A Model to Improve the Accuracy of WSN Simulations

Óscar Gama[1], Paulo Carvalho[1], P. M. Mendes[2]

[1]Informatics Dept., [2]Industrial Electronics Dept., University of Minho, Portugal
{osg, pmc}@di.uminho.pt, paulo.mendes@dei.uminho.pt

Abstract. Simulation studies have been extensively adopted in the networking research community. Nevertheless, the performance of the software components running within the network devices is often not modeled by generic network simulators. This aspect is particularly important in wireless sensor networks (WSN). As motes present very limited computing resources, the overhead of the software components cannot be ignored. Consequently, WSN simulation results may diverge significantly from the reality. After showing experimentally the validity of this assumption, the paper proposes a set of generic equations to model the performance of WSN software components. Validation tests using contention and multiplexing-based MAC protocols show that the inclusion of the proposed model in a WSN simulator improves the confidence degree in the simulation results significantly.

Keywords: wireless sensor network, simulation, real tests, parameterized model.

1 Introduction

Many studies within the WSN research community resort commonly to simulators. A review based on 151 wireless network articles from a five-year-period reported that 76% of those works used simulations [1]. The preference for simulators is justified by the difficulty of deploying real networks, as programming a large number of motes, gathering performance metrics, and managing the power sources is tedious and time consuming. Simulators allow building and modifying network scenarios easily, and tests are easily monitored. A comparison of simulators for WSNs is provided in [2].

WSN simulation studies use frequently unrealistic assumptions, such as, flat physical environment, circular radio transmission area, channel with bidirectional symmetry, and no fading or shadowing phenomena. These assumptions lead to simulation results that differ significantly from experimental results [3].

Since simulators can use different models to represent the same physical phenomenon, appreciable divergences in the results may be obtained using distinct simulators. The performance results of a simple algorithm using diverse simulators proved this fact [4]. Furthermore, models cannot represent reality with absolute accuracy [5]. Simulation scenarios can also ignore diverse hardware and software aspects that may influence the final results. An example is the time required by the base-station (BS) and the motes to process the incoming or outgoing packets.

Aware of the difficulty that a simulator may have in presenting accurate results, this work studies software-related aspects of a WSN that contribute to the differences between simulation results and real measurements. This important topic is usually neglected in WSN simulations. First, within the IEEE 802.15.4 domain, the results obtained in a simulated WSN are compared to those obtained in an analogous physical scenario, and the causes of divergence in the results are identified. Then, a model using empirical software-related parameters to improve the accuracy of the simulation results is proposed. Instead of trying to present accurate values for the model parameters, which are necessarily specific to each testbed, this work intends to model software-related issues which have influence on the testbed results, and which may also occur in another WSN testbed. The main contribution of this paper is to present a model reflecting the impact of the software components on a physical WSN. The proposed model is generic to be easily implemented in current WSN simulators, being also an important contribute for future development of simulation tools.

2 Experimental Platforms

The physical and simulated experimental platforms as well as the test conditions used in this work are presented next. The reference testbed is composed of 16 ZigBit-A2 motes [7] placed statically in a semi-circle around the BS, about one meter away from the BS. To evaluate the impact of software components on the performance of a WSN, a static small-area WSN was adopted to minimize the effects of additional source of errors, such as nodes mobility and fading phenomena. The testbed is limited to sixteen motes due to the RAM memory constraints of the BS. Indeed, a minimum amount of memory in the BS is required to hold data for packet statistical analysis, and this memory is dependent on the number of active motes in the WSN.

The ZigBit-A2 mote is an IEEE 802.15.4/ ZigBee-compliant module operating in the 2.4 GHz band. It contains one AT86RF230 transceiver and one ATmega1281V microcontroller. Motes run TinyOS, an event-oriented operating system. The testbed uses the BS available from the manufacturer. As the BS is built-in around a ZigBit-A2 module, in terms of software performance the BS is identical to a mote.

To study the validity of the proposed model in a different test scenario, traffic from another IEEE 802.15.4 WSN is admitted in the channel used by the reference testbed. The reference WSN and the interfering WSN have distinct personal area network identifiers, and are close enough to sense the carrier signals mutually.

The physical testbed scenario was equally implemented in the Castalia-2.3b [6], an open-source, discrete event-driven simulator designed specifically for WSNs.

