



CELAL BAYAR
ÜNİVERSİTESİ

T.R.

CELAL BAYAR UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

COMPUTER ENGINEERING DESIGN II

PROJECT REPORT

SIMULATION OF RFID PROTOCOLS

ON NS-2.35 SIMULATOR

Submitted by

Nobatgeldi Geldimammedov

June 2016

Abstract

RFID (Radio Frequency Identification) systems have gained popularity in recent times and have found large-scale deployment in commercial and enterprise domains. Passive Radio Frequency Identification (RFID) tags, due to their ability to uniquely identify every individual item and low cost, are well suited for supply chain management and are expected to replace barcodes in the near future. The passive RFID system was spotlighted as a future technology for automatic identification. However, unlike barcodes, these tags have a longer range in which they are allowed to be scanned, subjecting them to unauthorized scanning by malicious readers and to various other attacks, including cloning attacks. Therefore, a security protocol for RFID tags is necessary to ensure the privacy and authentication between each tag and their reader. We propose a security enhanced protocol with mutual authentication and data cryptographic mechanism between secure reader and tag for the UHF passive RFID system. RFID poses a number of research challenges, such as interference mitigation, throughput optimization and security over the RF channel. A number of new approaches to address these issues have been proposed recently, but due to the highly integrated nature of passive RFID tags, it is difficult to evaluate them in real-world scenarios. In this project, we used NS-2 environment for simulation of RFID systems, which implemented the tree walking singulation protocol(TWSP).

Note: It is difficult to physically test and validate the effectiveness of novel approaches that improve the speed at which large RFID tag populations can be identified and/or address some of the privacy and security concerns associated with RFID.

Index Terms—Automatic identification, interference, Tree Walking Singulation Protocol, radio-frequency identification (RFID), simulation.

1. Introduction

RADIO-BASED identification technology is gaining a recent explosion of development in both industry and academia. RFID is believed to be an indispensable foundation to realize ubiquitous computing paradigm. In RFID systems, RFID tag consists of three primary components: RFID transponder, RFID transceiver, and an application system. The RFID transponder is composed of a small microchip with data storage, limited logical functionality, and an antenna. The RFID transceiver can be distinguished based on the operation frequency (HF or UHF) or on the powering techniques (active, passive, or semi-passive). Communications between a transceiver and a transponder involve interrogating the transceiver to obtain data, writing data to the transponder, or delivering commands to the transponder. The application system is used to collect data through the transceiver and the database utilizes the data for a variety of purposes. RFID tags have been deployed for several years. Though tagging shipping containers is the largest business space for RFID, there are a number of emerging applicable scenarios. For instance, electronic payment, RFID passports, office folders, microensors, intelligent labels, port management, food production control, animal identification, and so on. It is strongly believed that many more scenarios will be identified when RFID principle is thoroughly understood, cheap components are available, and RFID security is guaranteed.

RFID security is a prerequisite to enable wide applications of RFID systems. RFID efforts are in progress to propose secure system architecture, secure protocols, and secure self-configuration. Due to unique characteristics of RFID systems, all components shall be sufficiently secured, including RFID tags, personal data, transaction, middleware, back-end system, and RFID readers. There are a number of challenges in designing efficient security schemes in RFID systems. First, the air interface of RFID systems poses an inherent problem as regards data confidentiality. Any data transferred over the air could be easily subject to eavesdropping if the transaction is unencrypted. Second, it is often desired to use RFID tags as cryptographic tokens, e.g., in a challenge–response protocol. In this case, the tag must be able to execute a secure cryptographic primitive. However, the low-cost demand for RFID tags (0.05–0.1\$) forces the lack of resources to perform true cryptographic operations. Typically, these systems can only store hundreds of bits

and have 5K to 10K logic gates, but only 250 to 3K can be devoted to security tasks. Finally, for RFID users, the foremost protection involves confidentiality, privacy, and non-reputability. The process of unique identification involves collecting large amounts of personal data, which needs to be highly confidential. Special care must be also taken to ensure that the tag–reader communications is adequately encapsulated and shielded. To address these issues, the security subjects are being explored in various scenarios and emerging standards. The topics include authentication, access control and authorization, attacks, privacy and trust, encryption, dynamic privacy protection, and hardware implementation of algorithms, case studies, and applications.

