Performance Evaluation of Default Active Message Layer (AM) and TKN15.4 Protocol Stack in TinyOS 2.1.2

Diego V. Queiroz[1], Ruan D. Gomes[1,2,3] and Cesar Benavente-Peces[1]
[1]*Signal Theory and Communications Department, Universidad Politecnica de Madrid, Madrid, Spain*
[2]*Informatics Coordination, Federal Institute of Paraiba, Guarabira, Brazil*
[3]*Post-Graduate Program in Electrical Engineering, Federal University of Campina Grande, Campina Grande, Brazil*

Keywords: Wireless Sensor Networks, TinyOS, TKN15.4, Active Message, IEEE 802.15.4.

Abstract: Wireless Sensor Networks (WSN) have become a leading solution to monitor and control smart buildings, health, industrial environments, and so on. Sensor nodes in a WSN have resource constraints, presenting low processing power and, in some cases, restrictions in power consumption. The resource constraints forced the researchers to develop Operating Systems (OS) for low-power wireless devices, and one of the most important and in active use is the TinyOS. This paper presents an experimental study to evaluate the performance of TinyOS default Active Message (AM) layer protocol in comparison to the fully 802.15.4 compliant protocol stack TKN15.4 developed for TinyOS. The AS-XM1000 802.15.4 mote modules were used to compare both protocols. The results showed that TKN15.4 protocol is better in both energy consumption and packet reception rate.

1 INTRODUCTION

Wireless Sensor Networks (WSN) present significant advantages in comparison to wired networks, such as flexibility, reconfigurability, easy installation/maintenance, ability of self-organization and local processing, becoming a promising platform to implement online systems for remote monitoring and controlling in different types of environment. Despite these advantages, the WSN work in an inherently unreliable communication medium, so they are subject to typical problems of wireless channels such as attenuation, multipath, shadowing, fading, noise and interference in the spectrum band used for communication and shared by a number of users, e.g. in the 2.4 GHz band.

There are many environments where the wireless nodes can be deployed. Among home, office, and industry, the industrial environment is harsher due to the unpredictable variations of temperature, pressure, humidity, and so on. In addition, the wireless channel in many industries is non-stationary for a long term, which can cause abrupt changes in the characteristics of the channel over time (Agrawal et al., 2014). In industry, the coherence bandwidth is low due to the high level of multipath fading, which causes differences in the characteristics of the different channels, since they are uncorrelated in frequency, and the impact of multipath is different for different channels. In addition, changes in the topology of the environment, such as the movement of a large metal structure, people and reflections, may cause changes in the characteristics of the channel over time (Gomes, R. D. et al., 2016).

The lack of reliability in the communication medium makes it difficult to establish Quality of Service (QoS) guarantees with reduced CAPital EXpenditure (CAPEX) and OPerational EXpenditure (OPEX). Therefore, the sensors, including their softwares, need to be low-cost, resulting in a set of restrictions, such as low data rate and low processing capabilities. For this reason, Operating Systems (OS) such as TinyOS, Contiki, OpenWSN, RIOT and FreeRTOS were developed, and designed to run on devices that are severely constrained in memory, power consumption, processing power, and communication bandwidth.

According to (Amjad et al., 2016), TinyOS is the most suitable OS to operate in a resource-starved network, such as WSN. It is an OS designed for low-power wireless embedded systems. Fundamentally, it is a work scheduler and a collection of drivers for microcontrollers commonly used in wireless embedded platforms. TinyOS programs are composed into event handlers and tasks with run-to-completion semantics.

Queiroz D., Gomes R. and Benavente-Peces C.
Performance Evaluation of Default Active Message Layer (AM) and TKN15.4 Protocol Stack in TinyOS 2.1.2.
DOI: 10.5220/0006204200690079
In *Proceedings of the 6th International Conference on Sensor Networks (SENSORNETS 2017)*, pages 69-79
ISBN: 421065/17

When an external event occurs, such as an incoming data packet or a sensor reading, TinyOS calls the appropriate event handler to deal with the event (Wang and Balasingham, 2010). Both the TinyOS system and programs developed for this OS are written in nesC, which is an extension of the C programming language.

As alternatives to TinyOS, stand out: Contiki, which is an open-source OS for the IoT, and connects tiny low-cost, low-power microcontrollers to the Internet; Berkeley OpenWSN, which is an open-source stack intending to implement low-power wireless standards such as IEEE 802.15.4e, 6LoWPAN, RPL and CoAP, and is rooted in the new IEEE802.15.4e TSCH; FreeRTOS, which is a popular Real Time Operating System (RTOS) that has been ported to many microcontrollers, and its preemptive microkernel has support for multi-threading with statically instantiated tasks; and RIOT, which is also a real time OS and was developed with focus on the requirements of IoT.

The latest version of TinyOS was released in 2012 (2.1.2), bringing support for updated msp430-gcc (4.6.3) and avr-gcc (4.1.2), and a complete 6lowpan/RPL IPv6 stack. Currently, TinyOS development was migrated to GitHub[1], where the researchers can contribute to its development. The OS includes TOSSIM, a high-fidelity mote simulator that compiles directly from nesC code, scaling to thousands of simulated nodes. TOSSIM gives the programmer an omniscient view of the network and greater debugging capabilities. Server-side applications can connect to a TOSSIM proxy just as if it were a real sensor network, easing the transition between the simulation environment and actual deployments (Levis, P. et al., 2009).

