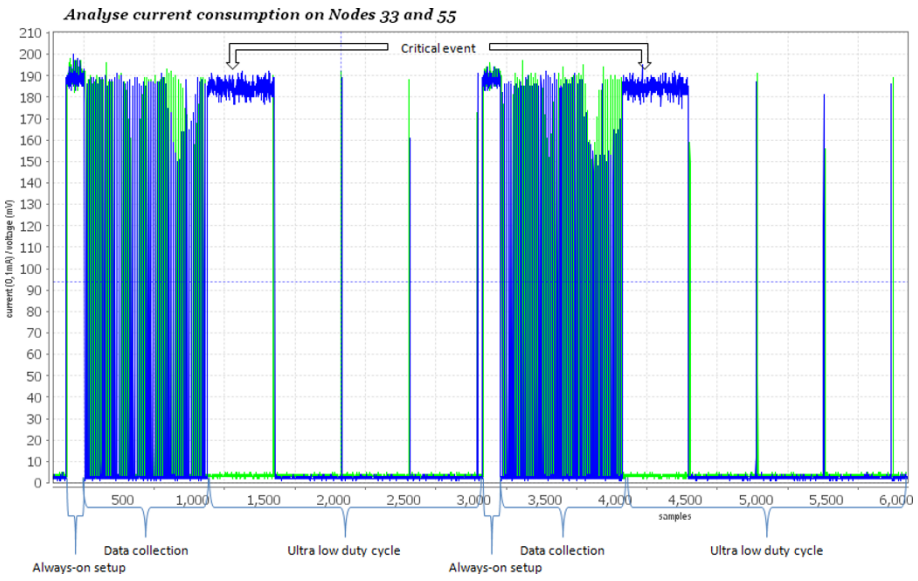

Figure 7 Screenshot of the current consumption measurements.

neighbouring nodes. This feature is not included in the proposed NMF but could be easily integrated when providing the appropriate collectors and aggregators. Active neighbour management will be included in future work.

9. Conclusion

In this article, we have described the smart container monitoring use case which made clear that multiple MAC strategies were needed to achieve a low-power system able to handle wireless communication with heterogeneous devices. We have presented our generic multi-MAC framework and NMF. We have implemented our architecture in TinyOS and have done experiments on the w-iLab.t testbed. We have shown that our framework works, and it does not introduce a significant overhead (e.g. 4 ms in the receive chain for both the NMF and the multi-MAC framework). We will start to implement more (complex) reference maclets, composed primitives and neighbour filters in the near future and hope that our framework will be used by many MAC designers to implement and test their design on real hardware.

Acknowledgements

The authors would like to thank IBBT for their advanced testing infrastructure (w-iLab.t), numerous discussions with our colleagues, and Vincent Sercu for his container artwork. This study was partly funded through the IBBT MoCo project.

Competing interests

Ghent University and IBBT have submitted a provisional patent application.

Received: 15 September 2011 Accepted: 10 May 2012
Published: 10 May 2012

References

1. IBBT MoCo, http://www.ibbt.be/en/projects/overview-projects/p/detail/moco-2
2. S Mahlknecht, S Madani, On architecture of low power wireless sensor networks for container tracking and monitoring applications, industrial informatics, in *5th IEEE International Conference, 2007*. **1**, 353–358 (23-27 June 2007). doi:10.1109/INDIN.2007.4384782
3. S Schaefer, Secure trade lane: a sensor network solution for more predictable and more secure container shipments, in *Companion To the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications OOPSLA '06*, Portland, Oregon, USA, ACM, New York, pp. 839–845 (22-26 October 2006). doi:10.1145/1176617.1176732
4. JO Lauf, D Gollmann, Monitoring and security of container transports. in *Proceedings of New Technologies, Mobility and Security (NTMS)*, Paris http://www.sva.tu-harburg.de/joomla/index.php/research/masc (2007). ISBN: 978-1-4020-6270-4
5. J Sung, TS Lopez, D Kim, The EPC sensor network for RFID and WSN integration infrastructure, in *Proceedings of the Fifth IEEE international Conference on Pervasive Computing and Communications Workshops (PERCOMW)*, IEEE Computer Society, Washington, DC, pp. 618–621 (19-23 March 2007). doi:10.1109/PERCOMW.2007.113
6. CSO EPC Global http://www.epcglobalinc.org/standards
7. D Rogoz, F O'Reilly, K Delaney, Multi-hopping and ad hoc networking for tracking systems. *WiSen, 2nd Workshop on Wireless Sensor Networks Research*, Ireland (2007). doi:10.1007/978-0-387-46264-6_18
8. R Jedermann, C Behrens, R Laur, W Lang, Intelligent containers and sensor networks, approaches to apply autonomous cooperation on systems with limited resources, in *Understanding Autonomous Cooperation & Control in Logistics - The Impact on Management, Information and Communication and Material Flow*, ed. by H?ü?lsmann M, Windt K Springer, Berlin, pp. 365–392 (2007). doi:10.1.1.165.1519
9. SJ Kim, G Deng, SKS Gupta, M Murphy-Hoye, Intelligent networked containers for enhancing global supply chain security and enabling new commercial value, in *3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE 2008)* 662–669 (6-10 Jan 2008). doi:10.1109/COMSWA.2008.4554493
10. SJ Kim, G Deng, SKS Gupta, M Murphy-Hoye, Enhancing cargo container security during transportation: a mesh networking based approach, technologies for homeland security, in *2008 IEEE Conference on* 90–95 (12-13 May 2008). doi:10.1109/THS.2008.4534429
11. C Binding, FB Dolivo, RJ Hermann, D Husemann, A Schade, US Patent 7,541,913. (2 June 2009)
12. J Mermet, B Pucci, US Patent Application 0149028. (17 June 2010)