2.1 Test Conditions

In the reference WSN, each mote transmits a packet with 90 bytes (B) of MAC payload to the BS every 250 ms approximately. This traffic volume is typical in WSNs with high data rates, such as e-health WSNs monitoring electrocardiographic signals. In these scenarios, the influence of the software components on the overall

network performance is not negligible. In the interfering WSN, one mote sends a packet with 100 B of MAC payload to its BS every 50 ms approximately.

The non-slotted CSMA-CA MAC protocol described in IEEE 802.15.4 standard was used in the reference WSN and interfering WSN. In the reference WSN, the CSMA-CA algorithm used the default parameters. The interfering WSN also used the default parameters apart from the maximum number of frame retries, which is zero to guarantee that the CSMA algorithm execution ends before 50 ms.

The reference WSN operated in a channel free of IEEE 802.15.4 traffic. For this purpose, a channel analyzer was used to find a free channel. To reduce the impact of spurious interferences, motes transmit at maximum power (3 dBm).

Tests were carried out in the physical testbed and in the simulator for an increasing number of motes in the WSN. The test duration was 16 minutes for each set of motes. Tests were run with and without 802.15.4 interfering traffic in the operating channel.

3 Experimental Results

The results for round-trip (RT) delay, and Delivery Error Ratio (DER) obtained both in the physical testbed and in the simulator are discussed next. Both metrics are considered from the perspective of the application layer. In the context of this study, RT delay is the time spent between sending an application data packet from a mote and the successful confirmation of the operation, which occurs after receiving the MAC ACK frame from the BS. DER expresses the probability of an application data packet sent from a mote to the application layer of the BS failing the delivery.

Fig. 1 and Fig. 2 present the results obtained *without the presence* of interfering traffic. Fig. 1 shows the simulation results for the DER when increasing the number of motes sending packets to the BS. The graphical bars correspond to the DER obtained in the physical testbed. For each number of active motes in the WSN, it is represented the maximum, average, and minimum DER values. Fig. 2 shows the maximum and average round-trip delays obtained in the simulator and in the physical testbed. In Fig. 1, while the simulation results reveal a WSN scaling up to 16 nodes with a maximum DER always below 1%, the physical testbed results show that above six active motes the maximum DER becomes higher than 1%. Fig. 2 reveals that the delays obtained in the simulator are significantly distinct from the real results.

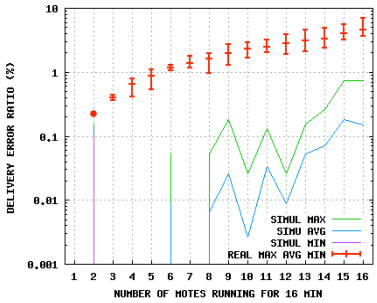

Fig. 1. DER without interferences.

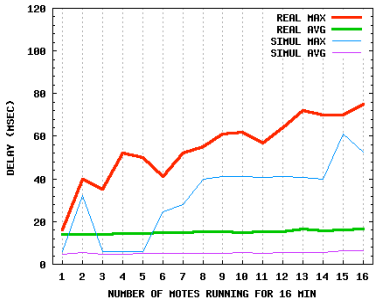

Fig. 2. RT delay without interferences.

Fig. 3 and Fig. 4 present the DER and round-trip delay results obtained *with the presence* of interfering traffic. The results shown in both figures were obtained in the simulator and in the physical testbed. As expected, the network performance degrades before the presence of interfering traffic. The differences in the results registered in the physical testbed and in the simulator are considerably distinct.

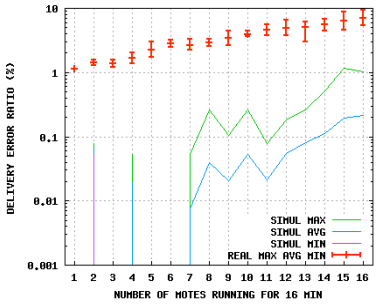

Fig. 3. DER with interferences.

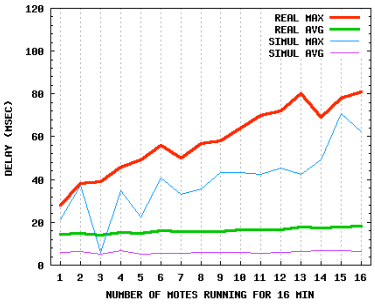

Fig. 4. RT delay with interferences.

4 Model Parameters

The main causes for the divergence of the results obtained in the simulator and in the physical testbed are identified and discussed next. As result of this analysis, a parameterized model is proposed for the simulator to minimize the differences to the testbed results. This aspect is of particular relevance to bring simulation scenarios close to real environments, increasing the meaningfulness of simulations results.

