2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

An Ns-2 Module for Simulating Passive RFID Systems

Rafael Perazzo Barbosa Mota
Department of Computer Science
University of São Paulo - USP
São Paulo, São Paulo, Brazil
Email: perazzo@ime.usp.br

Daniel M. Batista
Department of Computer Science
University of São Paulo - USP
São Paulo, São Paulo, Brazil
Email: batista@ime.usp.br

Abstract—**This paper presents the design and validation of an RFID module for the Network Simulator 2 which implements the ISO/IEC 18000-6C Class 1 Generation 2 standard, Schoute and Eom-Lee anti-collisions protocols in order to evaluate the performance of RFID systems and Internet of Things environments. The module includes mechanisms for the definition of parameters related to the standards and for QoS provision. Moreover, the implementation is open source and intended to be a reference tool for evaluating RFID protocols for anti-collision algorithms, security, privacy and other desired resources. Brief results obtained with the module showing the benefits of a QoS mechanism proposed by the authors are also presented. The module is important to the network community and this paper provides a scientific methodology attesting its functionality and conformity with existing standards and protocols.**

Keywords*—*RFID, IoT, Simulators, Performance Evaluation.

I. Introduction

The Internet of Things (IoT) is a novel paradigm that is being developed in the modern scenario of wireless communications. The basic idea of this concept is the pervasive presence around us of a variety of things or objects such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones and other technologies [1]. This new communication paradigm is expected to make everyday life more sophisticated, flexible and able to access any object across the world when compared with the current traditional Internet. The radio frequency identification system (RFID) allows us to identify objects, record metadata or control individual targets through radio waves. When RFID readers are connected to the Internet, they can identify, track and monitor the tagged objects. Consequently, RFID is being seen as a key technology for advancing the state of art of the IoT [2].

Simulation is an essential tool in the development and performance evaluation of communication networks, especially those employing new protocols and technologies. Network simulation is useful for researchers and developers who can design and analyze networks efficiently at low cost [3]. Among the applications available for network simulation, the Network Simulator (ns-2) is one of the most popular in the research community. Much of this popularity is because the ns-2 is a free software which implements a rich set of Internet protocols, including several kinds of wired and wireless mechanisms [4].

To the best of our knowledge, there is no published module proposed for simulation of RFID-based systems using ns-2. This paper presents the design and validation of an ns-2 simulation module for IoT scenarios based on RFID systems. The focus of the module implementation is the probabilistic anti-collision approaches such as Q Algorithm[5], Schoute [6], Eom-Lee [7] and mechanisms for QoS support. Also, the module allows IoT researchers to study open challenges and to evaluate a variety of new protocols and view results before acquiring any kind of IoT physical hardware.

The module was validated through comparison with results obtained from previous publications. We compared the main important parameters such as overall system efficiency [8], total number of slots [8] and identification time [9] of the ISO/IEC 18000-6C standard, also known as the Class 1 Generation 2 (C1G2) standard.

The contributions of this paper are:

- The presentation of a novel RFID module for the ns-2 simulator;
- The validation of the novel RFID module considering the ISO/IEC 18000-6C anti-collision standard;
- The presentation of details regarding the implementation of a new QoS mechanism for IoT using the novel RFID module.

This paper is different from those found in the literature because it introduces an open source ns-2 RFID module based on the global communication standard. Existing similar tools don't work with NS-2 and don't follow the part of the ISO/IEC 18000-6C standard related to anti-collision mechanisms.

The rest of this paper is organized as follows: Section II provides the background on RFID simulation tools and ns-2 modules in the literature. In Section III, we present the proposed module and explain how our contribution differs from that of prior work. In Sections IV and V we show the methodology of the experiments to validate our proposed module, comparing results using the Q Algorithm. Section VI suggests future work and Section VII concludes the paper.

II. Background and Related work

Some RFID simulators have been developed in the literature. In [10] an RFID simulation platform is presented, RFIDSim, which allows users to build their own virtual scenario and deploy RFID facilities in it. This simulation platform, which relies on a discrete event simulator, is designed to implement part of the ISO/IEC 18000-6C [5] communication standard related inventory procedures. It does not introduces other anti-collision protocols for evaluation neither trace files. The goal of RFIDSim is to provide users with a visual development platform for RFID applications. Another recent RFID simulator is introduced in [11]. It was developed in line with the ISO/IEC 18000-6C standard for passive tags. This simulator allows the user to know the behavior of the RFID system with a concrete number of tags. Tests have compared them with the latencies measured in a commercial RFID reader to validate the behavior of the simulator. As RFIDSim it does not allow the evaluation of other anti-collision protocols. Vuza, Chitu, and Svasta [12] presents a tag simulator which represents an example of hardware simulation by a combination of software and other hardware. Users must use a real reader and simulate tags with software. As the described simulators are specific to RFID technology only, the integration of other technologies such as Wireless Sensor Networks - WSN and Wifi - is not possible. Our proposed module, as it is implemented in ns-2, allows integration with any supported standard or technology already available in ns-2 which is essential for evaluating and testing IoT scenarios. For example, with our module it is easy to simulate a scenario wherein the RFID reader sends information to an FTP server remotely located, since the TCP/IP stack and FTP protocols are already simulated in ns-2. In an RFID-only simulator it is not trivial to simulate IoT scenarios.

ns-2 simulation results are used to model and validate several wireless protocols, proposals or technologies such as WiMax [13], [14], WSN [15], fixed wireless systems [16], communication among high-speed vehicles or between a vehicle and a roadside infrastructure network [17], refined Wifi [18] and MESH networks [19]. The consolidated acceptance of ns-2 thus justifies the development of an RFID module with the possibility of integration with other existing technologies in the IoT.

