On the Accuracy of OMNeT++ in the Wireless Sensor Networks Domain: Simulation vs. Testbed

Ugo Maria Colesanti
Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma
Via Ariosto 25
00185, Roma, Italia
colesanti@dis.uniroma1.it

Carlo Crociani
Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma
Via Ariosto 25
00185, Roma, Italia
crocicarlo@email.it

Andrea Vitaletti
Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma
Via Ariosto 25
00185, Roma, Italia
vitale@dis.uniroma1.it

ABSTRACT

In this paper we present a first effort in assessing the reliability of OMNeT++ and the MAC Simulator framework in simulating Wireless Sensor Networks. A collection of metrics on the flooding algorithm running on a simple testbed made of few Tmote Sky is used as reference to evaluate the quality of the simulation results. Our experiments show that simulation results tend to over-estimate the metrics collected in the testbed. A correcting factor derived from experimental evidences must be considered in order to improve the simulation results. At the best of our knowledge, this is the first result about the accuracy of OMNet++ in the wireless sensor network domain.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; B.8.0 [**Performance and Reliability**]: General

General Terms

Experimentation, Measurement, Performance, Reliability

Keywords

Wireless Sensor Networks, OMNeT++

1. INTRODUCTION

Wireless sensor networks (WSN) combine simple wireless communication, minimal computation facilities and the ability of sensing the physical environment. A WSN is made of a large number of sensor nodes that are densely deployed in the proximity of the monitored phenomenon. The number of sensor nodes may range from hundreds to thousands or even millions and the density of nodes can range from few sensor nodes to few hundred sensor nodes in a region [3]. There are two main approaches for validating large-scale sensor networks: testbeds and simulations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PE-WASUN'07, October 22, 2007, Chania, Crete Island, Greece.
Copyright 2007 ACM 978-1-59593-808-4/07/0010 ...$5.00.

Testbeds. Many of the logistical challenges related to the development, deployment, and debugging of realistic large-scale sensor networks have gone unmet. Manually reprogramming nodes, deploying them into the physical environment, and instrumenting them for data gathering is tedious and time-consuming. Furthermore WSNs face many problems that do not arise so acutely in other types of networks. First of all WSNs are strongly power constrained; sensor nodes are battery powered, and battery replacement is either uneconomic or unfeasible in most of the envisaged scenarios. The tiny sensor nodes are made of low-cost hardware and are thus fragile and prone to failures and since these devices can be deployed and operate in hostile environments, unexpected node failures is a likely event. Finally WSN programming is prone to bugs that typically arise in distributed, embedded and wireless systems. In most cases, bugs are hard to detect because of the lack of visibility into the nodes: the limited communication and computational resources prevent nodes from freely storing and transmitting debugging information, as this quickly depletes energy and network lifetime. As a result, once a sensor network is deployed, visibility into the network drops dramatically. Altough there is an effort in designing efficient debugging tools there is still a dearth of them [17].

A number of sensor node testbeds exist for supporting network and middleware research efforts, but only few of these support generic testability goals for a broad set of users [21, 2, 6]. Nevertheless all the testbeds share common features: the number of nodes involved is at most hundreds, but typical testbeds are made only of tens of nodes; nodes are deployed in a static grid topology (Kansey [6] offers also portable and mobile arrays even if an order of magnitude smaller in size of the stationary array); metrics and debug information are typically obtained through wired connections. This implies that testbeds impose strong constraints on the network both in terms of topology and size. Furthermore the cost of running an experiment on a testbed, in terms of setting-up the experiments, instrumenting the nodes, gathering the metrics on the performance, etc. is much higher than on a simulation and thus the simulation remains the most practical tool to obtain a feedback on the performance of a new solution.

Summarizing, over last few years the WSN research community has started to move beyond simulations as the only evaluation tool, but testbeds are not yet practical for large-scale wireless sensor network experiments.

Simulations. Due to the complexity and difficulty to implement real testbeds for ad hoc experiments, simulation tools are widely used. In particular simulations allow researchers to validate the WSN before the deployment, so that any further corrective action once the network is actually operating can be reduced, and enable the large scale experimentation of protocols and applications in a flexible environment. This is confirmed by the fact that in the vast majority of the papers on algorithms and protocols for wireless sensor networks, the performance evaluation is based mainly on simulation results; here we mention only few of them [14, 9, 20, 12, 8].