4.1 Software components

The first reason for the differences observed in the results is that the simulator does not take into consideration both the behavior of the operating system used in the network devices and the software processing time. As TinyOS can only schedule and handle single events and computing resources are very limited, significant delays may occur in scheduling and processing those events, as well as in processing the code of the protocol layers software. This overhead in terms of delay may be responsible for packet loss. To understand why, let us suppose that a packet has been received by the BS' transceiver. After processing it, the physical layer software triggers events to forward the payload to the upper protocol layers. Since the delivering time to the application layer is not null, another packet may be received by the BS' transceiver during this transactional phase. In this case, TinyOS does not attend the hardware interrupt from the transceiver indicating that a new packet is ready to be transferred to the microcontroller, and the new received packet is dropped. This situation was observed in the testbed. Yet, other operating system might attend the hardware interrupt from the transceiver indicating a new packet, and drop the packet in process previously received. In both cases, an incoming packet is completely processed by the

application layer only if a time interval elapses without other packet being received by the mote's transceiver. Next, it is proposed a model to reflect this behavior in a simulator. Its parameterized nature makes the model generic and independent of the type of operating system and hardware used in the WSN.

The model uses the delivery time parameters $T_{BS\ mac\rightarrow app}(n)$ and $T_{BS\ phy\rightarrow mac}(n)$. The parameter $T_{BS\ mac\rightarrow app}(n)$ indicates the time required by the BS to process the packet received from mote *n* at MAC layer and deliver the data to the application layer. Therefore, this parameter reflects both the event scheduling delay and the packet processing delay imposed by the link, network, and transport layers. The delivery time parameter $T_{BS\ phy\rightarrow mac}(n)$ reflects the time required by the BS to process the packet received from mote *n* at physical layer and deliver the payload to the MAC layer. Note that MAC layer tasks can be split between the transceiver and the microcontroller. In Zigbit motes, for example, address filtering, FCS check, and ACK transmission operations of a receiving MAC frame are carried out in the transceiver, but the MAC frame de-encapsulation and upper layer delivering are accomplished in the microcontroller. As software components timings are very hard to be measured directly in the transceiver's firmware, the parameters $T_{BS\ mac\rightarrow app}(n)$ and $T_{BS\ phy\rightarrow mac}(n)$ are measured relatively to the MAC layer component in the microcontroller.

The process time parameter $T_{BS\ app}(n)$ indicates the time required for the application layer of the BS to process the received payload from mote *n*. So, an incoming packet from mote *n* is completely processed by the application layer of the BS after a time interval $T_{BS\ totRX}(n)$:

$$T_{BS\ totRX}(n) = T_{BS\ phy\rightarrow mac}(n) + T_{BS\ mac\rightarrow app}(n) + T_{BS\ app}(n) . \quad (1)$$

The delivery time parameter $T_{BS\ phy\rightarrow mac}(n)$ includes the following partial times: i) the time required to receive the packet from mote *n*, $T_{RX}(n)$; ii) the packet processing time in the physical and MAC layers of the transceiver, $T_{BS\ phyRX}(n)$; iii) the time required by the microcontroller to read the bytes from the transceiver reception buffer through the peripheral communication interface, $T_{BS\ pciR}(n)$:

$$T_{BS\ phy\rightarrow mac}(n) = T_{RX}(n) + T_{BS\ phyRX}(n) + T_{BS\ pciR}(n) . \quad (2)$$

For a packet received from mote *n* with a physical header size PHY_h bytes, a MAC header plus trailer size MAC_h bytes, a MAC payload length $MAC_d(n)$ bytes, and a nominal transmission rate R bits/s:

$$T_{RX}(n) = (PHY_h+MAC_h+MAC_d(n)).8/R . \quad (3)$$

The parameter $T_{BS\ phyRX}(n)$ is very hard to be measured directly as it is related with the firmware performance of the transceiver. However, it can be obtained indirectly from the $T_{BS\ phy\rightarrow mac}(n)$ measurement, because $T_{RX}(n)$ and $T_{BS\ pciR}(n)$ are known.

Usually the peripheral communication interface between the microcontroller and the transceiver is a serial peripheral interface (SPI). In this case,

$$T_{pciR}(n) = (B_C+ MAC_h + MAC_d(n))*(8/S_{clk}+ T_{sep}) , \quad (4)$$

where B_C is the number of bytes of a read command, S_{clk} is the SPI clock frequency, T_{sep} is the separation time between the less significant bit of the last byte and the most significant bit of the next byte. For Zigbit motes, B_C=3 B, S_{clk}=4 MHz, T_{sep}=250 ns.