III. RFID NS-2 MODULE

The RFID module[1] was developed on release 2.35 of the ns-2 simulator. It is designed not only for the ISO/IEC 18000-6 standard but also for Schoute [6] and Eom-Lee [7] Dynamic Frame Slotted Aloha - DFSA algorithms. It also supports the ns-2 mobility model. The implementation was made in C++. Our contribution consists of the design and implementation of the entire module for ISO/IEC 18000-6C, Schoute and Eom-Lee systems. The scope of this module is the inventory part of the standard (query commands), such as the DFSA algorithms like Q Algorithm, Schoute and Eom-Lee backlog estimation. The session control specified in standard is also implemented. The commands specified on [5] such as SELECT, READ, WRITE are not implemented since they do not affect the identification process performance . In order to simulate the identification process, we implemented the reader-tag communication as shown in Figure 1b [5]. The difference from the complete standard (Figure 1a) is that in the standard there are messages being exchanged to send a random code (called as RNG16) before the tag send its own ID. As our focus is on anti-collision procedures we have omitted this step which does not affect the module accuracy since this step is not counted for performance evaluation.

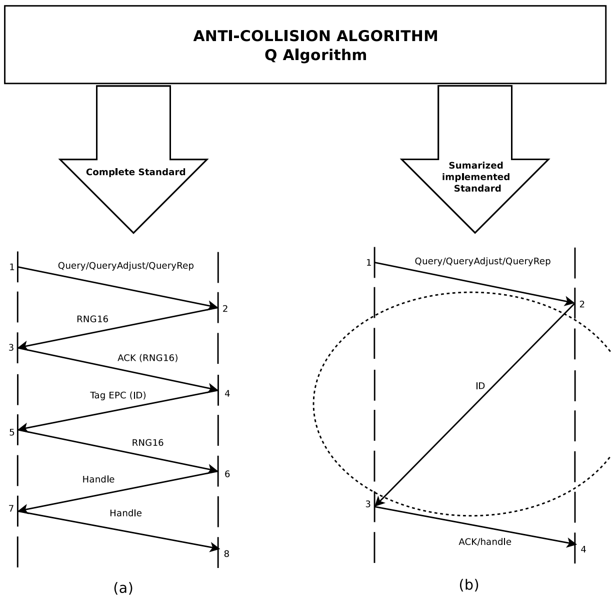

Fig. 1. Example of a single tag identification

Figure 2 presents the module's main classes. These classes were inherited from the ns-2 default *Agent* and *Packet* classes but, as seen in Figure 2, they were modified and redesigned to support specific RFID features. *Agent* class has base methods to allow the sending and receiving of data. *Packet* class is the base class used to specify any kind of data packet. The *RfidReader* class, as illustrated in Figure 2, stores much useful information for performance analysis such as slots counter (success, idle and collision), identification IDs (its own ID and last received tag ID) and standard parameter values (constant *c*, Q value - qValue_ , current command in progress - command_ and QoS provision - mechanism_). Table I shows the modified and created files added by the RFID module to ns-2 default installation. The first column indicates the modified/created files. The second column shows if the file was created (C) or modified (M). The third column summarizes the content of the files.

TABLE I. MODIFIED/CREATED FILES

File	(C)/(M)	Comments
apps/rfidReader.{cc,h}	C	RFID reader class definitions
apps/rfidTag.{cc,h}	C	RFID tag class definitions
apps/rfidPacket.{cc,h}	C	RFID packet class definitions
trace/cmu-trace.cc	M	Inclusion of rfidPacket trace information
tcl/lib/ns-default.tcl	M	Inclusion of RFID reader and tags parameters
common/packet.h	M	Inclusion of rfidPacket parameters
Makefile.in	M	Inclusion of reader, tag and packet classes

A. Reader Agent (RfidReader Class)

The Reader Agent models an RFID reader with its main functions such as all query commands used to identify a set of