It is difficult to physically test and validate the effectiveness of novel approaches that improve the speed at which large RFID tag populations can be identified and/or address some of the privacy and security concerns associated with RFID. An accurate evaluation of each protocol concept ideally requires a customized microchip and new reader firmware, which is prohibitively expensive in many cases. Board-level emulators are another method for testing new ideas in RFID. However, it is difficult to emulate the power-up limitations of passive RFID tags with powered circuits. Furthermore, a variety of real-world scenarios must be recreated and tested to truly understand the performance of a new idea. NS-2.35 RFIDSim provides a well-controlled test environment to compare different approaches without the need to deploy hundreds of RFID tags and move them consecutively past an RFID reader in experiments. NS-2.35 RFIDSim can be used to compare and classify different ideas in a variety of use scenarios and reduce the number of concepts which need to be prototyped. Due to the nature of the wireless channel, it is very challenging to accurately predict the signal strength distribution in a particular application environment. The objective of NS-2.35 RFIDSim is to facilitate the relative comparison of different medium access protocols, transmission control strategies, privacy and security enhancements.

2. Network Simulator 2

Simulation is a very important modern technology. It can be applied to different science, engineering, or other application fields for different purposes. Computer assisted simulation can model hypothetical and real-life objects or activities on a computer so that it can be studied to see how the system function. Different variables

can be used to predict the behavior of the system. Computer simulation can be used to assist the modeling and analysis in many natural systems. Typical application areas include physics, chemistry, biology, and human-involved systems in economics, finance or even social science. Other important applications are in the engineering such as civil engineering, structural engineering, mechanical engineering, and computer engineering. Application of simulation technology into networking area such as network traffic simulation, however, is relatively new.

Network simulators are used by people from different areas such as academic researchers, industrial developers, and Quality Assurance (QA) to design, simulate, verify, and analyze the performance of different networks protocols. They can also be used to evaluate the effect of the different parameters on the protocols being studied. Generally a network simulator will comprise of a wide range of networking technologies and protocols and help users to build complex networks from basic building blocks like clusters of nodes and links. With their help, one can design different network topologies using various types of nodes such as end-hosts, hubs, network bridges, routers, optical link-layer devices, and mobile units. NS-2 is one of the most popular open source network simulators. The original NS is a discrete event simulator targeted at networking research.

2.1 Overview

NS-2 is the second version of NS (Network Simulator). NS is originally based on REAL network simulator. The first version of NS was developed in 1989 and evolved a lot over the past few years. The current NS project is supported through DARPA. The current second version NS2 is widely used in academic research and it has a lot of packages contributed by different non-benefit groups.

2.2 Main features

First and foremost, NS-2 is an object-oriented, discrete event driven network simulator which was originally developed at University of California-Berkeley. The programming it uses is C++ and OTcl (Tcl script language with Object-oriented extensions developed at MIT). The usage of these two programming language has its reason. The biggest reason is due to the internal characteristics of these two languages. C++ is efficient to implement a design but it is not very easy to be visual and graphically shown. It's not easy to modify and assemble different components and to change different parameters without a very visual and easy-to-use descriptive language. Moreover, for efficiency reason, NS-2 separates control

path implementations from the data path implementation. The event scheduler and the basic network component objects in the data path are written and compiled using C++ to reduce packet and event processing time. OTcl happens to have the feature that C++ lacks. So the combination of these two languages proves to be very effective. C++ is used to implement the detailed protocol and OTcl is used for users to control the simulation scenario and schedule the events. A simplified user's view of NS2 is shown in figure 2. The OTcl script is used to initiate the event scheduler, set up the network topology, and tell traffic source when to start and stop sending packets through event scheduler. The scenes can be changed easily by programming in the OTcl script. When a user wants to make a new network object, he can either write the new object or assemble a compound object from the existing object library, and plumb the data path through the object. This plumbing makes NS2 very powerful.