If $T_{BS\ phy \rightarrow mac}(n)$ is lower than the Long Inter-Frame Spacing (LIFS) period (or Short IFS, if the received MAC frame size ≤ maxSIFSFrameSize), then it takes the respective IFS value. For an IEEE 802.15.4 WSN at 250 kbps, SIFS is 0.192 ms and LIFS is 0.640 ms, at least; the maxSIFSFrameSize is 18 B.

Analogously, $T_{totTX}(n)$ is the total time required for mote *n* to complete the transmission process of an application data packet. Hence, the application packet delay comes increased by the sum of $T_{totRX}(n)$ and $T_{totTX}(n)$, where

$$T_{totTX}(n) = T_{app}(n) + T_{app \rightarrow mac}(n) + T_{mac \rightarrow phy}(n) + T_{conf}(n) \ . \quad (5)$$

$T_{conf}(n)$ is the time required for the application layer to obtain the confirmation of the transmission request success, as required in common MAC protocols (e.g., IEEE 802.15.4); $T_{app}(n)$ is the time needed for the application layer of the mote *n* to prepare the data payload; $T_{app \rightarrow mac}(n)$ is the time required by mote *n* to deliver the data payload to the MAC layer, and prepare the MAC frame; $T_{mac \rightarrow phy}(n)$ is the time required by mote *n* to deliver the MAC frame to the physical layer, prepare the packet and transmit it. This last parameter includes the following partial times: i) the time required by the microcontroller to write the bytes in the transceiver's transmission buffer and registers through the peripheral communication interface, $T_{pciW}(n)$; ii) the packet preparing time in the physical layer (and MAC layer, if present) of the transceiver, $T_{phyTX}(n)$; iii) the listen state to transmission state switching latency, $T_{l \rightarrow tx}(n)$; iv) the time required to transmit the packet, $T_{RX}(n)$, which is equal to $T_{TX}(n)$.

$$T_{mac \rightarrow phy}(n) = T_{pciW}(n) + T_{phyTX}(n) + T_{l \rightarrow tx}(n) + T_{TX}(n) \ . \quad (6)$$

The parameter $T_{phyTX}(n)$ is very hard to be measured directly because it is related with the firmware performance of the transceiver. However, it can be obtained indirectly from the $T_{mac \rightarrow phy}(n)$ measurement, because $T_{TX}(n)$, $T_{BS\ pciW}(n)$, and $T_{l \rightarrow tx}(n)$ are known. $T_{l \rightarrow tx}(n)$ is read from the transceiver technical specifications of mote *n*.

If the peripheral communication interface between the microcontroller and the transceiver is SPI, then $T_{pciW}(n)$ can be calculated using Eq. 4, being B_C the number of bytes of a write command. For Zigbit motes, B_C is 2 B and $T_{l \rightarrow tx}$ is 0.18 ms.

After sending a packet, a mote must wait $T_{nextTX}(n)$ before sending another packet, where:

$$T_{nextTX}(n) = T_{totTX}(n) - T_{TX}(n) \ . \quad (7)$$

This equation is important since it may limit the performance of mote *n* regarding data throughput or retransmission trials.

Let us consider now that the application timers of mote *a* and mote *b* fire respectively at time T(a) and time T(b) to send application data, and that T(b) > T(a). Also, let us assume that both motes use a MAC algorithm which does not perform any Clear Channel Assessment (CCA) to detect a clear channel or backoff contention procedures, i.e., once the timer fired, packets are directly sent to the wireless channel.

This is a usual procedure in TDMA-based MAC protocols. In this context, mote *a* ends transmitting the physical packet *a* into the wireless channel at time $T_{endTX}(a)$:

$$T_{endTX}(a) = T(a) + T_{totTX}(a) - T_{conf}(a) \ . \quad (8)$$

Mote *b* starts sending the physical packet *b* into the channel at time $T_{startTX}(b)$:

$$T_{startTX}(b) = T(b) + T_{totTX}(b) - T_{TX}(b) - T_{conf}(b) \ . \quad (9)$$

If $T_{startTX}(b) < T_{endTX}(a)$ then a packet collision occurs and both packets are lost. To avoid this situation, $T_{startTX}(b)$ must occur after $T_{endTX}(a)$, which means that the application timer of mote *b* must trigger after T(a) the following time:

$$T(b) - T(a) > \max\{0, T_{TX}(b) + (T_{totTX}(a) - T_{totTX}(b)) + (PT_{conf}(b) - PT_{conf}(a))\} \ . \quad (10)$$