Different types of hardware platforms support TinyOS in the WSN domain. These different platforms introduce their own interrupts relating to their hardware designs (Hill, Jason, and David Culler, 2002). To port from one hardware platform to another, TinyOS developers have introduced a hardware abstraction architecture, and it can be classified into three layers (Amjad et al., 2016), (Handziski et al., 2005), as follows:

1. Hardware Interface Layer (HIL): comprises hardware-independent components, interfaces and events;
2. Hardware Presentation Layer (HPL): close to the hardware layer. The components of this layer are not picked by applications but are used by hardware in some particular tasks;

[1]https://github.com/tinyos - Access in 09/12/2016.

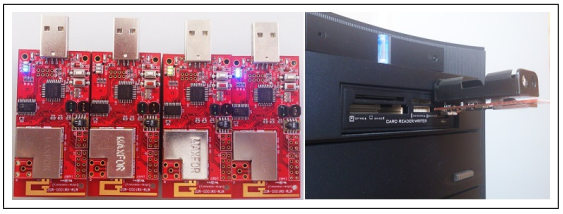

Figure 1: Model of testbed used in the performance evaluation, XM1000 (Advanticsys), and one coordinator connected to USB port.

3. Hardware Adaptation Layer (HAL): layer that favors hardware functionality, and is closer to the HPL.

Among the main platforms supported by TinyOS, are TelosB, Tmote Sky, MicaZ, Mica2, Zolertia Z1, GINA, IRIS, and XM1000, which is based on TelosB. In this paper, XM1000 platform is used to perform the experiments. In the experiments, seven motes were used, and five of them are depicted in Fig. 1.

The XM1000 is the new generation of mote modules, based on TelosB technical specifications, with upgraded 116Kb-EEPROM and 8Kb-RAM and integrated Temperature, Humidity and Light sensors. Besides TinyOS 2.1.2, it is also compatible with Contiki 2.7 (latest version of Contiki is 3.0[2]). Its processor belongs to Texas Instruments MSP430 family, the RF Chip is CC2420 and has range of around 120m (outdoor), and 20-30m (indoor), in which longer ranges are possible with optional SMA antenna attached. Its current consumption is of 18.8mA for RX, 17.4mA for TX, and 1uA when the device is in sleep mode.

The experiment described in this paper evaluates the performance of the default Active Message (AM) protocol of TinyOS and TKN15.4, and compares both in order to choose the best protocol suited for general-purpose applications, including industrial applications. For this, two example applications designed by TinyOS and TKN15.4 developers were used, in which one sends and receives packets using timers, and the other uses the beacon-enabled mode of IEEE 802.15.4 standard, respectively. Actually, in this work both applications were adapted to perform the same tasks. The only difference between them is that in TKN15.4, beacons were used to synchronize with its coordinator. The motivation for these tests is to see the feasibility of using those protocols for WSN in industrial environments/applications without stringent requirements on reliability and predictable real-time performance, as defined by (De Guglielmo et al., 2016). Examples of applications with stringent timing requirements are Military/Defense and Healthcare applications, and in some hazardous industrial

[2]Released in 26 Ago 2015.

environments, such as chemical/biotechnology.

It is worthy to note that the IEEE 802.15.4 MAC frame format is different from the frame format used in default AM TinyOS protocol. Therefore, there is an optional implementation called *TKN*154*ActiveMessageP.nc* that abstracts AM over the nonbeacon-enabled variant of IEEE 802.15.4 MAC. In this optional implementation, the upper layer in TinyOS will see the actual AM payload, and before passing frames down to the 15.4 MAC, the implementation makes sure that the AM type (and network byte) are moved to the MAC payload portion. This workaround involves extra *memmove* functions (C library function that copies *n* characters from *str*2 to *str*1, which is a safer approach than *memcpy* function for overlapping memory blocks). Since it requires more memory and processing than the protocols analyzed in this paper, it was not considered.

The remainder of this paper is structured as follows. In Section 2, AM and TKN15.4 protocols are introduced. Section 3 discusses the related works, highlighting those that used testbeds with TinyOS. Section 4 describes the measurement techniques, implementation details of the protocols, and the experimental methodology. The analysis of the results and the conclusions are given in Sections 5 and 6, respectively.