13. J Vanhie-Van Gerwen, E De Poorter, B Latre, I Moerman, P Demeester, Real-life performance of protocol combinations for wireless sensor networks, in *International Conference on, 2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing* 189–196 (2010). doi:10.1109/SUTC.2010.49
14. K Langendoen, The MAC Aphabet Soup served in Wireless Sensor Networks. http://www.st.ewi.tudelft.nl/~koen/MACsoup
15. A Bachir, M Dohler, T Watteyne, KK Leung, MAC essentials for wireless sensor networks, in *IEEE Communication Surveys Tutorials*. **12**(2), 222–248 (2010). doi:10.1109/SURV.2010.020510.00058
16. K Arisha, M Youssef, M Younis, Energy-aware TDMA-based MAC for sensor networks, in *Proceedings of IEEE IMPACCT*, New York City, NY, (May 2007). doi:10.1007/0-306-47720-3_2
17. S Coleri-Ergen, P Varaiya, PEDAMACS: power efficient and delay aware medium access protocol for sensor networks. IEEE Trans Mob Comput. **5**(7), 920–930 (2006). doi:10.1109/TMC.2006.100
18. A Hagedorn, D Starobinski, A Trachtenberg, Rateless deluge: over-the-air programming of wireless sensor networks using random linear codes, in *Proceedings of the 7th Int Conf On Information Processing in Sensor Networks (IPSN)* (2008). doi:10.1109/IPSN.2008.9
19. S Dawson-Haggerty, NWprog. http://openwsn.berkeley.edu/browser/tinyos-2.x/tos/lib/net/blip/doc/README-NWPROG?rev=620. (last accessed April 30, 2012)
20. D Hughes, K Thoelen, W Horré, N Matthys, J Del Cid, S Michiels, C Huygens, W Joosen, Building wireless sensor applications using LooCI. Int J Mob Comput Multim Commun. **2**(4), 38–64 (2010)
21. OpenWSN: Implementing the Internet of Things http://openwsn.berkeley.edu
22. w-iLab.t Office (Wireless Lab) http://ilabt.ibbt.be
23. E De Poorter, E Troubleyn, I Moerman, P Demeester, IDRA: a flexible system architecture for next-generation wireless sensor networks, Wirel Netw, Springer, 7(6): pp. 1423–1440 (August 2011). doi:10.1007/s11276-011-0356-5
24. T Parker, G Halkes, M Bezemer, K Langendoen, The lambda-MAC framework: redefining MAC protocols for wireless sensor networks, in *Wirel Netw*, vol. 16. (Springer, 2010), pp. 2013–2029. doi:10.1007/s11276-010-0241-7
25. C Doerr, M Neufeld, J Fifield, T Weingart, D Sicker, D Grunwald, MultiMAC–an adaptive MAC framework for dynamic radio networking, in *IEEE DYSPAN* 548–555 (2005). doi:10.1109/DYSPAN.2005.1542668
26. K Langendoen, A Baggio, O Visser, Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture, in *IEEE Parallel and Distributed Processing Symposium* (2006). doi:10.1109/IPDPS.2006.1639412
27. J Polastre, J Hui, P Levis, J Zhao, D Culler, S Shenker, I Stoica, Unifying link abstraction for wireless sensor networks, in *3rd ACM Conf on Embedded Networked Sensor Systems*, San Diego, CA, (2005). doi:10.1145/1098918.1098928
28. R Fonseca, O Gnawali, K Jamieson, P Levis, Four bit wireless link estimation, in *Proceedings of the 6th Workshop on Hot Topics in Networks (HotNets VI)* (2007). doi:10.1.1.74.2030
29. O Gnawali, TinyOS Extension Protocol 124, The Link Estimation Exchange Protocol (LEEP). http://www.tinyos.net/tinyos-2.x/doc/txt/tep124.txt

doi:10.1186/1687-1499-2012-166
Cite this article as: De Mil *et al.*: **PluralisMAC: a generic multi-MAC framework for heterogeneous, multiservice wireless networks, applied to smart containers.** *EURASIP Journal on Wireless Communications and Networking* 2012 **2012**:166.