In this case, if the condition

$$T_{startTX}(b) + T_{TX}(b) > T_{endTX}(a) + T_{BS\ totRX}(a) - T_{RX}(a) \quad (11)$$

holds ($T_{RX}(a)$ is subtracted because it is included in both $T_{endTX}(a)$ and $T_{BS\ totRX}(a)$), then mote *b* finishes the transmission after the BS having completely processed the packet *a*. In this case, the BS ends processing packet *b* at time $T_{BS\ end}(b)$, where:

$$T_{BS\ end}(b) = T_{startTX}(b) + T_{BS\ totRX}(b) \ . \quad (12)$$

However, if Eq.(11) is false, then mote *b* finishes the transmission while the BS is still processing the packet *a*, and consequently one of the packets is dropped (packet *b* in Zigbit motes). To guarantee that packet *b* is successfully processed by the BS, it must not collide with packet *a*, and it must be totally received after the BS finishes processing packet *a*. The first condition is expressed by Eq.(10). The second condition implies that T(b) must be incremented by $T_{BS\ totRX}(a) - T_{RX}(a)$. Additionally, if $T_{BS\ totRX}(a) - T_{RX}(a) > T_{TX}(b)$, then T(b) can be decremented by $T_{TX}(b)$, because the transceiver can receive packet *b* while the microcontroller processes packet *a*. So,

$$T_{endTX}(b) - T_{endTX}(a) > T_{BS\ totRX}(a) - T_{RX}(a) - (T_{BS\ totRX}(a) - T_{RX}(a) > T_{TX}(b) \ ? \ T_{TX}(b) : 0) \ . \quad (13)$$

Let us consider that mote *b* is ready to transfer data from the microcontroller to the transceiver, and mote *a* is transmitting to the BS. The transceiver of mote *b* must listen for packet *a* to read its physical and MAC headers. In Zigbit motes, it was observed that the transceiver of mote *b* can only accept data from the microcontroller after its radio circuit has finished the listening for the whole packet *a*. No channel collision occurs between packet *a* and packet *b*. This phenomenon imposes an additional delay, $T_{hdrD}(b,a)$, when sending a packet *b* to the channel due to the influence of packet *a*, where $T_{hdrD}(b,a) \leq T_{TX}(a)$. This delay must be added to $T_{totTX}(b)$, expressed in Eq.(5). As T(b) cannot occur before T(a), it results altogether that,

$$T(b) - T(a) > \max\{0, T_{TX}(b) + (T_{totTX}(a) - T_{totTX}(b)) + (PT_{conf}(b) - PT_{conf}(a)) + T_{BS\ totRX}(a) - T_{RX}(a) - (T_{BS\ totRX}(a) - T_{RX}(a) \ > T_{TX}(b) \ ? \ T_{TX}(b) : 0) - T_{hdrD}(b,a)\} \ . \quad (14)$$

If mote *a* and mote *b* are identical, run the same software, and send packets with the same size, then $T_{totTX}(a)=T_{totTX}(b)$, $PT_{conf}(b)=PT_{conf}(a)$, and Eq.(14) simplifies to,

$$T(b) - T(a) > T_{BS\ totRX}(a) - T_{TX}(a) - T_{hdrD}(b,a) \ . \quad (15)$$

In the ideal case of mote *a* and mote *b* presenting a null delay in all software components and sending equal size packets, Eq.(14) becomes $T(b) - T(a) > T_{TX}(a)$.

Let us assume now that mote *a* and mote *b* use a contention-based MAC protocol. Since random backoffs and CCA operations are carried out by the CSMA algorithm to find a clear channel, it is not possible to establish an equation relating T(b) with T(a). However, packet *a* and packet *b* are successfully processed by the BS only if the condition expressed in Eq.(13) holds.

Experimental tests with Zigbit motes revealed that the BS' transceiver is able to send a MAC ACK frame to a mote only if T_{ack} milliseconds have passed since the transmission of the MAC ACK frame of the last received packet.