OMNeT++ [19] is a public-source, component-based, modular and open-architecture simulation environment with GUI support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks. OMNeT++ has been designed with two main objectives in mind [13]: *i)* to reduce the complexity ("*...unnecessary interdependency between modules*") of the widely used ns2 simulator, and its extremely steep learning curve, making easier the development and test of new protocols, *ii)* to improve the efficiency of the simulation ("*(OMNeT++)... executes at least an order of magnitude faster than ns2 while using memory more efficiently*").
The ease of modifying the sensor network properties and its scalability (i.e. the number of nodes that can be simulated) makes OMNeT++ an excellent tool for the simulation of WSNs. The success of this approach is proved by many papers (see e.g. [20, 12, 8]). In [20] the authors present simulation results on a scalable algorithm for estimating the localization of nodes. The problem of determining the node locations in ad-hoc sensor networks is also studied in [12]. The authors compare three distributed localization algorithms (Ad-hoc positioning, Robust positioning, and N-hop multilateration) on a single simulation platform. In [8] the authors present a study on the energy saved by S-MAC and T-MAC protocols that are optimized for wireless sensor networks, with respect to standard CSMA/CA. The comparison is based on extensive simulation driven by traffic that varies over time and location.

A natural question arises. What is the reliability of OMNeT++? To answer this question, we built very simple testbeds made of few TMote Sky sensor nodes densely deployed running the flooding protocol. Metrics on the performance of these testbeds were collected and compared to the metrics obtained by simulations of the same scenarios. At the best of our knowledge, this is the first result about the accuracy of OMNet++ in the wireless sensor network domain.
Contribution of the paper. We present a first effort to evaluate the reliability of OMNeT++. We consider a very simple experimental set-up made of up to six Tmote Sky sensor nodes and we test on it the performance of the flooding algorithm. The results of the testbed are compared with the results of the simulations of the same scenarios on OMNeT++. We chose to consider a simple experimental set-up in order to have a clear and manageable environment in which the behavior of the network is not affected by complex technical details. The rationale behind this choice is that, if a distance between simulation and testbed emerges in a very simple and clear environment, this distance would be further extended by a more complex one.

We observe significant differences between the experimental and the simulated results; the simulation always show better performance. In particular we show that high density of nodes and number of data sources strongly influences the performance of the flooding protocol in practice and this is not captured by simulations.

The reasons of such differences are analyzed and key parameters are determined and quantified. Finally, in order to assess and to confirm our studies we simulate again the flooding algorithm tuned in order to take into account above issues. We remark that such parameters are usually disregarded in simulations [11]. These new simulations give results that are closed to the experimental data.

Therefore the main contribution of this paper is twofold. First of all we provide a quantitative assessment of the role of parameters that affect the performance of WSN. At application layer we considered the effects of the asynchronous events generated both by sending and receiving messages. At link and physical layers we considered the effect of radio and MAC timings and environmental noises. We remark that OMNeT++ "as it is" only consider collisions and the effects of the distance in the propagation model.

The second contribution is methodological. We observe that quantifying and assessing all possible parameters that influence the performance in a complex practical scenarios is a formidable task that might give rise to models with many (possibly interacting) parameters. Quantifying and assessing such parameters is beyond the current state of the art ; therefore simulations will most likely give rise to unrealistic evaluations.

Here we propose to consider simplified scenarios that allow us to distill and to assess the role of specific parameters. We remark that the above simplified scenario allows us to tune the simulations and to provide performance close to the real ones and we believe that this effort will be a valuable first step in understanding how to tune the parameters of more complex scenarios.