2 TinyOS PROTOCOLS

Sensor nodes are network-centric devices. Much of their software complexity comes from network protocols and their interactions. TinyOS provides a number of interfaces to abstract the underlying communications services and a number of components that provide these interfaces. All of these interfaces and components use a common message buffer abstraction, called "*message_t*", which is implemented as a nesC struct (similar to a C struct):

```
typedef nx_struct message_t {
  nx_uint8_t header[sizeof(message_header_t)];
  nx_uint8_t data[TOSH_DATA_LENGTH];
  nx_uint8_t footer[sizeof(message_footer_t)];
  nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

Some of the basic interfaces of TinyOS are described as follows (Developers, 2013):

- *Packet*: Provides the basic accessors for the *message_t* abstract data type;
- *Send*: Provides the basic address-free message sending interface;

DA	LSA	ML	GID	AM_TYPE	SM_ID	SC
FF FF	00 00	4	22	6	00 02	00 0B

Figure 2: Message format in AM Protocol.

- *Receive*: Provides the basic message reception interface;
- *PacketAcknowledgements*: Provides mechanism for requesting acknowledgements (ACK) on a per-packet basis;
- *RadioTimeStamping*: Provides time stamping information for radio transmission and reception.

Both AM and TKN15.4 protocols use some of these interfaces to send and receive packets, so the following subsections evaluate the two protocols of TinyOS.

2.1 Active Message Protocol

Since it is very common to have multiple services using the same radio to communicate, TinyOS provides the AM layer to multiplex access to the radio.

AM packets have a field called *AM_TYPE* (8 bits) which determines what the rest of the packet looks like. This term refers to the field used for multiplexing, and is similar in function to the Ethernet frame type field, IP protocol field, and UDP port in that all of them are used to multiplex access to a communication service (Developers, 2013). Fig. 2 depicts an example of the overall message format (ignoring the first 00 byte), and the definition of each field is as follows:

- DA: Destination Address (2 bytes);
- LSA: Link Source Address (2 bytes);
- ML: Message length (1 byte);
- GID: Group ID (1 byte);
- AM_TYPE: Active Message handler type (1 byte);
- PA: Payload (up to 28 bytes), in which ;
  SM_ID: Source Mote ID (2 bytes);
  SC: Sample Counter (2 bytes).

As AM services, TinyOS implements *AMPacket* and *AMSend*. The first one is similar to *Packet*, defined in Section 2, and provides the basic AM accessors for the *message_t* abstract data type. This interface provides commands for getting a node's AM address, and packet's destination/type. The second one is *AMSend*, which is similar to *Send*, and provides the basic AM sending interface. The key difference between *AMSend* and *Send* is that *AMSend* takes a destination AM address in its send command.

The AM accessors provide the functionality for querying packets. AM is a single-hop communication protocol, therefore, fields such as source and destination represent the single-hop source and destination. Multihop sources and destinations are defined by the corresponding multihop protocol (if any). Variants of the basic AM stack exist that incorporate lightweight, link-level security (Levis, P. et al., 2009).

Several components implement the communications and active message interfaces, as follows[3]:

- *AMReceiverC* - Provides the Receive, Packet, and AMPacket interfaces;
- *AMSenderC* - Provides AMSend, Packet, AMPacket, and PacketAcknowledgements as ACK;
- *AMSnooperC* - Provides Receive, Packet, and AMPacket;
- *AMSnoopingReceiverC* - Provides Receive, Packet, and AMPacket;
- *ActiveMessageAddressC* - Provides commands to get and set the node's active message address.

The basic components of programming a mote with AM are exemplified as follows, with methods to get the payload of a packet, to send and receive the messages:

```
uses interface Packet;
uses interface AMSend;
uses interface Receive;
...
BlinkToRadioMsg* btrpkt =
    (BlinkToRadioMsg*)(call Packet.getPayload(&pkt,
        sizeof(BlinkToRadioMsg)))
...
call AMSend.send(AM_BROADCAST_ADDR,
        &pkt, sizeof(BlinkToRadioMsg)) == SUCCESS
...
event message_t* Receive.receive(message_t* msg,
        void* payload, uint8_t len){...}
...
```

2.2 TKN15.4 Protocol

While the first steps in protocol design can often be made with the help of analytical models and simulation, the last steps require the use of real hardware, in realistic environmental conditions and experimental setups. TKN15.4 provides a stable, open-source 802.15.4 MAC platform independent implementation for the 2.1 and later releases of TinyOS (Hauer, 2009). It was developed by the Telecommunication Networks (TKN) Group from Technical University Berlin on March 2009.

[3]http://tinyos.stanford.edu/tinyos-wiki/ - Access 12/12/2016.

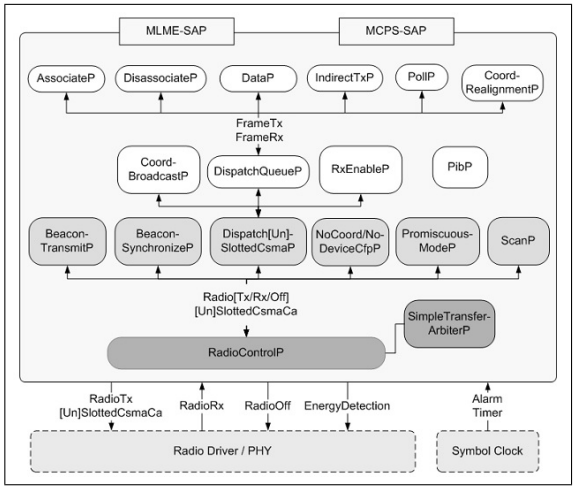

Figure 3: TKN15.4 architecture with components represented by rounded boxes, interfaces by connection lines. The radio driver and symbol clock components are external to TKN15.4.

2.2.1 Architecture

The architecture of TKN15.4 is depicted in Fig. 3. On the lowest level, the *RadioControlP* component manages the access to the radio: with the help of an extended TinyOS 2 arbiter component, it controls which of the components on the level above is allowed to access the radio at what point in time. Most components on the second level represent different parts of a superframe: the *BeaconTransmitP*/*BeaconSynchronizeP* components deals with beacons, *DispatchSlottedCsmaP* manages frame communication during the CAP, and *NoCoordCfpP*/*NoDeviceCfpP* components are responsible for the CFP. *ScanP* and *PromiscuousModeP* components provide services for channel scanning and promiscuous mode, respectively. Finally, the components on the top level implement the remaining MAC data and management services, for example, PAN association or requesting (polling) data from a coordinator. A component on this level typically provides a certain MAC MLME/MCPS primitive to the next higher layer, for example, *DataP* component provides the MCPS-DATA primitive to the next higher layer to send a frame to a peer device.

An example of programming a mote with TKN15.4 in a beacon-enabled mode implementation is as follows:

```
uses interface MCPS_DATA;
uses interface MLME_RESET;
uses interface MLME_GET;
uses interface MLME_SCAN;
uses interface MLME_SYNC;
```