MAC ACK frames sent by the BS' transceiver while the BS microcontroller is processing a received packet deteriorate the DER. To understand why, let us consider that the BS microcontroller is processing packet *a* when packet *b* is received by the BS' transceiver, and the respective MAC ACK frame arrives with success to mote *b*. As BS is processing packet *a*, packet *b* will be dropped. Since no retransmission will occur at mote *b*, packet *b* will not be delivered to the application layer of the BS. However, if the MAC ACK frame is not sent by the BS, packet *b* may be retransmitted and delivered successfully to the application layer of the BS, if meanwhile packet *a* has been completely processed.

4.2 Time drift

The second reason for the differences in the results is that the motes present an appreciable time drift. The cause of this time drift is distinct of the CPU clock time drift, which is typically a few microseconds per second. While the latter is due to physical characteristics of the semiconductor components, the former is mainly due to the CPU internal software performance running under limited computing resources. To reflect this feature, the drift parameter D_{ab} was introduced in the simulator. To set this parameter correctly, measurements were carried out using the BS and pairs of motes. Generically, if the drift between mote *a* and the BS is D_a, and the drift between mote *b* and the BS is D_b, then the drift between mote *a* and mote *b* is $D_{ab}= D_a - D_b$. This means that if mote *a* and mote *b* start transmitting separated in time by T_{ab}, and if $D_a > D_b$, then both motes will contend for the wireless channel after sending T_{ab} / D_{ab} packets. The D_{ab} value can be calculated experimentally through the relation:

$$D_{ab} = ((Ta_{i+1} - Tb_{i+1}) - (Ta_i - Tb_i)) / (Tb_{i+1} - Tb_i) \ , \quad (16)$$

where Ta_i, Ta_{i+1}, Tb_i, and Tb_{i+1} express the local time of the BS when packet *i* and packet *i+1* are received from mote *a* and mote *b*, respectively. It is assumed that packet *i* from mote *b* arrives after packet *i* from mote *a*, as well as all successive

received packets from both motes during the period Tb_i and Tb_{i+1}. Since $T_{ab} < 125$ ms in the physical testbed, and assuming $D_{ab} = 0.1\%$, channel contentions between a pair of motes may occur whenever 125 packets are sent at maximum. However, no channel contention occurs if D_{ab} is zero and T_{ab} is above the full-loaded packet transmission time. In this situation, the simulator results presented a null DER in a WSN with more than sixteen active motes. To prevent this unrealistic situation, the simulation results in Figs. 1, 2, 3, 4 were taken using a D_{ab} equal to 0.005%.

4.3 Setting of the model parameters

Whenever possible, the tuning of the model parameters was accomplished from measurements performed in the physical testbed. Table 1 presents the values found for the defined parameters, expressed in milliseconds, which are specific to this physical testbed. MAC payloads of 30 B and 90 B were considered. These values were measured on an analogical oscilloscope, and may present an error of +/– 0.5 ms. The values in *italic* were calculated analytically: $T_{BS\ phyRX}$ derives from Eq.(2); T_{RX} and T_{TX} from Eq.(3), $T_{BS\ pciR}$ and T_{pciW} from Eq.(4), T_{phyTX} from Eq.(6); $T_{l\rightarrow tx}$ was obtained from the transceiver technical specifications. Recall that the IEEE 802.15.4 protocol stack is implemented in the firmware of the nodes' transceiver.

Table 1. Values of the model parameters for: the BS (left) and the motes (right).

MAC_d	**30 B**	**90 B**
$T_{BS\ app}$	**1.8** ms	**1.8** ms
$T_{BS\ mac\rightarrow app}$	**1.0**	**1.3**
$T_{BS\ phy\rightarrow mac}$	**1.0**+T_{RX}	**1.4**+T_{RX}
T_{ack}	**3.3**	**3.7**
$T_{BS\ pciR}$	*0.10*	*0.23*
$T_{BS\ phyRX}$	*0.90*	*1.17*
T_{RX}, T_{TX}	*1.50*	*3.42*
$T_{BS\ totRX}$	3.8+T_{RX}	4.5+T_{RX}

MAC_d	**30 B**	**90 B**
T_{app}	**1.8** ms	**2.0** ms
$T_{app\rightarrow mac}$	**1.2**	**2.0**
$T_{mac\rightarrow phy}$	**1.4**+T_{TX}	**2.5**+T_{TX}
T_{conf}	**4.0**	**4.0**
T_{pciW}	*0.10*	*0.23*
T_{phyTX}	*1.12*	*2.09*
$T_{l\rightarrow tx}$	*0.18*	*0.18*
T_{totTX}	**8.4**+T_{TX}	**10.5+**T_{TX}

Measurements showed that the software time drift between motes may have values up to 0.3%, depending on the pair of motes used. The time drift between a pair of motes varies along the time. The simulator was programmed so that each mote at start-up chooses an average time drift D_{ab} up to 0.3% randomly.