1.1 Related Work

Most similar to our work in goals is [10]. In this paper is evaluated the accuracy of the widely used ns2 simulator in the WiFi domain. The authors present the validation of a wireless network model built with ns2 done by comparing the network characteristics of a simulated, an emulated, and a real 802.11 wireless network. The results show that the packet delivery ratios, the connectivity graphs, and the packet latencies are represented in the model with an average error of 0.3%, 10%, and 57% respectively. Based on their results the authors provide recommendations for future development of the ns2. We observed that some relevant parameters are usually disregarded in the simulation tools. In [11] the authors provide a comprehensive review of six assumptions that are part of many ad hoc network simulation studies: 1) The world is flat; 2) A radio's transmission area is circular; 3) All radios have equal range; 4) If I can hear you, you can hear me (symmetry); 5) If I can hear you at all, I can hear you perfectly; 6) Signal strength is a simple function of distance. An extensive set of measurements from a large outdoor routing experiment demonstrates the weakness of these assumptions, and shows how these assumptions cause simulation results to differ significantly from experimental results.

Simulators are models of the real world, and thus different models of the same phenomenon (i.e. different simulators) may produce very significant differences in the simulation results. This is confirmed in [4] where the authors present the simulation results of a straightforward algorithm using several popular simulators. The results tend to show that significant divergences exist between the simulators. This can be explained partly by the mismatching of the modelisation of each simulator and also by the different levels of detail provided to implement and conn the simulated scenarios. An extensive comparison of the simulators for WSN from a functionalities point of view is provided in [5] to aid developers in the selection of an appropriate simulation tool.

Simulators are invaluable for rapidly testing new ideas, but their results usually cannot be blindly trusted when migrating to real applications. Over last few years, the WSN research community has started to move beyond simulations as the only evaluation tool and a number of sensor node testbeds have been developed, but only few of these support generic testability goals for a broad set of users [21, 2, 6]. MoteLab [21] is a Web-based sensor network testbed and consists of a set of permanently deployed sensor network nodes (30 MicaZ) connected to a central server which handles reprogramming and data logging while providing a web interface for creating and scheduling jobs on the testbed. MistLab from MIT [2] consists of a mixture of 47 mica2 nodes and 14 Cricket nodes spread across multiple rooms located on the 9th floor of MIT's CS department. Moreover at the moment only the Kansei testbed [6] at The Ohio State University is made of hundreds of nodes. In particular Kansei is made of 210 Extreme Scale Motes (XSM) hooked individually onto 210 Extreme Scale Stargates (XSS). The stargates are connected using both wired and wireless ethernet. Kansei provides a testbed infrastructure to conduct experiments with 802.11b networking and XSMs and exports a web interface on which experiments can be scheduled and the results retrieved.

2. EXPERIMENT SET-UP AND METRICS

Sensor networks are typically densely deployed [3], for this reason we placed the sensor nodes on the plane at an average distance of only 2 meters. The resulting topology is always a clique, namely there exists a connection between every two nodes in the network. One or more nodes are *generators*. Generators sample the environment at a given *sampling frequency* and communicate the acquired data to the rest of the network by means of the *flooding protocol*.

2.1 Design principle

We designed our experiments to be as simple and clear as possible. The rationale behind this choice is that, if a distance between simulation and testbed emerges in a very simple and clear environment, this distance would be further extended by a more complex one. Following this principle, we considered a limited number of nodes connected in a dense topology (i.e. a clique), and running the basic flooding protocol (see Algorithm 1) instead of a more complex routing algorithm. In this way we hope to limit the effect of the environmental parameters and do not affect the results of the experiments by the effects of complex technical details.

2.2 Scenarios

In our experiments the sampling interval I_{samp} is fixed to 5, 50 and 500ms. The number of generators $G = 1$ or 2 and the total number of nodes is less than 6. Each generator reads the built-in photodiode every I_{samp}ms and floods the network with the data on the luminosity. The experiment last at least $1'30''$. The maximum number of packets generated during the experiment is limited to 18000 for $I_{samp} = 5$ms and to 1800 for 50 and 500ms. The nodes form a clique on the same plane and the average distance between the nodes is 2 meters. Our experiments are designed to be representative of networks with *high density* and a traffic load that is only function of the monitored data. For example when $I_{samp} = 5$ms the bit-rate of a node should be about 60Kbps (the packet size is 37 bytes); we will see that the actual bit-rate is less than 30Kbps.