[1]http://www.ime.usp.br/~perazzo/rfid

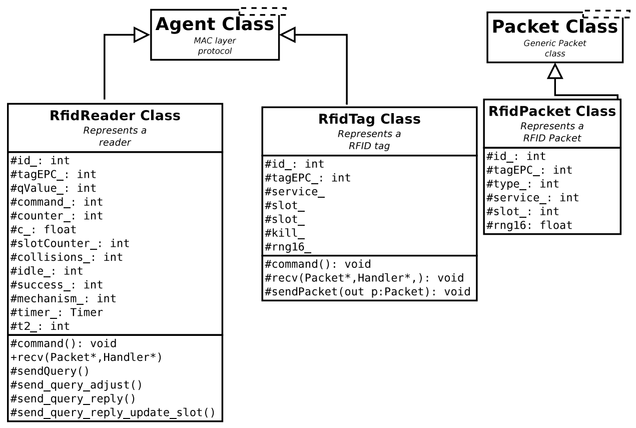

Fig. 2. Class diagram for ns-2 RFID module

tags using the Q algorithm explained in [9] and presented in Algorithm 1. The reader is the node responsible for tag identification according to Algorithm 1 as defined in the standard. It exchanges packets with tags through Query commands (*Query, QueryAdjust and QueryRep*) until all tags are identified. The first step of Algorithm 1 is to get the initial Q value, which represents the initial frame size (possible value range to be randomly selected by each tag). The reader rounds the Q_{fp} value to the nearest integer and assigns this value to Q in the Query command (lines 1-2). The identification process, initialized by a *Query* command (line 3), can be divided in frames of Q size as shown in Figure 3 - I. When there is more than one tag reply during the wait time (t_2 - line 5), we have a collision slot (lines 6-10 and Figure 3 - II). In this case, the value of Q is considered too small or there are several remaining unidentified tags [9]. Therefore, Q_{fp} is incremented by the value of c (line 7) as seen in Figure 3 - II, which is a real number in the interval of $[0.1, 0.5]$, provided by the user before the identification process begins. Otherwise, if there are no tag replies (lines 11-15), we have an idle slot (Figure 3 - II). The reader will not receive any replies if no tags have selected the value of zero for the slot counter. This means that the value of Q is too large or there are a small number of unidentified tags remaining [9]. Therefore, Q_{fp} is decremented by the same value of c (line 12). If only one tag replies (lines 16-20), we have a success slot, and then the reader sends a *QueryRep* command (line 19) to receive the tag ID and to ensure all other tags have their slots decremented by one unit. In the case of collision or idle slots (lines 6-15), the reader tries to adjust the frame size by exchanging the Q value through a *QueryAdjust* command (line 9 or 14) as seen in Figure 3 - II. While the Q value is greater than zero (line 4) the reader keeps exchanging packets with unidentified tags through *QueryAdjust and QueryRep* commands (lines 4-21). *QueryAdjust* works by sending packets to unidentified tags with adjusted Q values. *QueryRep* command sends a packet to all tags changing the identified tag state to *ACKNOWLEDGE* and making all other tags decrease their slots. Table II summarizes the internal implemented reader structure and operations used by Algorithm 1. We designed reader implementation with six groups of variables as seen in Table II:

- Identification: Stores its own identification ID and tag ID (id_ and tagEPC_);
- ISO/IEC 18000-6C standard properties: Deals with all needed parameters related to the Q Algorithm such as Q value (qValue_), command type (command_), replies counter (counter_) and wait time (t_2_);
- Counters: Used to count all kinds of slots such as collisions (collisions_), success (success_), idle (idle_) and total (slotCounter_);
- ISO/IEC 18000-6C standard methods: Implementation of *Query*, *QueryAdjust* and *QueryRep* commands used to identify a population of tags (send_*);
- Send/Receive methods: Used to implement send and receive packets operations (recv() and command()) ;
- QoS provision: Applied to provide QoS for tracking scenarios (mechanism_ and timer_).

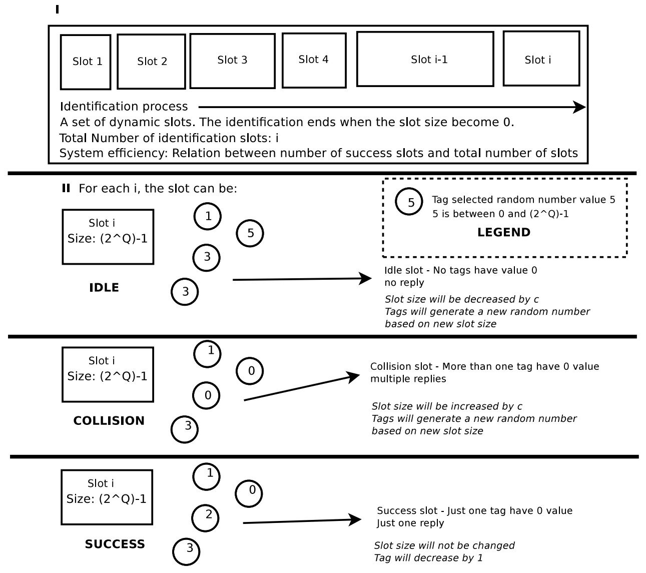
I
Slot 1
Slot 2
Slot 3
Slot 4
Slot i-1
Slot i
Identification process
A set of dynamic slots. The identification ends when the slot size become 0.
Total Number of identification slots: i
System efficiency: Relation between number of success slots and total number of slots
II For each i, the slot can be:
Tag selected random number value 5
5 is between 0 and (2^Q)-1
LEGEND
Slot i
Size: (2^Q)-1
IDLE
Idle slot - No tags have value 0
no reply
Slot size will be decreased by c
Tags will generate a new random number based on new slot size
COLLISION
Collision slot - More than one tag have 0 value
multiple replies
Slot size will be increased by c
Tags will generate a new random number based on new slot size
SUCCESS
Success slot - Just one tag have 0 value
Just one reply
Slot size will not be changed
Tag will decrease by 1