```
uses interface MLME_BEACON_NOTIFY;
uses interface IEEE154Frame as Frame;
uses interface IEEE154BeaconFrame as BeaconFrame;
...
event void MLME_RESET.confirm(ieee154_status_t
    status){...}
...
event message_t* MLME_BEACON_NOTIFY.indication
    (message_t* frame){...}
...
event void MCPS_DATA.confirm{...}
```

2.2.2 TKN15.4 Applications

As default, in TKN15.4 implementation of TinyOS there are 11 examples of applications to help the developers to build their own applications; six applications for beacon-enabled mode networks (*TestAssociate, TestData, TestGTS, TestIndirect, TestMultihop and TestStartSync*), four for nonbeacon-enabled mode networks (*TestActiveScan, TestAssociate, TestIndirectData and TestPromiscuous*), and one application that works as packet sniffer.

In this work, the application *TestData* for beacon-enabled mode networks was adapted to the experiment, since it is a basic application example that implements direct transmissions of data from a device to the PAN coordinator, also called sink node. In this application, the coordinator transmits periodic beacons and waits for incoming DATA frames. The other nodes act as devices, and scan the pre-defined channel for beacons from the coordinator. Once they find a beacon, they try to synchronize to and track all future beacons. They then start to transmit DATA frames to the coordinator (direct transmission in the Contention Access Period - CAP).

As the IEEE 802.15.4 standard defines, the time interval between two beacon frames is called the Beacon Interval (BI), or Superframe, and is divided into an active period and an optional inactive period. During the inactive period, nodes can be kept in sleep mode to conserve their energy. The length of the active period is Superframe Duration (SD) and contains 16 equal length time slots (from 0 to 15). The 16 time slots in the superframe are subdivided into smaller slots known as the Backoff Period (BP). The active period comprises CAP and Contention Free Period (CFP). During CAP, nodes use the slotted CSMA/CA algorithm to access the channel. During CFP, up to seven Guaranteed Time Slots (GTS) can be allocated by the coordinator for each superframe, which allow the node to operate on the channel that is dedicated exclusively to it. A node with an assigned GTS has full access to the channel during its GTS. Nodes activity during it should be completed before the start

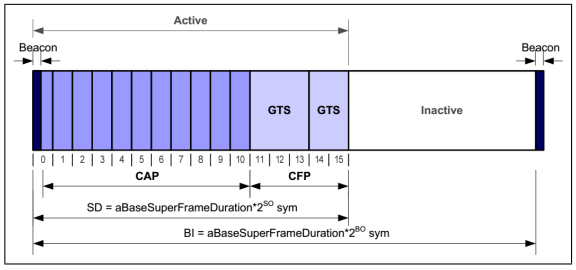

Figure 4: Superframe Structure with BO and SO parameters.

of the next GTS or the end of the CFP, as depicted in Fig. 4.

For mesh networks when using CSMA/CA transmissions and duty-cycles (fraction of time that the node is awake) lower than 100%, the delay in the communication will be mainly dependent on the network load (application traffic) and the number of hops to the sink node (Villaverde et al., 2010).

The duty-cycle (DC) is the ratio of the length of an active period SD to the length of a BI, and is calculated as $(\frac{1}{2})^{BO-SO}$. The default values defined by the standard for both parameters are $0 \leq SO \leq BO \leq 14$. The constant *aBaseSlotDuration* represents the number of symbols, and it has 60 symbols. Considering each symbol is 16μs in length for the 2.4-GHz band, it is possible to calculate the DC of the combination BO $\times$ SO. If BO = 5 and SO = 5 (TKN15.4 default values), the DC is calculated as follows:

- $aBaseSlotDuration$ = 60 symbols;
- $aBaseSuperframeDuration$ = 60 symbols $\times$ 16μs = 960μs;
- $BI = 960 \times 2^5$ symbols = 491520μs ($\approx$ 491ms);
- $SD = 960 \times 2^5$ symbols = 491520μs ($\approx$ 491ms);
- $DC = \frac{SD}{BI}$ = 1;

This implementation has a DC of 100% with active and inactive periods (SD) of 0.491s and BI of 0.491s. In this example, the PAN coordinator will generate around 2 beacons/second (1000/491BI $\approx$ 2). During each BI, the devices work for about 491ms (SD) and would keep quit for the rest of time, if there was any in this case. If SO was defined as 0, for example, it would minimize the ON time for the device during the CAP. If it were increased, it would allow more packets to be transferred during the CAP at the expense of higher power consumption

Due to non-deterministic nature of CSMA/CA transmissions, an exact calculation of delay cannot be provided nor can specific delay limitations be guaranteed (Carballido Villaverde et al., 2012). To guarantee specific delay requirements, the network parameters

cannot be selected randomly. Therefore, before comparing AM (does not use IEEE 802.15.4 MAC) and TKN15.4 protocols, this paper provides a delay analysis with regard to different MAC layer parameters so the best configuration of those can be selected.

Several works have been done in literature in order to study the behavior of the protocol when considering DC configuration, and how this configuration affects the performance of the network. Usually, the analysis is evaluated by using simulation tools; some of the works performed experiments using real devices, as in (Despaux et al., 2013). Since the parameters are dependent on the packet rate and the environment, in order to have more packets received and to be energy-efficient at the same time, this analysis is important to be done before each implementation.

3 RELATED WORK

The related works presented in this Section are classified based on experiments performed with one of the two protocols related in this paper.