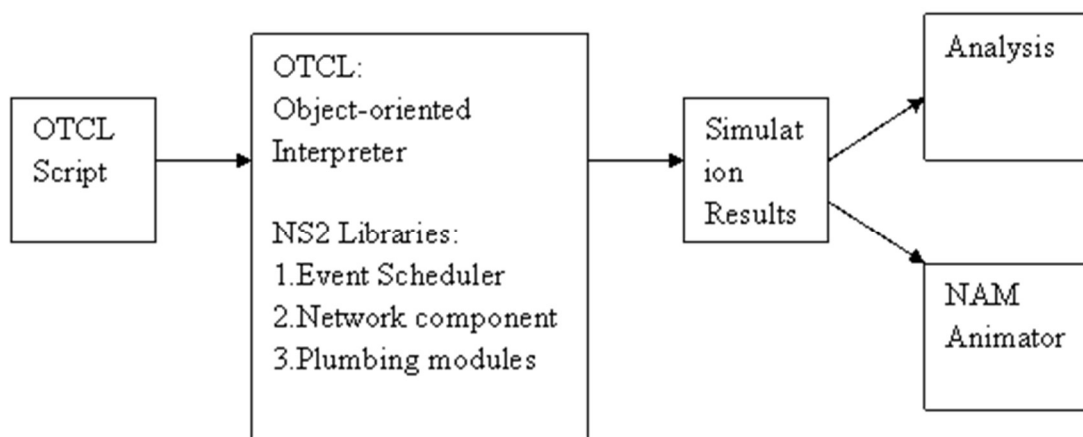


Figure 1. Simplified User's View of NS2

Another feature of NS2 is the event scheduler. In NS2, the event scheduler keeps track of simulation time and release all the events in the event queue by invoking appropriate network components. All the network components use the event scheduler by issuing an event for the packet and waiting for the event to be released before doing further action on the packet.

2.3 Installing Network Simulator 2 (NS2) on Ubuntu

2.3.1 Download and Extract ns2

Download the all in one package for ns2

from <https://sourceforge.net/projects/nsnam/files/latest/download>

The package downloaded will be named "ns-allinone-2.35.tar.gz". Copy it to the home folder. Then in a terminal use the following two commands to extract the contents of the package.:

```
cd ~/
tar -xvzf ns-allinone-2.35.tar.gz
```

All the files will be extracted into a folder called "ns-allinone-2.35".

2.3.2 Building the dependencies

Ns2 requires a few packages to be preinstalled. It also requires the GCC- version 4.3 to work correctly. So install all of them by using the following command:

```
sudo apt-get install build-essential autoconf automake libxmu-dev
```

One of the dependencies mentioned is the compiler GCC-4.3, which is no longer available, and thus we have to install GCC-4.7 version. To do that, use the following command:

```
sudo apt-get install gcc-4.7
```

The image below shows the output of executing both the above commands. If you have all the dependencies pre-installed, as I did, the output will look like the image below:

```
nobat@ubuntu: ~/ns-allinone-2.35
nobat@ubuntu:~/ns-allinone-2.35$ sudo apt-get install build-essential autoconf automake libxmu-dev
[sudo] password for nobat:
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version.
automake is already the newest version.
build-essential is already the newest version.
libxmu-dev is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 239 not upgraded.
nobat@ubuntu:~/ns-allinone-2.35$
```

Once the installation is over , we have to make a change in the "ls.h" file. Use the following steps to make the changes:
Navigate to the folder "linkstate", use the following command. Here it is assumed that the ns folder extracted is in the home folder of your system.

```
cd ~/ns-allinone-2.35/ns-2.35/linkstate
```

Now open the file named "ls.h" and scroll to the 137th line. In that change the word "error" to "this->error". The image below shows the line 137 (highlighted in the image below) after making the changes to the ls.h file. To open the file use the following command:

```
gedit ls.h
```

```
nobat@ubuntu: ~/ns-allinone-2.35/ns-2.35/linkstate
nobat@ubuntu:~/ns-allinone-2.35/ns-2.35$ ls
adc          config.sub   link         realaudio
allinone     configure    linkstate    release_steps.txt
analyze.pl   configure.in mac           routealgo
aodv         COPYRIGHTS  Makefile     routing
aomdv        dccp        Makefile.in  rtproto
apps         delaybox    makefile.vc  satellite
asim         diffserv    mcast        sctp
atualizar.sh diffusion    mdart        sensor-nets
autoconf.h   diffusion3  mobile       src_rtg
autoconf.h.in doc          mpls         stats.pl
autoconf-win32.h dsdv        nix          tcl
BASE-VERSION dsr         ns           tcp
baytcp       empweb      ns.1         test-all
bin          emulate     ns_tclsh.cc  tmix
bitmap       FILES       nstk         TODO.html
cenarios     gaf         packmime     tools
CHANGES.html gen          pgm          tora
classifier   HOWTO-CONTRIBUTE plm          trace
common       inet        puma         validate
conf         indep-utils pushback     validate.out
config.guess install-sh   qs           VERSION
config.h     INSTALL.WIN32 queue        webcache
config.log   lib         rap          wpan
config.status LICENSES    README      xcp
nobat@ubuntu:~/ns-allinone-2.35/ns-2.35$ cd linkstate/
nobat@ubuntu:~/ns-allinone-2.35/ns-2.35/linkstate$ gedit ls.h
```