Algorithm 1 Flooding algorithm

```
 1: repeat
 2:
 3:     if (the node is a generator) then
 4:         data ← ADC.data()        ▷ Get data from sensor
 5:         msg ← data
 6:         bcast(msg)
 7:         num_msg++
 8:         wait($I_{samp}$)
 9:     end if
10:
11: until ($num\_msg == MAX\_MSG$)
12:
13: receive(msg)                          ▷ All nodes
14: if (¬ Received(msg)) then             ▷ Received a new msg
15:     store(msg)
16:     bcast(msg)
17:
18: else
19:     discard(msg)
20: end if
```

2.3 Metrics

In our experiment we focus on the metrics reported in table 1. We considered the number of messages sent/received both at application and MAC layer. Observe that since a message can be delivered in just one packet, we will use these terms as synonymous.

Metrics	Description
TX_{APP}	packets sent by the application layer;
RX_{APP}	packet received by the application layer;
Dup	duplicates received and consequently discarded by the application layer;
TX_{MAC}	transmissions at MAC layer;
RX_{MAC}	reception at MAC layer;
CRC	corrupted packets. These packets show a bad CRC and are discarded;

Table 1: Metrics

Following the design principles introduced in section 2.1 we did not consider metrics on other important aspects such as power consumption. We assumed that nodes are always active.

2.4 The Testbed

In our testbed we used the Tmote SKY nodes produced by Moteiv corporation (http://www.moteiv.com) equipped with Boomerang. Boomerang is the Moteiv-certified distribution of the TinyOS open source operating system that is shipped with the nodes. We limited our attention only on Boomerang for two reasons: a) Boomerang is the suggested and certified operating system, b) we would like to verify our results on an "industrial" solution.
The radio physical layer is 802.15.4 compliant and the link layer protocol is BMAC [15]. We implemented the flooding protocol over BMAC in the *FloodM* module. In order to keep the module as simple and clear as possible we used the *GenericComm* component to send and receive messages. In Boomerang, GenericComm component is deprecated and it is only a wrapper to SP [16]. Sensornet protocol (SP) is an abstraction layer that provides shared neighbor management and a message pool. In our experiments the impact of SP is limited to the following aspects: a) the size of messages increases by 12 bytes (SP header), b) messages are buffered in the SP_MESSAGE_POOL[1], c) messages are re-transmitted when a TX fail occurs, d) the *Mac Backoff* and *Mac Control* interfaces implemented in BMAC are enabled, e) the MAC header size increases by 1 byte. The other functionalities of SP related to the neighbor management, data aggregation, optimization of the duty cycle and are not used. In particular we do not exploit the optimization related to the use of unicast messages; our nodes are always active (i.e. the component used is CC2420AlwaysOn) and messages are sent in broadcast.
In the following we briefly discuss the TinyOS functions used to implement the flooding protocol (see Algorithm 1). Observe that most of the events are asynchronous.

1. Generators start a sample procedure by the *ADC.getData()* (line 4);

2. Once the event *ADC.dataReady()* is captured (i.e. the sampling procedure concludes), the generator call a task *SendData()* to broadcast the message (line 6);

3. the event *SendMsg.sendDone()* concludes the procedure to send a message and a new timer of duration I_{samp}ms is started to schedule the next sampling procedure (line 8).

Non-generators receive the message (*ReceiveMsg.receive()*), discard duplicates and forward the new messages. Metrics at application level (i.e. TX_{APP}, RX_{APP}, Dup) are all collected in the flooding module, while the MAC layer metrics are all collected in the CC2420Radio module.
Once the experiment is completed, the metrics are delivered to the sink using the WSN itself and data can be eventually accessed to be analyzed.

2.5 The Simulation Environment

Objective Modular Network Test-bed in C++ (OMNeT++) is a public-source, component-based, modular simulation framework [18]. It is has been used to simulate communication networks and other distributed systems. The OMNeT++ model is a collection of hierarchically nested modules . The top-level module is also called the System Module or Network. This module contains one or more sub-modules each of which could contain other sub-modules. The modules can be nested to any depth and hence it is possible to capture complex system models in OMNeT++. Modules are distinguished as being either simple or compound. A simple module is associated with a C++ file that supplies the desired behaviors that encapsulate algorithms. Simple modules form the lowest level of the module hierarchy. Users implement simple modules in C++ using the OMNeT++ simulation class library. Compound modules are aggregates of simple modules and are not directly associated with a C++ file that supplies behaviors. Modules communicate by exchanging messages. Each message may be a complex data structure. Messages may be exchanged directly between simple modules (based on their unique ID) or via a series of gates and connections. Messages represent frames or packets in a computer network. The local simulation time advances when a module receives messages from another module or from itself. Self-messages are used by a module to schedule events at a later time. The structure and interface of the modules are specified using a network description language. They implement the underlying behaviors of simple modules. Simulation executions are easily configured via initialization files. It tracks the events generated and ensures that messages are delivered to the right modules at the right time.