Fig. 3. Q Algorithm explanation

B. Tag Agent (RfidTag Class)

RFID tags are the hardware attached to objects to allow their identification. The tag agent in the module was modeled to operate as a passive tag according to Algorithm 2. This means that tag agents are not able to initiate a communication. Internal structure follows the standard defined in [5] such as *rng16_* (generated number based on received Q value) [5, 46], *memory_* (optional memory), *tag state* (ready, arbitrate, reply and acknowledge) [5, 45] and *service_* as shown in [20] and detailed in Section III-E. The tag operation starts when it receives a command from the reader. It checks the command type (line 3). If the command is a *Query or QueryAdjust* then the tag generates and stores a new random number based on the received Q value (line 5) and changes its state to *ARBITRATE* (line 3). If the generated number is equal to zero (line 6), it immediately replies (lines 7-9). If the received command is a

Algorithm 1: Reader operation (based on [9])

```
Input: $c\ in\ [0.1, 0.5]$
1: $Q_{fp} = 4.0$ {Q value as a float number}
2: $Q = round(Q_{fp})$
3: Send Query Command
4: while $Q \geq 0$ do
5:    Wait $t_2$
6:    if (Received more than one reply) then
7:       $Q_{fp} = Q_{fp} + c$
8:       $Q = round(Q_{fp})$
9:       QueryAdjust()
10:   end if
11:   if (Received no reply) then
12:      $Q_{fp} = Q_{fp} - c$
13:      $Q = round(Q_{fp})$
14:      QueryAdjust()
15:   end if
16:   if (Received just one reply then
17:      $Q_{fp} = Q_{fp} + 0$ {Q value keeps unchanged}
18:      $Q = round(Q_{fp})$
19:      QueryRep()
20:   end if
21: end while
```

TABLE II. READER PROPERTIES AND FUNCTIONS

Parameter	Value
id_	Reader unique identification
tagEPC_	Last received tag ID
qValue_	Generated Q value (frame size) [5, 96]
command_	Current command in progress (query, queryAdjust, queryRep) [5, 53]
t_2_	Reader wait time (time the reader waits after a query command, before change Q value)
counter_	Number of received replies during t2_
collisions_	Total number of collision slots
idle_	Total number of idle slots
success_	Total number of success slots
slotCounter_	Total number of slots (any type)
send_query()	Query command [5, 57]
send_query_adjust()	Query adjust command [5, 58]
send_query_rep()	Query reply command [5, 59] (directed to the tag that had replied)
send_query_reply_update_slot()	Query reply command [5, 59] (directed to all tags that did not reply)
command()	Send an implemented command
recv()	Handle the received packets
mechanism_	QoS mechanism 0 - no QoS; 2 explained in Section III-E
timer_	ns-2 structure to control t_2 waiting time [5, 94]

QueryRep and the tag has zero value on slot number (line 12) then it goes to *ACKNOWLEDGE* state (line 13). Otherwise, if the command is also a *QueryRep* but the slot is not zero then it goes to *ARBITRATE* state (line 16) and decrements its slot number by one unit (line 17). A new verification is made by the tag to check if the new slot number is equal to zero (line 18). If the verification is true then the tag goes to *REPLY* state (line 19) and replies immediately (lines 20-21) and so the process begins again.

Algorithm 2: Tag operation (based on [9])

```
Input: RfidPacket pkt
1: Get cmd type from pkt
2: if $(cmd = Query)||(cmd = QueryAdjust)$ then
3:    $state = ARBITRATE$
4:    Get Q value from pkt
5:    $slotValue = rng16(0, (2^Q) - 1)$
6:    if $(slotValue = 0)$ then
7:       $state = REPLY$
8:       Create new packet p
9:       Send Packet p to requesting reader
10:   end if
11: end if
12: if $(cmd = QueryRep)\&\&(slotValue = 0)$ then
13:    $state = ACKNOWLEDGE$
14: end if
15: if $(cmd = QueryRep)\&\&(slotValue \neq 0)$ then
16:    $state = ARBITRATE$
17:    $slotValue = slotValue - 1$
18:    if $(slotValue = 0)$ then
19:       $state = REPLY$
20:       Create new packet p
21:       Send Packet p to requesting reader
22:    end if
23: end if
```

C. Packet Class (RfidPacket Class)

This class is designed to help tasks that analyze post-simulation generated data, as the trace file is generated through data contained on exchanged packets. The RFID packet provides all information needed to simulate the standard behavior. We modeled the packet class with all the required parameters defined in [5] and some other parameters to allow a detailed trace file. Required parameters include Q value generated by the reader, random number calculated by tags (based on received Q value), flow type (reader to tag or tag to reader), reader ID, tag ID, type of service (tracking service, for instance), command (*Query, QueryAdjust and QueryRep*) and QoS mechanism (based on Algorithm 3 discussed in Section III-E). Additional data include a set of counters (collision, idle, success and total slot counters) and session control (in the case of scenarios with periodic requests such as a tracking scenario).

D. Trace File

The trace file is one of the most important ns-2 components because it allows analysis of the simulation. It enables relevant information to be extracted about the entire simulation. We modeled a trace file to show the main information for RFID protocol analysis such as number of collisions, idle and success slots allowing users to calculate information like system efficiency and total number of slots for anti-collision algorithms. This information allows not only the study of final protocol parameters but also the intermediate steps taken during the entire protocol execution. Our customized RFID trace file presents the information described in Table III.