Save that file and close it.

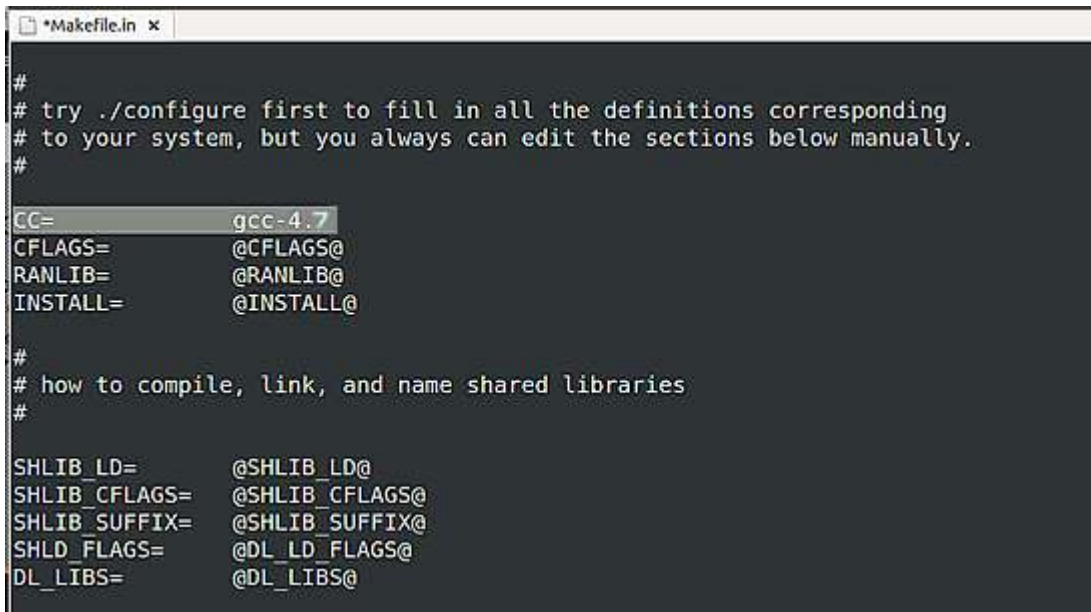
```
// this next typedef of iterator seems extraneous but is required by gcc-2.96
typedef typename map<Key, T, less<Key> >::iterator iterator;
typedef pair<iterator, bool> pair_iterator_bool;
iterator insert(const Key & key, const T & item) {
    typename baseMap::value_type v(key, item);
    pair_iterator_bool ib = baseMap::insert(v);
    return ib.second ? ib.first : baseMap::end();
}

void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
T* findPtr(Key key) {
    iterator it = baseMap::find(key);
    return (it == baseMap::end()) ? (T *)NULL : &((*it).second);
}
```

Now there is one more step that has to be done. We have to tell the ns which version of GCC will be used. To do so, go to your ns folder and type the following command:

```
sudo gedit ns-allinone-2.34/otcl-1.13/Makefile.in
```

In the file, change `CC= @CC@` to `CC=gcc-4.7`, as shown in the image below.



```
*Makefile.in x
#
# try ./configure first to fill in all the definitions corresponding
# to your system, but you always can edit the sections below manually.
#
CC= gcc-4.7
CFLAGS= @CFLAGS@
RANLIB= @RANLIB@
INSTALL= @INSTALL@
#
# how to compile, link, and name shared libraries
#
SHLIB_LD= @SHLIB_LD@
SHLIB_CFLAGS= @SHLIB_CFLAGS@
SHLIB_SUFFIX= @SHLIB_SUFFIX@
SHLD_FLAGS= @DL_LD_FLAGS@
DL_LIBS= @DL_LIBS@
```