To simulate the WSN communication model we exploited the MAC Simulator framework for OMNeT++ [12]. MAC Simulator supports several MAC protocols such as TMAC, TMACP, LMAC, SMAC, CSMA and CSMAACK, but not BMAC [15], the media access control implemented in Boomerang. Our first effort was to extend the Basic MAC module of MAC Simulator to implement the BMAC protocol. We replaced the modules for the traffic generation (Pattern and AppsSelector) with the flooding module. The flooding module is connected to a module implementing the basic functionalities of SP as described in section 2.4 and this module is eventually connected to the BMAC module. We finally set-up the radio timings according to the Chipcon CC2420 radio hardware [7](see section 3.1 for further details).

3. RESULTS

3.1 Preliminaries

In section 2.4 we briefly discuss the implementation of flooding in TinyOS.
Recall that once a message is sent (*SendMsg.Send())* a new timer is scheduled only after the reception of the asynchronous *SendDone* event. We run a simple experiment to evaluate the time elapsed between a call to a Send function and the corresponding SendDone event. The average latency is about 7.9ms. This implies that even if $I_{samp} = x$ ms , messages are actually sent only after $x + 7.9$ms. Thus we modified the timings in the simulation environment in order to reflect this behaviour. We then run a preliminary simulation with 2 nodes and 1 generator and $I_{samp} = 5$ms. The simulation ended after only 18 simulated seconds having generated only 81 messages instead of 18000. Similar, even if slightly

[1]Since a new message is sent only after a SendDone(), the SP_MESSAGE_POOL buffer cannot contain more than one message

name	value	description	old/new
TIME_ON	152.5μs	time to check the CCA	new
BYTE_TIME	32μs	time to tx a byte	was 86.8μs
SFD_TIME	32μs	time to insert the Sync Byte SFD	new
RSSI_CCA	128μs	time for RSSI/carrier sense	new
TXON_SETUP	192μs	turnaround rx:tx	new
DATA_LATENCY	2μs	to access the radio buffer	new

Table 2: CC2420 timings

better results, were obtained also for $I_{samp} = 50$ and 500ms. The problem was that the radio timings were not coherent with the CC2420 radio hardware. We then set these timings according to the mote radio datasheet [7], as summarized in table 2.

In the following we discuss the results of our experiments. Metrics are averaged on 10 runs. $I_{samp} = 5, 50, 500$ms. We limited our attention to four scenarios:

A) 2 nodes, 1 generator;

B) 2 nodes, 2 generators;

C) 6 nodes, 1 generator;

D) 6 nodes, 2 generators.

3.2 Scenario A

We first analyze the case in which $I_{samp} = 5$ms. The number of transmitted and received messages should be in the ideal case (no lost, no error) $TX_{APP} = TX_{MAC} = 18000$. The average number of messages transmitted at application layer is 17569. This is the average of generator and non-generator messages and non-generator node can re-transmit only the messages received from the generator. We observe that the number of messages actually transmitted at mac layer is $TX_{MAC} = 16807 < TX_{APP}$. The number of received messages is even less, only $RX_{MAC} = 16056$.
The reasons of this apparently poor behaviour is twofold: 1) On the transmission side, a new send is possible if and only if the asynchronous event *SendDone()* associated with the previous send is captured. If a new send is generated at application layer before this event is captured, the corresponding message is dropped and does not reach the MAC layer. In our experiments the probability of drop $P_{drop} = 4.3\%$; 2)Once a message has been transmitted at MAC layer, it can experience problems on the channel due to collisions, congestion, noise, interferences etc. Even if the channel is free and clear, and thus the packet could reach the receiver, the radio of the receiver can be in an incompatible state which does not allow to receive a new packet (e.g. it is transmitting). If a packet is experiencing any of the above problems we say it is *cancelled*, thus the corresponding event has a probability $P_{canc} = 4\%$.
Only 10 of the received packets are discarded because of a bad CRC (i.e. $P_{CRC} = 0.06\%$). In the sequel we dubbed the events corresponding to P_{drop}, P_{canc} and P_{CRC} as *experimental evidences.*