TABLE III. TRACE FILE FORMAT

Flag	Value	Comments
s/r	Send (s) or receive (r)	s indicates a sending packet and r a receiving packet
-t	Event time	Exact time the event has occurred
-Zt	Flow type	0 represents reader to tag and 1 tag to reader
-Zi	Node ID	The identification ID if the node is a reader or EPC Code if it is a tag
-Zs	Source ID	Identifies who has sent the packet
-Zd	Target ID	Identifies who should get the packet
-Zc	Command type	0 - *Query*; 1 - *QueryAdjust*; 2 - *QueryRep*; 3 - Reader ACK; 4 - Reader NAK; 5 - Tag reply
-Zq	Type of service	Represents what kind of service is being established: 2 - Pure Q Algorithm; 3 - Pure Q Algorithm with QoS provision [20] 4 - Schoute; 5 - Eom-Lee
-Zr	Generated random number	The random number generated by the tags based on the formula: $r = random(0, 2^Q - 1)$
-Zv	Q value	Current reader Q value
-Zz	Slots counter	Counts the total number of slots including success, idle and collision slots
-Zc	Collision slots counter	Counts the total number of collision slots during one identification process
-Zi	Idle slots counter	Counts the total number of idle slots during one identification process
-Zs	Success slots counter	Counts the total number of success slots during one identification process
-Ss	Session ID	Session number (each new *Query* command generates a new session number)

E. QoS Mechanism

Mota and Batista [20] proposed a QoS mechanism for application in RFID tag operation, as described in Algorithm 3. We have implemented this mechanism and made possible its utilization in the proposed ns-2 RFID module. This feature is designed for simulation analysis of IoT tracking scenarios. ()Welbourne, Battle, Cole, et al. [21]) described a real application of this type of scenario. Our mechanism is based on the principle that the tags do not need to reply to all reader queries. Each tag, when it receives a *Query* command, has to check the reader ID (lines 10-11). The first request (line 2) means no reader has sent any previous packet, so the tag replies (lines 3-4) and waits for *ACK* (line 5) before record reader ID (lines 6-7) and goes to *ACKNOWLEDGE* state (line 8). If the request was sent by the same reader (line 11) the tag does not reply (line 12); otherwise, the tag sends its ID (line 15) and waits for the *ACK* (16). When the *ACK* command is properly received the tag records reader ID on memory to avoid sending another reply to the same reader (lines 17-18). The mechanism needs to be implemented only on tags. We have not considered the energy comsumption, making necessary to observe that all tags can change to READY state when there is no more power to store the ACKOWLEDGE state.

IV. EXPERIMENTS

We conducted two sets of experiments for the module evaluation. The first one was used for standard compliance validation. This first set of experiments aimed to get a Q Algorithm performance evaluation comparison based on the most evaluated parameters such as system efficiency, for instance [8], [9], [22], [23]. The second one was done to obtain a performance analysis of the QoS mechanism proposed by Mota and Batista [20] and compare it with the Q Algorithm and the Binary Tree Slotted Aloha Algorithm (BTSA) [2].

Algorithm 3: QoS mechanism proposed in [20]

```
Input: Query Command
    Considering the tag has 0 as slot number
 1: Receive Reader Query Command
 2: if (First request) then
 3:    Go to REPLY state
 4:    Send ID
 5:    if (received an ACK command)) then
 6:       Store Reader ID on optional memory
 7:       $tagMemory \leftarrow readerID$
 8:       Go to ACKNOWLEDGE state
 9:    end if
10: else
11:    if (tagMemory=readerID) then
12:       Stays on ARBITRATE state
13:    else
14:       Go to REPLY state
15:       Send ID
16:       if (received an ACK command) then
17:          Store Reader ID on optional memory
18:          $tagMemory \leftarrow readerID$
19:          Go to ACKNOWLEDGE state
20:       end if
21:    end if
22: end if
```

For the first set of experiments, the system efficiency (S_{ef}) [8], total slot counter (T_{slots}) [8] and identification time (Id_{time}) [9] for the number of tags was calculated according to Equations 1, 2 and 3 respectively.

$$S_{ef} = \frac{\sum_{i=1}^{100} \frac{suc_i}{suc_i + col_i + idl_i}}{100} \tag{1}$$

$$T_{slots} = \frac{\sum_{i=1}^{100} suc_i + col_i + idl_i}{100} \tag{2}$$

$$Id_{time} = \frac{\sum_{i=1}^{1000} time_i}{1000} \tag{3}$$

where

- $i-$ Iteration;
- col_i- Number of collision slots from iteration i;
- idl_i- Number of idle slots from iteration i;
- suc_i- Number of success slots from iteration i;
- $time_i-$ Identification time (in ms).

All required variables were extracted from the trace file for both sets of experiments. The S_{ef} shown in Equation 1 represents the 100-observation average of the ratio between the number of success slots and the total number of slots. Equation 2 calculates the 100-simulation results mean between the total number of slots. The time average calculation is performed through the 1000-simulation results mean of the time taken to identify all tags in the range. All parameters in the first set of experiments are listed in Table IV.