The computing performance of the BS in the physical testbed is similar to a mote. This situation is not normally found in a WSN since a BS presents typically stronger computing resources and a more efficient operating system than motes. In this case, the value of T_{totRX} and T_{totTX} may be negligible. However, in a multi-hop WSN the packets may be routed through the motes, and so the value of these parameters can influence significantly the network performance.

5 Simulation results with the model

In order to validate the proposed model, tests were carried out in the physical and simulation platforms using both TDMA and CSMA-based MAC protocols. The motes used in the experiments are identical in terms of hardware, and run the same software.

5.1 TDMA algorithm

Validation tests of the proposed model were carried out in the physical and simulation platforms using a simple TDMA-based algorithm. The BS sends a beacon every 100 ms. This value was chosen to minimize the effect of the time drift D_{ab}. In each superframe, two or three motes transmit once with the minimum time gap that guarantees a null DER. Table 2 compares the values obtained in both platforms. Simulation tests were accomplished with and without the proposed model implemented in the simulator. As illustrated, the inclusion of the proposed model in the simulator, brings the simulation outcome close to the real results, with differences below 0.5 ms. The registered differences are justified taking into account the accuracy error that affects the measured values. T_{hdrD} presented a null value in all tests, excepting the test marked with an asterisk, where T_{hdrD} was 1.0 ms.

An important conclusion taken from the real results is that slots should be allocated to the motes in accordance with the respective packet sizes to be transmitted. Whenever possible, smaller packets should be sent first, otherwise bandwidth waste occurs. This is shown in Table 2 when mote *a* sends 90 B and mote *b* sends 30 B.

Table 2. Results from the physical and simulation (with and without the model) testbeds.

mote *a*	mote *b*	mote *c*	Real	Simul. w/ model	Simul. w/o model
30 B	30 B	30 B	**4.0**	3.8	1.5
30 B	30 B	-	**4.0**	3.8	1.5
90 B	90 B	90 B	**4.5**	4.5	3.4
90 B*	90 B*	-	**3.0**	3.5	3.4
30 B	90 B	-	**0.5**	0.0	1.5
90 B	30 B	-	**8.5**	8.5	3.4

5.2 CSMA algorithm

Validation tests of the proposed model were also carried out in the physical and simulation platforms using the IEEE 802.15.4 MAC protocol. Fig. 5 and Fig. 6 show the simulation results using the proposed model when IEEE 802.15.4 interfering traffic was not present. It is observed that the DER simulation results approximate closely to the DER values found in the physical scenario (the corresponding physical testbed results are also replicated for better comparison). The results of the maximum and average delays also become close to those obtained in the physical scenario.

Simulations without using the proposed model showed that the average DER *improves* over 75% when the MAC payload decreases from 90 B to 30 B. As the channel occupation decreases, the number of collisions diminishes, and so the DER

improves. However, tests on the physical platform revealed that the average DER *degrades* about 20% when the MAC payload decreases from 90 B to 30 B. The same degradation was observed in the simulations with the proposed model, confirming the validity of the model. As the packet size decreases, the probability of having multiple packets arriving without collisions to the BS during T_{totRX} becomes higher, and consequently the DER increases too. Fig. 7 and Fig. 8 present the simulation results when IEEE 802.15.4 interfering traffic was present. The DER results keep close to the DER values found in the physical scenario. The results of the average and maximum delays are also identical to those obtained in the physical scenario.

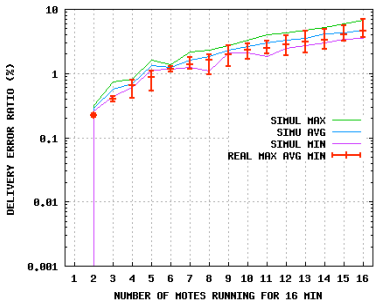

Fig. 5. DER without interferences.

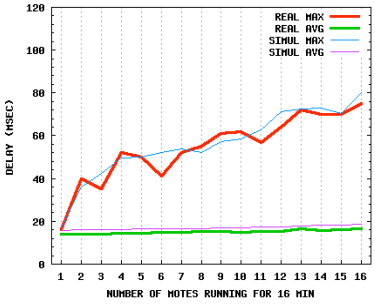

Fig. 6. Delay without interferences.