In MAC Simulator experimental evidences are only partially considered. Indeed packets can be lost in the propagation model due to collisions and in the radio model due to inconsistencies in the radio state (e.g. a node receives a packets while it is transmitting as in P_{canc}). Packets can also be lost due to an excessive distance between nodes, but since the nodes in our experiments are all in visibility, this cannot happen. In figure 1 we give a pictorial view of where each of the experimental evidences occur. From the figure, only $(1 - P_{drop})$ packets generated at the application layer are received by the MAC layer and forwarded to the physical layer. Similarly only $(1 - P_{CRC})$ packets received at physical layer are error-free and are thus forwarded to the MAC layer, while P_{canc} packets do not reach the intended receiver at all. Observe that coherently with the points outlined in the introduction, we restrict our attention in modelling the effects of the experimental evidences. We leave the modelling of the causes for a future work.

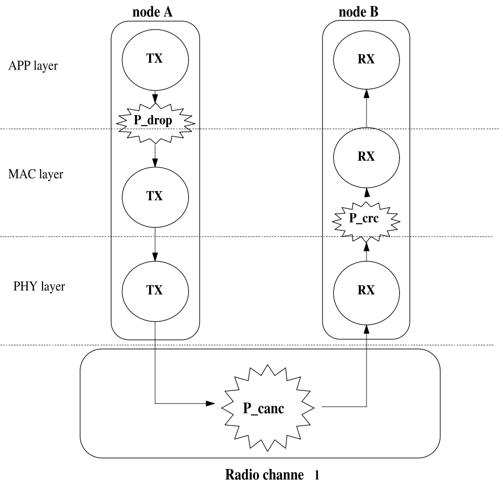

Figure 1: Localization of experimental evidences on simulation model

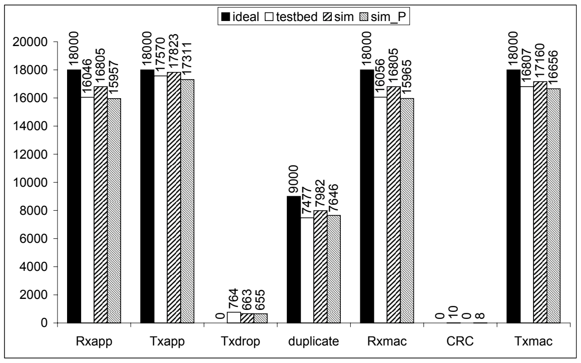

Figure 2: Scenario A I_{samp}=5ms

In figure 2 the results on all the metrics for the ideal case (ideal), the testbed (testbed), the MAC Simulator "as it is" (sim) and finally the case in which the simulation environment is enriched by the experimental evidences (sim_P), are reported. The figure shows that the introduction of the probabilities associated to the experimental evidences make the simulation more reliable.