In (dos Santos et al., 2014), the authors performed experiments with TinyOS using a MicaZ platform. They proposed a localized algorithm to enable detection, localization and extent determination of damage sites using the resource constrained environment of a WSN. The data collection stage starts at a given time, as requested by the sink node. A message is sent from the sink to the cluster managers, and those are responsible for sending messages to schedule the next sensing task on their subordinated sensors. For their paper, the default implementations of 802.15.4 protocol for lower level communication handling, and AM protocol for higher-level communication handling in TinyOS 2.1 were used. The reason for this choice was that they wanted a lean implementation of the whole system in their prototype.

In (Ouadjaout et al., 2014), the authors present a low cost and energy efficient wireless sensor mote platform for low data rate monitoring applications, called DZ50. This platform is based on ATmega328P micro-controller and RFM12b transceiver, and is compared with MicaZ and TelosB platforms. They ported all device drivers of DZ50 to TinyOS 2.x, which eases the programming of the platform and allows using many protocols already developed for TinyOS platform. As well as the previous paper, this one used AM to transmit and receive packets via an abstract interface.

In (Willig et al., 2010), the authors study passive discovery of IEEE 802.15.4 networks operating in beacon-enabled mode. To validate their analytical model, they performed experimental evaluations with an implementation on Tmote Sky using TKN15.4 protocol.

The work in (Carballido Villaverde et al., 2012) presents the InRout route selection algorithm, where local information is shared among neighboring nodes to enable efficient, distributed route selection while satisfying industrial application requirements and considering sensor node resource limitations. They used data frames sizes of 127 bytes, which is the maximum possible size in IEEE 802.15.4 networks. Since the sensor nodes have strict memory limitations, the buffer size at MAC layer for all nodes is restricted to 10 packets. A buffer size of 10 packets is chosen based on the default buffer size used in the IEEE 802.15.4 MAC standard implementation for TinyOS-2.x TKN15.4.

In (Macbeth and Sarrafzadeh, 2009), the authors evaluate the performance of a Listen-and-Suppress Carrier Sense Multiple Access (LAS-CSMA) scheme in order to reduce power consumption, network bandwidth usage and delays by suppressing node unnecessary packet transmissions. The scheme is evaluated with IRIS and MicaZ platforms, and AM protocol. The authors argue that AM allows for the overlap and integration of communication and computation, which is indispensable for efficient in-network data aggregation in sensor networks. In addition, it also allows multiple applications to use communication resources simultaneously.

In (Paczesny et al., 2012), it is presented the concept, design, and implementation of the proxy mote, a Linux-based TinyOS platform able to execute a TinyOS applications, called ProxyMotes. The main use case for the proxy mote is to expose a non-TinyOS (legacy) sensor/actuator device to TinyOS applications. To evaluate the proxy network, the authors used the Oscilloscope application a TinyOS demo, which generates AM packets to create the traffic.

The authors in (Dalton et al., 2009) presented a visualization toolkit for TinyOS 2.0 to aid in program comprehension. To make the concepts more concrete, they considered a variant of the Blink application included as part of TinyOS distribution, which uses AM protocol.

In (Shnayder et al., 2004), it is presented the PowerTOSSIM, a scalable simulation environment for WSN that provides an accurate, per-node estimate of power consumption. For the experiments, the authors used Mica2 sensor node and oscilloscope, and used AM protocol.

4 METHODOLOGY

In order to compare AM and TKN15.4 protocols' scalability, quantitative metrics are used to measure and evaluate the performance of both protocols. For all metrics, the average over multiple experiments is determined. The set of performance metrics used for comparing the selected protocols of this work can be described briefly as follows (Hac, 2003):

- *Packet generation rate*: it is the number of packets that the sensor node transmits in one period, which is usually one second.
- *Network throughput*: the end-to-end network throughput measures the number of packets per second received at the destination. It is considered here as an external measure of the effectiveness of a protocol;
- *Network delay*: it measures the average end-to-end delay of data packet transmission. This delay implies the average time taken between a packet initially sent by the source, and the time for successfully receiving the message at the destination. This measure takes into account the queuing and the propagation delay of the packets;
- *Success rate*: it is the total amount of packets received at the destinations verses the total number of packets sent from the source;
- *Energy consumption*: it is the energy consumed by a node in the network in which the periods of transmission, reception, and idling are taken into account. Assuming each transmission consumes an energy unit, the total consumption is equivalent to the total number of packets sent in the network. Note that there are many factors influencing the overall energy consumption, and the results presented in this paper should only be regarded as indicative for what is possible to achieve in systems with similar hardware.