TABLE IV. FIRST SET OF EXPERIMENTS PARAMETERS

Parameter	Value
Simulated time	10 seconds
Number of tags	50 to 1850
Steps	50
Number of readers	1
Initial positions	Random (in reader interrogation zone)
Movements	No movements
Meaning	Tags population identification
Readers queries	One query
Random Number Generator class	Random::uniform(0,$2^q - 1$)
Initial Q value	4
Constant c	0.3
Slot time(T_2)	1 ms (according to [9])
Iterations for S_{ef} and T_{slots}	100
Iterations for Id_{time}	1000

The second set of experiments introduces the *Delay slots* - T_{delay} concept, defined in Equation 4. It represents a mean sum of generated delay slots (idle and collisions) for each reader during the entire simulation. We evaluated this parameter to analyze the reduction of wasted slots (delay slots) when applying the QoS mechanism proposed in [20] compared with the traditional Q Algorithm and the BTSA. This reduction is important because the energy consumption is directly related to the delay slots between the reader and the tag communication[24]. The parameters used in the experiments are detailed in Table V.

$$T_{delay} = \sum_{i=1}^{r} \sum_{j=1}^{m} (col_j + idl_j) \tag{4}$$

where

- $i-$ Reader i;
- $r-$ Number of readers;
- $j-$ Frame j;
- $m-$ Number of frames during the entire simulation;
- col_j- Number of collision slots during frame j;
- idl_j- Number of idle slots during frame j;
- $T_{delay}-$ Total number of delay slots during the entire simulation;

TABLE V. SECOND SET OF EXPERIMENTS PARAMETERS

Simulated time	1800 seconds
Number of tags	100 to 900
Number of readers	3
Steps	200
Initial positions	Random (0-30 meters x 0-15 meters)
Movements	Random Way Point(0-30 meters x 0-15 meters)
Meaning	3 rooms with a reader covering only one location
Readers queries	Each reader issues a query command every minute
Topology	30x15 (meters)
Nodes speed	0-1.5 (meters/second)

Both sets of experiments were performed on a server equipped with an Intel Core i7-2700K 3.5GHz, 16GB of RAM and 1TB of disk space running the Debian GNU / Linux version 6.0 operating system.

V. SIMULATION RESULTS AND VALIDATION

This section presents the simulation results of the experiments we carried out to validate the performance of our proposed RFID module. Figure 4 shows the time taken for tag identification when the population of tags varied between 50 and 500. To verify if the results are as expected, we compared them with the results presented in [9]. A curve with our results and a curve with the results from [9] are presented in Figure 4. This simulation result shows the linear relationship between the number of tags and time spent during the identification process [9], [17], [25], [22], [2]. Comparing our results with [9] we found an average difference of $\cong 11\%$ when the tag population is 50 and 5% when the number of tags is 500. The average difference tends to stay around $5-6\%$ when the number of tags is about 150. The difference is justified because parameters as bandwidth and random number generator are not informed from [9] which affect the results.

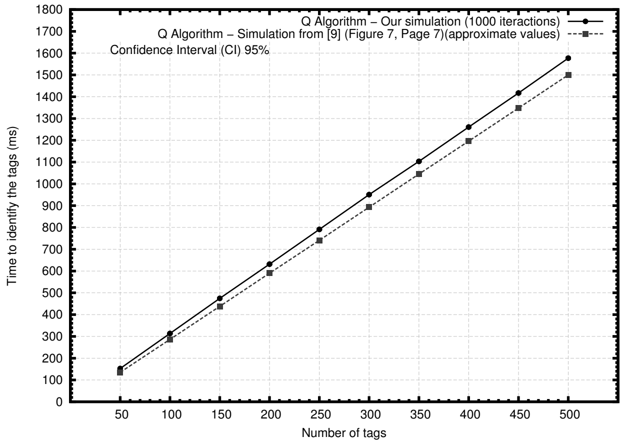

Fig. 4. Tags population identification time

In Figure 5, the curves show the average system efficiency obtained by previous works (we did not re-simulate the previous works but used the numbers presented in these works) and by simulations with our module. Our curves are close to each other (difference around $\cong 6-7.5\%$), appearing between the lower ($\cong 0.29$) and higher ($\cong 0.34$) results, representing a average S_{ef} value of 0.32. The important aspect is that our curve is also near to a constant value for all numbers of tags. Our results are also near to the analytic values ($\cong 0.34$) provided by [9]. Not all simulations in the previous works used the same number of tags. Therefore some curves are not plotted to all values of the x axis.

Figure 6 shows another important parameter used by the authors to evaluate RFID system performance, the total slots generated to identify all tag populations. All curves are close to each other ($\cong 3-5\%$ when the number of tags is higher than 500). The main aspect to be observed in this graph is the type