When I_{sample} increases to 50 or 500 ms the effects of

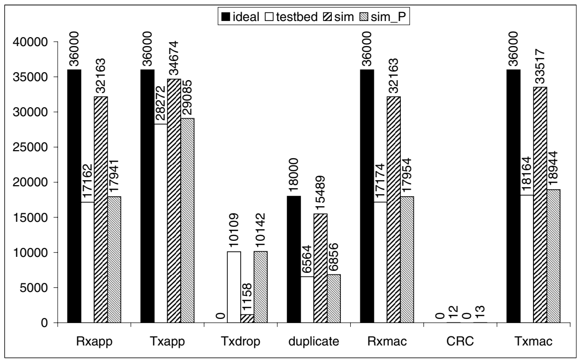

Figure 3: Scenario B I_{samp}=5ms

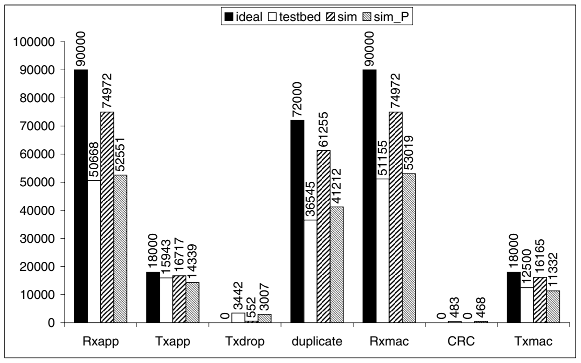

Figure 4: Scenario C I_{samp}=5ms

the experimental evidences are negligible. Indeed time constraints are relaxed ant thus asynchronous events can be easily completed without generating critical conditions.

3.3 Scenario B

In this scenario, each node should transmit $TX_{MAC} = 36000$ messages in the ideal case; 18000 generated by the node itself and 18000 forwarded when received from the other generator. Instead each node transmits an average of 18164 messages and receives only 17174. The problems outlined in the previous section are even more evident with two generators because the number of events to be managed by the nodes strongly increase; each node acts both as a generator and as a forwarder and thus it must manage an higher number of inter-correlated events. In this scenario, when $I_{samp} = 5$ms $P_{drop} = 36\%$, $P_{canc} = 5\%$ and $P_{crc} = 0.06\%$. Observe that P_{crc} is equal to that of scenario A, indeed the actual traffic generated and the environmental conditions are similar to those of scenario A.

When I_{samp} increases, the effects of P_{drop} and P_{canc} are less evident. In particular when $I_{samp} = 50$ms, $P_{drop} = 11\%$ and $P_{canc} = 0.9\%$. P_{drop} and P_{canc} are negligible when the sampling interval is 500ms.

Experimental results are shown in figure 3. The distance between the results of the ideal case and the testbed is always very relevant on all the metrics. To obtain more reliable results we have again to introduce the experimental evidences.

3.4 Scenario C

In this case there are 6 nodes and one of them is a generator. We first consider $I_{samp} = 5$ms. Only 12500 packets are sent at MAC layer by each node and 51156 are received (should be $12500 \cdot 5$ and $18000 \cdot 5$ in the ideal case). The number of received packets is strongly affected by the high generated traffic. This is witnessed by the high value of $P_{canc} = 18\%$ that in this scenario is comparable with $P_{drop} = 22\%$. The number of packets received with a bad CRC is about $P_{crc} = 1\%$. When the sampling interval increases, the percentage of sent messages is 99% of the ideal case (the effects of experimental evidences are less relevant) and the percentage of received messages is 88% of the ideal case.

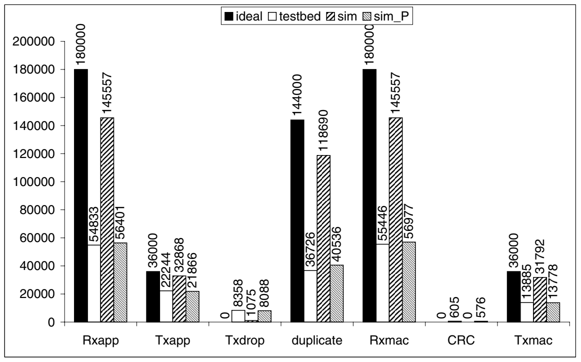

Figure 5: Scenario D I_{samp}=5ms

3.5 Scenario D

The average number of transmitted packets at MAC layer with $I_{samp} = 5$ms is 13885 (should be 36000) and the number of received packets is 55446 ($18000 \cdot 5 \cdot 2$ in the ideal case). The effects of two generators on the traffic are sensible; $P_{canc} = 20\%$ and $P_{drop} = 38\%$ and the number of received packets with a bad CRC is 1%. We observe that in this scenario, when the sampling interval is 50ms performance improvements are less evident with respect to previous scenarios, indeed $P_{canc} = P_{drop} = 13\%$ and P_{crc} is still 1%. When $I_{samp} = 500$ms, P_{drop} is null, while the effects of the other experimental evidences remain significative ($P_{canc} = 11\%$).