In this work, seven nodes were used in a star topology, working on channel 26 (2.480GHz) and with -20dBm of transmission power. In TKN15.4, one node plays the coordinator role, and the others play the device role. In both protocols, the devices send two packets per second, and all the end nodes send packets at the same time in order to analyze the *Network throughput*, *Success rate*, and the *Network delay* metrics.

The sink node is connected to a USB port of the computer to forward the packet information (payload) to it, and the devices are set to send 2000 packets to their coordinators without ACK. It is important to notice that interference occurs in the laboratory, since the nodes are working on 2.4 GHz frequency, and the laboratory receives the signal of seven Wi-Fi networks.

For both protocols, the application tasks were the same, just changing the parameters of sending the packets. The TKN15.4 application in this work was developed with beacons enabled, so before sending the packets, the nodes need to synchronize with their coordinator. The beacon packets were not considered in the *Network throughput*, *Success rate*, and *Network delay* metrics, since AM does not use beacons to communicate with its sink node. However, the beacon packets were evaluated in the *Energy consumption* metric.

In order to measure the power consumed by the devices, a power supply and an oscilloscope were used, as depicted in Fig. 5, where a loop is used to measure the current and shows its shape in the oscilloscope.

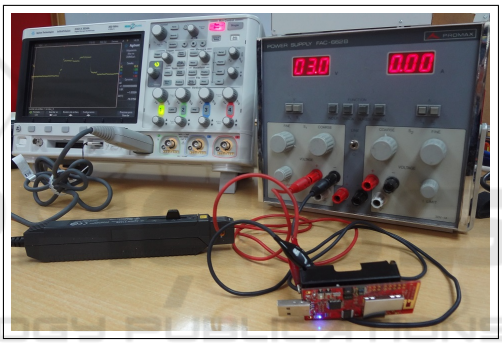

Figure 5: Measurement set-up: Power supply and oscilloscope connected to the transmitter.

The environment where the experiments were performed is as depicted in Fig. 6. It is a laboratory with around 43m^2 with some metal objects and people moving around. The nodes were put in random location, some of them with line of sight (LOS), and some without LOS. In Fig. 6, one sender and the receiver (coordinator) nodes are in the extremes right and left of the picture as an example without LOS. There were performed eight replications for each protocol.

Figure 6: Environment where the experiments were performed.

As stated before, in order to define the best parameters to achieve low power consumption for ZigBee

devices, there were stablished varying values for BO and SO, and the values adopted were used to compare TKN15.4 and AM protocols. For each value, four replications were performed, and the results are in Tab. 1.

Table 1: Percentage of packets received by changing BO and SO parameters.

SO \ BO	4	5	6	7
2	58	41	N	N
3	70	48	N	N
4	82	61	N	N
5	X	92	61	N
6	X	X	91	59
7	X	X	X	83

In this table, it is shown the percentage of packets received in different values of BO and SO. For each one, four replications were performed. The values of X were not used in the experiments because of the limitation in IEEE 802.15.4 standard, as discussed in Section 2.2.2 ($SO \leq BO$). Since the duty-cycle in N values of the table leads to a decrease in the number of packets received, as can be seen in $BO = 4/SO = 2$, $BO = 5/SO = 2$, and $BO = 5/SO = 3$, they were not analyzed in this work. Such values of packet reception are not acceptable. In this table, the best values of packet reception are using $BO = 5/SO = 5$ and $BO = 6/SO = 6$, and had almost the same results. Since in $BO = 6/SO = 6$, BI lasts more than in $BO = 5/SO = 5$, it spends more energy, so $BO = 5/SO = 5$ parameters were used to compare TKM15.4 and AM protocols.

5 RESULTS

Regarding the first metric to evaluate the performance of both protocols, *Packet generation rate*, this value was set to two packets per second. It takes 1000 seconds (nearly 17 minutes) to send 2000 packets. In TKN15.4, the beacon time synchronization was not considered, which lasts around 491ms for $BI = 5$, and around 983ms for $BI = 6$.

Each node is set to send 2 packets/s, i.e., the network sends 12 packets to the sink node. The sink node could not process correctly all packets that were received when the transmission rate was more than two packets per second for each of the six nodes, so this rate was adequate for the amount of packets to be processed and forwarded to the computer. With this configuration, the metric *Network Throughput* results in an average of 11 packets per second received/processed by the sink in AM protocol. This result can be explained because AM is a very simple protocol, and it cannot deal with so much packets at the same time. Only TRUE or FALSE attribute is used as congestion control method to determine if the sink is busy with processing of other packet. If so, the next packet will be discarded. Concerning TKN15.4, a device must sense an idle channel twice before it may transmit, as explained in Section 2.2.1, so the control is more organized and therefore the sink node will process more packets.

Regarding the *Network delay* metric, for each replication, each node was analyzed considering the time when it sends one packet, and the time when the packet is received by the sink. For all the packets received by the sink, for each sender, it was calculated the average time from sending to receiving tasks, and for AM protocol, the average value was 488.25ms. That is, the first packet from "Sender 1" was sent at 0ms; after 488.25ms, this packet was received by the sink; 11.75ms after that, the second packet was sent by "Sender 1", and then this packet was received by the sink at 988.25ms, and so on. Almost the same results were obtained by TKN15.4 protocol that lasted 488.2ms to send a packet, and this one to be received by the sink node.