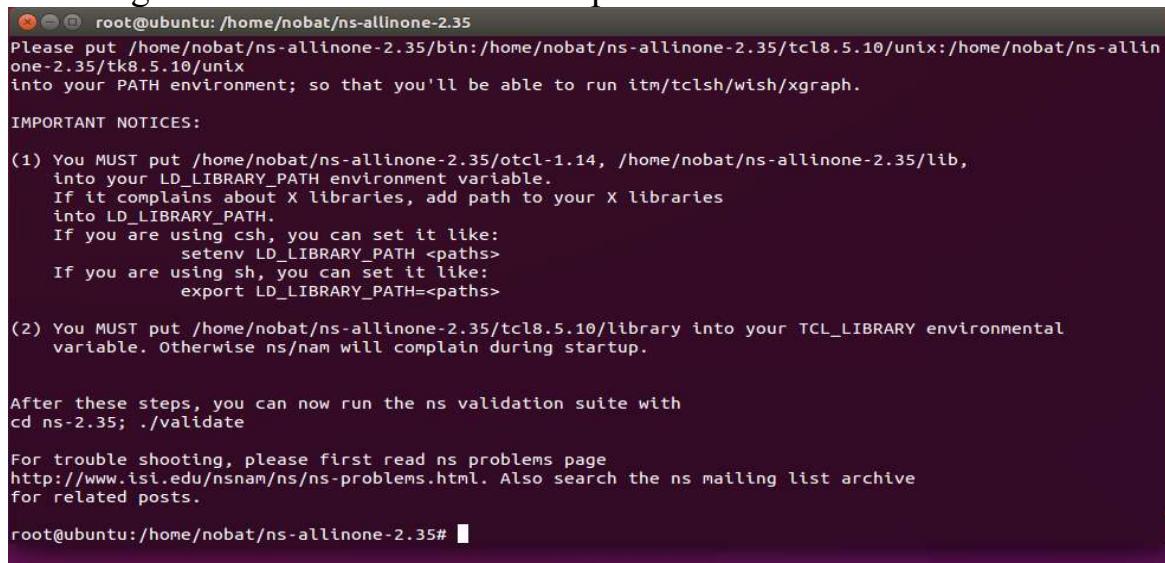
Save that file and close it.

2.3.3 Installation

Now we are ready to install ns2. To do so we first require root privileges and then we can run the install script. Use the following two commands:

```
sudo su cd ~/ns-allinone-2.35/./install
```

The image below shows how it looks upon successful execution



```
root@ubuntu: /home/nobat/ns-allinone-2.35
Please put /home/nobat/ns-allinone-2.35/bin:/home/nobat/ns-allinone-2.35/tcl8.5.10/unix:/home/nobat/ns-allinone-2.35/tk8.5.10/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.

IMPORTANT NOTICES:

(1) You MUST put /home/nobat/ns-allinone-2.35/otcl-1.14, /home/nobat/ns-allinone-2.35/lib,
into your LD_LIBRARY_PATH environment variable.
If it complains about X libraries, add path to your X libraries
into LD_LIBRARY_PATH.
If you are using csh, you can set it like:
    setenv LD_LIBRARY_PATH <paths>
If you are using sh, you can set it like:
    export LD_LIBRARY_PATH=<paths>

(2) You MUST put /home/nobat/ns-allinone-2.35/tcl8.5.10/library into your TCL_LIBRARY environmental
variable. Otherwise ns/nam will complain during startup.

After these steps, you can now run the ns validation suite with
cd ns-2.35; ./validate

For trouble shooting, please first read ns problems page
http://www.isi.edu/nsnam/ns/ns-problems.html. Also search the ns mailing list archive
for related posts.

root@ubuntu:/home/nobat/ns-allinone-2.35#
```

It took almost 6 minutes to build and install ns2 on my system. But before we run it, we need to add the build path to the environment path.