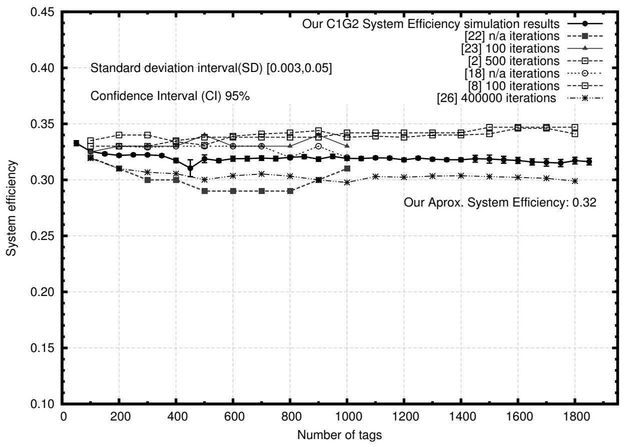

Fig. 5. System efficiency

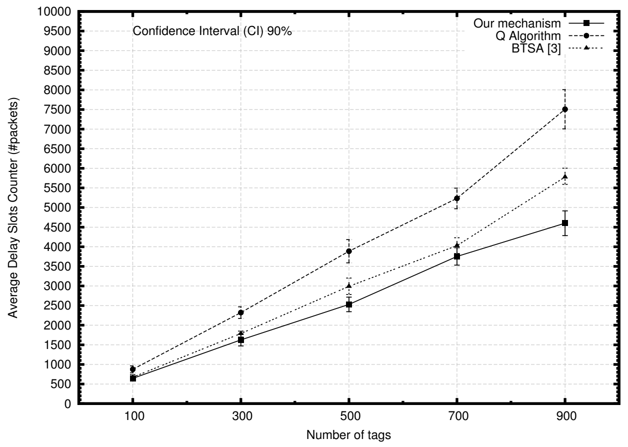

Fig. 7. IoT tracking scenario applying different anti-collision approaches

of curve: all of them have almost linear plots. For both S_{ef} and T_{slots} the observed differences occur because of the different system parameters and the random number generators used. Some curves on this graph also are not plotted to all values of the x axis, because some authors used different numbers of tag population.

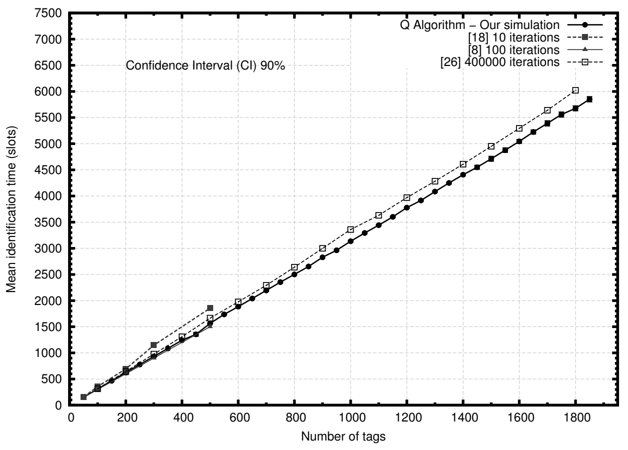

Fig. 6. Total slots counter

The last graph, in Figure 7, shows the simulation results for the second set of experiments. Our results show the obtained gains ($\cong 6 - 37\%$) compared with BTSA and ($\cong 10 - 40\%$) compared with ISO/IEC 18000-6C when we applied the QoS mechanism proposed in [20].

The results presented in this section were obtained to validate our RFID module, and they confirm some observations presented in the literature [8], [9], [22], [23]:

- Id_{time} has a linear curve when the number of tags increases;
- S_{ef} follows an almost constant value;
- T_{slots} has an almost linear plot when related to the number of tags;

These results validate the proposed module, since the protocol behavior works as expected. Besides, our new QoS mechanism was simulated with the RFID module and it has a clear advantage when compared with the Q Algorithm and the BTSA, which is an interesting new finding, as presented in Figure 7.

VI. Future Work

In our future work we plan to extend the current module to allow power consumption evaluation and the complete identification process as shown in Figure 1a instead of summarized one, allowing the evaluation of not only complex IoT plus RFID systems but also simple RFID systems. We will also work towards creating a user friendly interface from which a simulation script can be designed allowing the module features already provided to be used. As RFID ns-2 module is written in a modular way following the object oriented programming (OOP) principles, it is fairly easy to extend it in order to support new anti-collision aproaches and also security/privacy mechanisms. Currently, we are using the module to implement new anti-collision algorithms. Other future work will focus on interoperability between RFID technology and other wireless standards, making research on multi-technology IoT applications possible. We also will add support for active RFID systems.

VII. Conclusions

We introduced a novel RFID module for the ns-2 simulator. The module introduces an improved simulation platform for the RFID and IoT application deployment. The module implements the inventory part of ISO/IEC 18000-6C communication standard and supports passive systems, capture and tag mobility models as well. It also supports other DFSA algorithms such as Schoute and Eom-Lee. It has strong expansibility enabling future integration implementations using other wireless technologies since ns-2 has a lot of extension modules which are freely available. Experiments were designed to evaluate the implementation of the main ISO/IEC 18000-6C MAC functionalities. The results indicate that the module works according to standard requirements and they are similar to those obtained with other tools and via analytical models in the literature. The RFID module can benefit research on RFID/IoT networks, especially those on QoS provision.

Acknowledgments

We thank Leonardo D. Sanchez and Victor M. Ramos of the Universidad Autonoma Metropolitana, Mexico City for providing the simulation results described in [26].

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] H. Wu, Y. Zeng, J. Feng, and Y. Gu, "Binary Tree Slotted ALOHA for Passive RFID Tag Anticollision," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 19–31, 2013.

[3] C. Bouras, S. Charalambides, M. Drakoulelis, G. Kioumourtzis, and K. Stamos, "A Tool for Automating Network Simulation and Processing Tracing Data Files," *Simulation Modelling Practice and Theory*, vol. 30, pp. 90–110, 2013.

[4] S. Ben-Guedria, B. Sans, and J.-F. Frigon, "PolyMAX, a Mobile WiMAX Module for the ns-2 Simulator with QoS and MAC Support," *Simulation Modelling Practice and Theory*, vol. 19, no. 9, pp. 2076 – 2101, 2011.