Concerning the energy of TKN15.4, besides data packet transmitted, it must be considered the beacons received from the coordinator. Fig. 8 depicts one beacon received by the node. After receiving it, the node waits 491ms (BI) to receive the next one. The process of sending all packets lasts 1000000ms, and each packet is sent every 500ms, as depicted in Fig. 7. Without considering the first milliseconds of beacon packets for synchronizing, which is performed once, when the button is pushed to start, the node begins to send packets, and the time is started. After the first packet sent, the first beacon is received from the coordinator.

Depicted in Fig. 8, the energy consumption of one single beacon is shown, which results in about 0.023 Joules. Fig. 9 depicts the amount of energy of a single data packet, which consumes around 0.026 Joules. Depending on the platform, these results might be slightly different, even if the same application is used. In other periods than the packet interval, the node enters in a CPU Power-Save mode, therefore this time interval was not considered in the consumption, as it can be neglected compared to the main consumption. In 1000000ms, the node received around 2036 beacons, resulting in a total of 48 Joules of energy consumption for the beacon packets. The amount of data packets during all the experiment results in a consumption of 52 Joules. The result for TKN15.4 pack-

Table 2: Performance evaluation with metrics.

Metrics / Protocols	Packets sent	Packet Rate (pkts/s)	Network Throughput (pkts/s)	Network delay (ms)	Success Rate (%)	Energy Consumption (Joules)
AM	2000	2	11	488.25	90.7	486
TKN15.4	2000	2	12	488.2	94.8	100

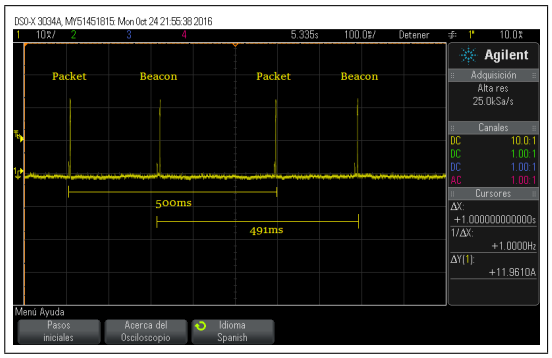

Figure 7: Beacon and Packet traces.

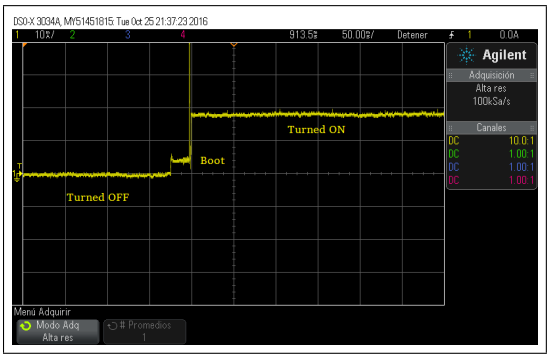

Figure 10: Restart of AM Protocol.

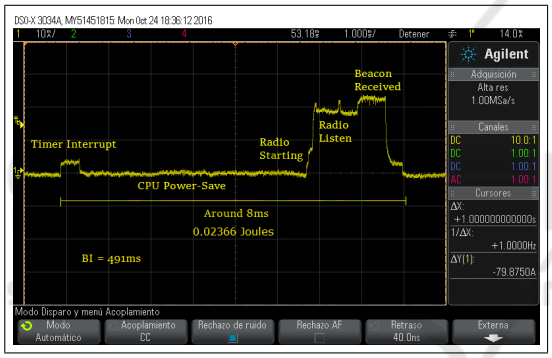

Figure 8: Characteristic of Beacon Packets.

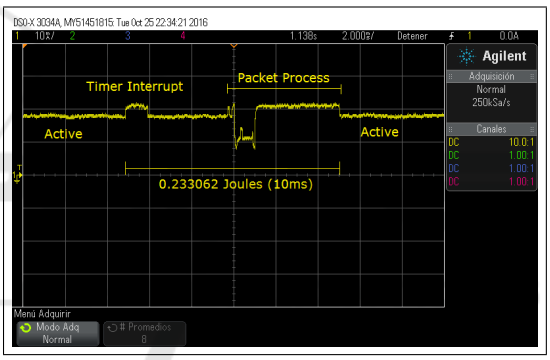

Figure 11: Characteristic of AM Packet.

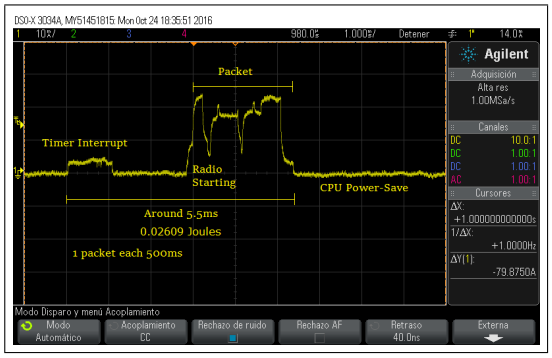

Figure 9: Characteristic of TKN15.4 packets.

ets is around 100 Joules, as illustrated in Tab. 2.

Regarding AM protocol, as seen in Fig. 10 and Fig. 11, it does not implement CPU Power-Save mode, i.e., the node remains active all the time, with around 18mA (54mW), even without receiving or transmitting any packet. Fig. 10 depicts the moment when the node is restarted, and Fig. 11 shows the packet transmission. The time from processing and sending the packet in AM protocol lasts around 10ms and consumes 0.233 Joules for each packet (466 Joules for 2000 packets, 20000ms). Considering that the node sends packet almost immediately after restarting, and the processing of packet lasts around 10ms, the period that the node does nothing in the interval between two packets is 490ms, resulting in 500ms for each packet sent. During the 2000 packets sent, the interval when the nodes does nothing (but remains in active) is 980000ms, which corresponds to around 20 Joules. Adding the consumption of active period, and packet processing, the total consumption is around 486 Joules, almost five times more than TKN15.4 during 16.7 minutes.

The most important differences of the measurement set-up regard the *Success rate* and *Energy consumption*. It is worthy to note that TKN15.4 protocol shows a higher success rate than AM protocol, and has much less energy consumption than AM Pro-

tocol. Therefore, AM protocol is not appropriate to monitor environments with restriction of energy and large amount of data.

6 CONCLUSIONS

This paper presented a performance evaluation of two protocols developed for TinyOS for communication in WSNs. AM protocol is simpler than TKN15.4 and allows multiple services using the same radio to communicate. TKN15.4 protocol uses the IEEE 802.15.4 standard and uses CSMA/CA to reduce collisions. It has several applications, and in the case of this paper, the application with beacon-enabled mode was used and compared to AM protocol regarding network throughput, network delay, success rate, and energy consumption.

Although simpler and allowing multiple services, AM has several drawbacks. The only congestion control method used is a variable that indicates if the sink node is busy. It also consumes much more energy, although in this work the duty-cycle used in TKN15.4 was 100%. TKN15.4 is better in energy consumption and success rate, since it uses CSMA/CA to control the access. These are two decisive parameters for choosing TKN15.4. Maybe with more nodes and during more time, the results would become even more differentiated. The positive point of AM is that it allows multiple services using the same radio, but causes an excessive energy consumption.

However, when considering stringent requirements on reliability and predictable real-time performance, TKN15.4 (IEEE 802.15.4) is not considered a good choice because of its several limitations, already highlighted by many studies, such as in (Anastasi et al., 2011). As main limitations of IEEE 802.15.4 (De Guglielmo et al., 2016), are inefficiency of slotted CSMA/CA in beacon-enabled mode, and in nonbeacon-enabled mode for a large number of nodes transmitting at the same time, and no protection against interference/fading.