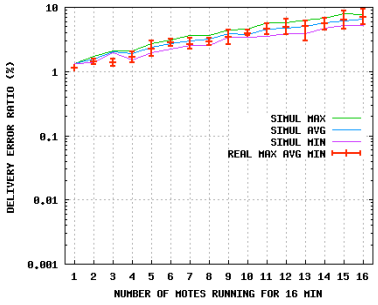

Fig. 7. DER with interferences.

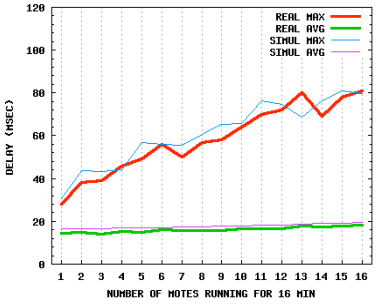

Fig. 8. Delay with interferences.

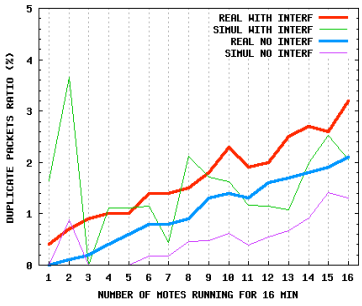

Fig. 9. <DPR> without the model.

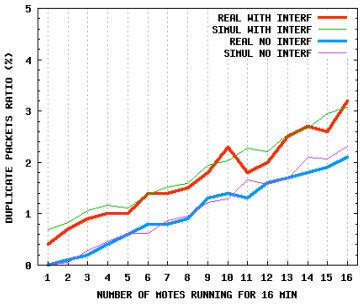

Fig. 10. <DPR> with the proposed model.

With the CSMA-CA algorithm, a mote may send a duplicate packet if it does not receive the MAC ACK frame from the BS. The average Duplicate Packets Ratio (<DPR>) is defined as the percentage of the total number of data packets received in duplicate by the application layer of the BS comparatively to the number of application data packets received for the first time from all motes in the WSN.

Fig. 9 shows the <DPR> obtained with and without the presence of IEEE 802.15.4 interfering traffic, not using the proposed parameterized model. Fig. 10 presents de <DPR> using this model. In this latter case, the simulation results are very identical to those obtained in the physical testbed.

6 Conclusions

Since motes present typically very limited computing resources, the performance of the operating system and high-level software running inside the motes impose significant constrains to the overall performance of a WSN. This paper showed that if the limitations of the software components are not considered, the simulation tests may produce results significantly more optimistic than those obtained in real conditions. Indeed, tests showed that it is difficult to obtain satisfactory simulation results using only the parameters of the wireless channel, the physical layer, and the MAC layer provided by the WSN simulator. This important aspect is often neglected in many works presenting WSN evaluation studies carried out on simulators.

In order to obtain satisfactory simulation results, a parameterized model was proposed, tuned, and included in the simulator. Simulation tests showed that the results obtained with the proposed model match satisfactory to those obtained in real conditions. Therefore, the inclusion of this model in a WSN simulator helps to improve the confidence on the simulation results. The model is generic enough to be also included in simulators running network scenarios other than WSNs.

References

1. Kurkowski, S., Camp, T., Colagrosso, M.: Manet Simulation Studies: the Incredibles. ACM Mobile Computing and Communications Review, vol. 9, no. 4, pp. 50–61, (2005)
2. Singh, C. P., Vyas, O. P., Tiwari, M. K.: A Survey of Simulation in Sensor Networks. IEEE, Computational Intelligence For Modelling Control & Automation, Washington, (2008)
3. Kotz, D., Newport, C., Gray, R. S., Liu, J., Yuan, Y., Elliott, C.: Experimental Evaluation of Wireless Wimulation Assumptions. Intern. Conf. on Modeling, Analysis and Simulation of Wireless & Mobile Systems, ACM, New York, USA, (2004).
4. Cavin, D., Sasson, Y., Schiper, A.: On the Accuracy of Manet Simulators. Principles of Mobile Computing, ACM, pp. 38–43, New York, USA, (2002)
5. Banks, J., Carson, J., Nelson, B.: Discrete-Event System Simulation. Prentice Hall, (1996)
6. Castalia: A Simulator for WSN, http://castalia.npc.nicta.com.au. (accessed in March 2011)
7. ZigBit Modules, http://www.atmel.com/dyn/resources/prod_documents/doc8226.pdf (*idem*)