2.3.4 Setting the Environment Path

The final step is to tell the system, where the files for ns-2 are installed or present. To do that, we have to set the environment path using the ".bashrc" file. In that file, we need to add a few lines at the bottom. The things to be added are given below. But for the path indicated below, many of those lines have **"/home/user_name/ns-allinone-2.35/..."**, but that is where I have my extracted folder. Make sure you replace them with your path. For example, if you have installed it in a folder **"/home/user_name"**, then replace **"/home/user_name/ns-allinone-2.35/otcl-1.14"** with **"/home/user_name/ns-allinone-2.35/otcl-1.14"**. Do this for all the required lines.

```
sudo gedit ~/.bashrc
```

Lines to be added:

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/ user_name /ns-allinone-2.35/otcl-1.14
NS2_LIB=/home/ user_name /ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB

# TCL_LIBRARY
TCL_LIB=/home/ user_name /ns-allinone-2.35/tcl8.5.10/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/home/akshay/ns-allinone-2.35/bin:/home/akshay/ns-allinone-2.35/tcl8.5.10/unix:/home/akshay/ns-allinone-2.35/tk8.5.10/unix
#the above two lines beginning from xgraph and ending with unix should come on the same line
NS=/home/ user_name /ns-allinone-2.35/ns-2.35/
NAM=/home/ user_name /ns-allinone-2.35/nam-1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Once the changes have been made, save the file and restart the system.

2.3.5 Running ns2

Once the system has restarted, open a terminal and start ns2 by using the following command:

```
ns
```

If the installation is correct then the terminal looks like the image below:

```
root@ubuntu: /home/nobat/ns-allinone-2.35
one-2.35/tk8.5.10/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.

IMPORTANT NOTICES:

(1) You MUST put /home/nobat/ns-allinone-2.35/otcl-1.14, /home/nobat/ns-allinone-2.35/lib,
into your LD_LIBRARY_PATH environment variable.
If it complains about X libraries, add path to your X libraries
into LD_LIBRARY_PATH.
If you are using csh, you can set it like:
    setenv LD_LIBRARY_PATH <paths>
If you are using sh, you can set it like:
    export LD_LIBRARY_PATH=<paths>

(2) You MUST put /home/nobat/ns-allinone-2.35/tcl8.5.10/library into your TCL_LIBRARY environmental
variable. Otherwise ns/nam will complain during startup.

After these steps, you can now run the ns validation suite with
cd ns-2.35; ./validate

For trouble shooting, please first read ns problems page
http://www.isi.edu/nsnam/ns/ns-problems.html. Also search the ns mailing list archive
for related posts.

root@ubuntu: /home/nobat/ns-allinone-2.35# ns
% █
```

2.4 RFID module for ns-2 simulator

The RFID for NS-2 module was developed by Rafael Perazzo Barbosa Mota, member of the **Computational Systems research group, Institute of Mathematics and Statistics (IME), University of São Paulo (USP)**, Brazil. This module is a free software. This project implements a RFID module based on the EPC_{TM} Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz ? 960 MHz Version 1.2.0(25/10/2008) standard for the ns-2 simulator, release 2.35. The focus of this implementation is the Network layer and its mechanisms for anti-collision.

2.4.1 Features

These are the features of the RFID ns-2 module:

- RFID readers and tags nodes
- Q algorithm anti-collision mechanism
- Trace file including all nodes identification, time and identified collisions
- QoS mechanism for use in Internet of Things (IoT) scenarios.
- Configurable variables: Q initial value; Constant c; type of service (tracking is the only service implemented until now); Nodes ID.
- MAC layer based on ns-2 mac (mac.cc).

2.4.2 Installation

To install the module, follow the instructions below:

1. Download the latest ns-2 RFID module source code from [github](https://github.com/Nobatgeldi/ns-2_rfid).
`https://github.com/Nobatgeldi/ns-2_rfid`
2. Unzip the ns-2 RFID module source code file inside ns-allinone-2.35 folder.
3. Go to the ns-allinone-2.35/ns-2.35 directory and execute `./configure`, `"make clean"` and `"make"`.

2.4.3 Running the simulation

To run the simulation, go to **cenarios** folder.