Ongoing work extends this one to address the development of IEEE 802.15.4e standard in TinyOS, specifically the DSME (Deterministic and Synchronous Multi-Channel Extension) behavior mode. The choice for this mode is because there is already a small public project of IEEE 802.15.4e TSCH (Time-Slotted Channel Hopping) behavior mode in development called TKN-TSCH[4]. The implementation of DSME in TinyOS will improve the old standard of TKN15.4 by introducing mechanisms such as time slotted access, multichannel communications and channel hopping. Differently from the other behavior modes, DSME remains using the CAP and CFP methods of channel access derived from IEEE 802.15.4, which eases its implementation. There are other implementations of IEEE 802.15.4e (TSCH behavior mode) in OpenWSN[5] and Contiki[6].

[4]https://github.com/tinyos/tinyos-main/pull/361 - Access 13/12/2016.

ACKNOWLEDGEMENTS

The authors would like to thank the support of the Institute for Advanced Studies in Communications (Iecom), the Brazilian Council for Research and Development (CNPq), the Coordination for the Improvement of Higher Education Personnel (CAPES), and the SMART 2 Project of the Erasmus Mundus Programme.

REFERENCES

Agrawal, P., Ahlen, A., Olofsson, T., and Gidlund, M. (2014). Characterization of long term channel variations in industrial wireless sensor networks. In *IEEE International Conference on Communications*, pages 1–6.

Amjad, M., Sharif, M., Afzal, M. K., and Kim, S. W. (2016). TinyOS - New Trends, Comparative Views, and Supported Sensing Applications: A Review. *IEEE Sensors Journal*, 16(9):2865–2889.

Anastasi, G., Conti, M., and Di Francesco, M. (2011). A Comprehensive Analysis of the MAC Unreliability Problem in IEEE 802.15.4 Wireless Sensor Networks. *IEEE Transactions on Industrial Informatics*, 7(1):52–65.

Carballido Villaverde, B., Rea, S., and Pesch, D. (2012). InRout A QoS aware route selection algorithm for industrial wireless sensor networks. *Ad Hoc Networks*, 10(3):458–478.

Dalton, A. R., Wahba, S. K., Dandamudi, S., and Hallstrom, J. O. (2009). Visualizing the runtime behavior of embedded network systems: A toolkit for TinyOS. *Science of Computer Programming*, 74(7):446–469.

De Guglielmo, D., Brienza, S., and Anastasi, G. (2016). {IEEE} 802.15.4e: A survey. *Computer Communications*, 88:1–24.

Despaux, F., Song, Y.-Q., and Lahmadi, A. (2013). Measurement-based Analysis of the Effect of Duty Cycle in IEEE 802.15.4 MAC Performance. In *2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 620–626. IEEE.

Developers, T. (2013). Official tinyos documentation wiki. http://tinyos.stanford.edu/tinyos-wiki.

[5]http://openwsn.atlassian.net/wiki - Access 09/12/2016.

[6]https://github.com/contiki-os - Access 09/12/2016.

dos Santos, I. L., Pirmez, L., Lemos, É. T., Delicato, F. C., Vaz Pinto, L. A., de Souza, J. N., and Zomaya, A. Y. (2014). A localized algorithm for Structural Health Monitoring using wireless sensor networks. *Information Fusion*, 15:114–129.

Gomes, R. D. et al. (2016). Evaluation of link quality estimators for industrial wireless sensor networks. In *XXXIV Simposio Brasileiro de Telecomunicacoes e Processamento de Sinais*, pages 1–5.

Hac, A. (2003). *Wireless Sensor Network Designs*.

Handziski, V., Polastre, J., Hauer, J., Sharp, C., Wolisz, A., and Culler, D. (2005). Flexible hardware abstraction for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks.*, pages 145–157. IEEE.

Hauer, J.-H. (2009). Tkn15.4: An ieee 802.15.4 mac. implementation for tinyos 2. Technical report, Technical University Berlin.

Hill, Jason, and David Culler (2002). A wireless embedded sensor architecture for system-level optimization. Technical report, UC Berkeley.

Levis, P. et al. (2009). Tinyos: An operating system for sensor networks. In *Ambient Intelligence*, W. Weber, J. Rabaey, and E. Aarts (Eds.). Springer-Verlag.

Macbeth, J. and Sarrafzadeh, M. (2009). Press the Cancel Button! A Performance Evaluation of Scalable In-Network Data Aggregation. In *2009 International Conference on Information and Multimedia Technology*, pages 449–457. IEEE.

Ouadjaout, A., Lasla, N., Bagaa, M., Doudou, M., Zizoua, C., Kafi, M. A., Derhab, A., Djenouri, D., and Badache, N. (2014). DZ50: Energy-efficient Wireless Sensor Mote Platform for Low Data Rate Applications. *Procedia Computer Science*, 37:189–195.

Paczesny, T., Tajmajer, T., Domaszewicz, J., and Pruszkowski, A. (2012). ProxyMotes: Linux-based TinyOS Platform for Non-TinyOS Sensors and Actuators. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 255–261. IEEE.

Shnayder, V., Hempstead, M., Chen, B.-r., Allen, G. W., and Welsh, M. (2004). Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, page 188. ACM Press.

Villaverde, B. C., Alberola, R. D. P., Rea, S., and Pesch, D. (2010). Experimental Evaluation of Beacon Scheduling Mechanisms for Multihop IEEE 802.15.4 Wireless Sensor Networks. In *2010 Fourth International Conference on Sensor Technologies and Applications*, pages 226–231. IEEE.

Wang, Q. and Balasingham, I. (2010). Wireless Sensor Networks - An Introduction. In *Wireless Sensor Networks: Application-Centric Design*. InTech.

Willig, A., Karowski, N., and Hauer, J.-H. (2010). Passive discovery of IEEE 802.15.4-based body sensor networks. *Ad Hoc Networks*, 8(7):742–754.