[5] IncTM, EPCglobal, "EPCTM Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860MHz-960MHz Version 1.2.0," 2008.

[6] F. Schoute, "Dynamic frame length aloha," *Communications, IEEE Transactions on*, vol. 31, no. 4, pp. 565–568, 1983.

[7] J.-B. Eom and T.-J. Lee, "Accurate tag estimation for dynamic framed-slotted aloha in rfid systems," *Communications Letters, IEEE*, vol. 14, no. 1, pp. 60–62, 2010.

[8] F. Baloch and R. Pendse, "Comparison of Transmission Control Protocols Based on EPC C1G2 Standard," *Int. J. Com. Net. Tech*, vol. 1, no. 1, pp. 83–94, 2013.

[9] V. Namboodiri, M. DeSilva, K. Deegala, and S. Ramamoorthy, "An Extensive Study of Slotted Aloha-based RFID Anti-collision Protocols," *Computer Communications*, vol. 35, no. 16, pp. 1955–1966, 2012.

[10] T. Zhang, Y. Yin, D. Yue, Q. Ma, and G. Yu, "A Simulation Platform for RFID Application Deployment Supporting Multiple Scenarios," in *Proceedings of the Eighth International Conference on Computational Intelligence and Security (CIS)*, 2012, pp. 563–567.

[11] H. Landaluce, A. Perallos, and I. Angulo, "A Simulation Tool for RFID EPC Gen2 Protocol," in *Proceedings of the 7th Iberian Conference on Information Systems and Technologies (CISTI)*, 2012, pp. 1–6.

[12] D. Vuza, S. Chitu, and P. Svasta, "An RFID Tag Simulator Based on the Atmel AT91SAM7S64 Micro Controller," in *Proceedings of the 33rd International Spring Seminar on Electronics Technology (ISSE)*, 2010, pp. 427–432.

[13] M. Barria and A. Cordiviola, "Proposal and Evaluation of Load-Dependent Distributed Scheduling Algorithm for WiMAX in Mesh Mode," *IEEE Latin America Transactions*, vol. 10, no. 6, pp. 2309–2315, 2012.

[14] J. Freitag and N. da Fonseca, "WiMAX Module for the ns-2 Simulator," in *Proceedings of the 18th IEEE PIMRC*, 2007, pp. 1–6.

[15] A. Abu-Mahfouz and G. Hancke, "ns-2 Extension to Simulate Localization System in Wireless Sensor Networks," in *Proceedings of the AFRICON*, 2011, pp. 1–7.

[16] S. Castillo and L. Ahumada, "On the Simulation of Fixed Wireless Users in NS-2," in *Proceedings of the 15th IEEE CAMAD*, 2010, pp. 16–20.

[17] S. Lee, K. Jin, D. Kang, S. An, and J. Cha, "Enhancement of 802.11 Modules in ns-2 for Wireless Access on Vehicular Environments and Performance Evaluation," in *Proceedings of the 44th Annual Simulation Symposium*. Society for Computer Simulation International, 2011, pp. 198–204.

[18] Q. Chen, F. Schmidt-Eisenlohr, D. Jiang, M. Torrent-Moreno, L. Delgrossi, and H. Hartenstein, "Overhaul of IEEE 802.11 Modeling and Simulation in ns-2," in *Proceedings of the 10th ACM MSWiM*, 2007, pp. 159–168.

[19] V. Mhatre, "Enhanced Wireless Mesh Networking for ns-2 Simulator," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, pp. 69–72, Jul. 2007.

[20] R. P. B. Mota and D. Batista, "Um Mecanismo para Garantia de QoS na "Internet das Coisas" com RFID," in *Proceedings of the SBRC 2013 (Brazilian Symposium on Computer Networks and Distributed Systems)*, 2013, To be published.

[21] E. Welbourne, L. Battle, G. Cole, K. Gould, K. Rector, S. Raymer, M. Balazinska, and G. Borriello, "Building the Internet of Things using RFID: the RFID Ecosystem Experience," *IEEE Internet Computing*, vol. 13, no. 3, pp. 48–55, 2009.

[22] B. Li, Y. Yang, and J. Wang, "Anti-Collision Issue Analysis in Gen2 Protocol," *Auto-ID Labs White Paper WP-HARDWARE-047*, vol. 3, 2009. [Online]. Available: http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-HARDWARE-047.pdf

[23] Z. Wang, D. Liu, X. Zhou, X. Tan, J. Wang, and H. Min, "Anti-Collision Scheme Analysis of RFID System," *Auto-ID Labs White Paper*, 2007. [Online]. Available: http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-HARDWARE-045.pdf

[24] D. Klair, K.-W. Chin, and R. Raad, "On the Energy Consumption of Pure and Slotted Aloha Based RFID Anti-Collision Protocols," *Computer Communications*, vol. 32, no. 5, pp. 961 – 973, 2009.

[25] W.-T. Chen, "A New RFID Anti-Collision Algorithms for the EPC-global UHF Class-1 Generation-2 standard," in *Proceedings of the 9th International Conference on Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC)*. IEEE COMPUTER SOC, 2012, pp. 811–815.

[26] L. D. Sanchez M and V. M. Ramos R, "Adding Randomness to the EPC Class1 Gen2 Standard for RFID Networks," in *Proceedings of the 23rd IEEE PIMRC*, 2012, pp. 609–614.