In table 3.5 we report the value of the accuracy of RX_{MAC}, TX_{MAC}, RX_{APP} and TX_{APP}. Accuracy measured in terms of distance between the metrics provided by the testbed and those provided by the simulator enriched with the experimental evidences. We notice that accuracy is always greater than 90% independently from I_{samp} and the considered scenario. Furthermore, as expected, when the sampling in-

I_{samp}	Scenario A	Scenario B	Scenario C	Scenario D
5ms	$> 98\%$	$> 95\%$	$> 90\%$	$> 90\%$
50ms	100%	$> 96\%$	$> 96\%$	$> 95\%$
500ms	100%	$> 99\%$	$> 92\%$	$> 92\%$

Table 3: Accuracy of simulations (sim_P) for RX_{MAC},TX_{MAC},RX_{APP} and TX_{APP} metrics

terval increases (time constraints are less strict), accuracy increases.

4. CONCLUSIONS

This work is a first effort to evaluate the reliability of OMNeT++ and the MAC Simulator framework in simulating WSNs. We were able to characterize and localize some of the problems emerging in the simulation experiments when compared with testbed results. Experimental evidences show that a more sophisticated model of the node must be provided to improve the reliability of OMNeT++. We restricted our attention in modelling the effects of the experimental evidences and we show how this approach can actually improve the reliability of simulation results making them a good upper-bound of the testbed results. Nevertheless the validity of this approach is limited to a specific class of experiments. A further effort in modelling causes instead of effects is required. Recently we started to investigate the reliability of other frameworks for WSN modelling in OMNeT++, such as Castalia [1]. Furthermore we plan to extend the study on the reliability also to other simulators such as NS2 and OPNET.

5. REFERENCES

[1] http://castalia.npc.nicta.com.au/.

[2] Mistlab website. http://mistlab.csail.mit.edu/.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirc. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[4] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of manet simulators. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43, New York, NY, USA, 2002. ACM Press.

[5] D. Curren. A survey of simulation in sensor networks. http://www.cs.binghamton.edu/ kang/teaching/cs580s/david.pdf.

[6] E. Ertin, A. Arora, R. Ramnath, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, and M. Nesterenko. Kansei: a testbed for sensing at scale. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 399–406, New York, NY, USA, 2006. ACM Press.

[7] C. P. from Texas Instrument. *CC2420 datasheet.* 2003.

[8] G. P. Halkes, T. van Dam, and K. G. Langendoen. Comparing energy-saving mac protocols for wireless sensor networks. *Mob. Netw. Appl.*, 10(5):783–791, 2005.

[9] D. I., E. C., and F. Alagoz. Mac protocols for wireless sensor networks: a survey. *Communications Magazine, IEEE*, 44:115–121, 2006.

[10] S. Ivanov, A. Herms, and G. Lukas. Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. In *4th Workshop on Mobile Ad-Hoc Networks (WMAN'07)*, 2007.

[11] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 78–82, New York, NY, USA, 2004. ACM Press.

[12] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Comput. Networks*, 43(4):499–518, 2003.

[13] C. Mallanda, A. Suri, V. Kunchakarra, S. S. Iyengar, R. Kannan, A. Durresi, and S. Sastry. Simulating wireless sensor networks with omnet++.

[14] P. Naik and K. M. Sivalingam. A survey of mac protocols for sensor networks. pages 93–107, 2004.

[15] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, New York, NY, USA, 2004. ACM Press.

[16] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, New York, NY, USA, 2005. ACM Press.

[17] N. Ramanathan, E. Kohler, and D. Estrin. Towards a debugging system for sensor networks. *Int. J. Netw. Manag.*, 15(4):223–234, 2005.

[18] A. Varga. The omnet++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001)*, 2001.

[19] A. Varga. *OMNET++ Discrete Event Simulation System User Manual.* 2006.

[20] S. Wang, K. Liu, and F. Hu. Simulation of wireless sensor networks localization with omnet. In *Mobile Technology, Applications and Systems, 2005 2nd International Conference on Mobile Technoloy, Applications and Systems*, 2005.

[21] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: a wireless sensor network testbed. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 68, Piscataway, NJ, USA, 2005. IEEE Press.