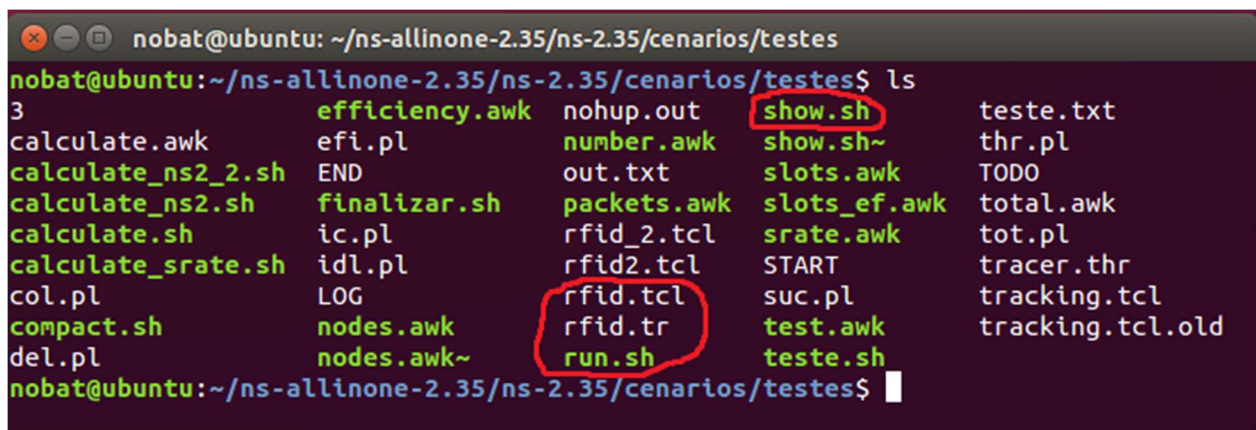
```
cd ~/ns-allinone-2.35/ns-2.35/cenarios
```

There are many cenarios, we use **testes**.

```
cd ~/ns-allinone-2.35/ns-2.35/ cenarios/testes
```

First, we need to see all files which are in this folder.

```
cd ~/ns-allinone-2.35/ns-2.35/ cenarios/testes# ls
```



```
nobat@ubuntu: ~/ns-allinone-2.35/ns-2.35/cenarios/testes
nobat@ubuntu:~/ns-allinone-2.35/ns-2.35/cenarios/testes$ ls
3                efficiency.awk  nohup.out       show.sh         teste.txt
calculate.awk    efi.pl          number.awk      show.sh~        thr.pl
calculate_ns2_2.sh  END            out.txt        slots.awk       TODO
calculate_ns2.sh  finalizar.sh    packets.awk     slots_ef.awk    total.awk
calculate.sh      ic.pl          rfid_2.tcl     srate.awk      tot.pl
calculate_srate.sh idl.pl         rfid2.tcl      START          tracer.thr
col.pl           LOG            rfid.tcl       suc.pl         tracking.tcl
compact.sh       nodes.awk      rfid.tr        test.awk       tracking.tcl.old
del.pl           nodes.awk~    run.sh         teste.sh
nobat@ubuntu:~/ns-allinone-2.35/ns-2.35/cenarios/testes$
```

As we can see, we will use this four files.

Show.sh file content: (ns <Scenario File> <Trace File> <Number Of Nodes>)

```
rm -f *.tr
rm -f *.csv
rm -f *.nam
ns rfid.tcl $1 $2 rfid.tr 5
awk -f nodes.awk rfid.tr
```

Run.sh file content:

```
ns rfid.tcl
./filtrar.sh
pico rfid_filtrado.tr
```

Rfid.tcl file content:

```
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model: TwoRayGround/FreeSpace
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 1000 ;# max packet in ifq
set val(nn) [lindex $argv 1] ;# number of mobilenodes
set val(rp) DumbAgent ;# routing protocol
#set val(rp) DSDV ;# routing protocol
set val(x) 30 ;# X dimension of topography
set val(y) 30 ;# Y dimension of topography
set val(stop) 10 ;# time of simulation end
set ns [new Simulator]

.....
...file      have      continued      https://github.com/Nobatgeldi/ns-2\_rfid/blob/master/ns-2.35/cenarios/testes/rfid.tcl
```


Rfid.tr file content: result file

```
s -t 0.000000000 -Zt 0 -Zi 0 -Zs 200 -Zd -1 -Zc 0 -Zq 2 -Zv 4 -Zz 0 Xc 0 Xi 0 Xs 0 -Ss 1
r -t 0.000050005 -Zt 0 -Zi 3 -Zs 200 -Zd -1 -Zc 0 -Zq 2 -Zv 4 -Zz 0 Xc 0 Xi 0 Xs 0 -Ss 1
r -t 0.000050006 -Zt 0 -Zi 4 -Zs 200 -Zd -1 -Zc 0 -Zq 2 -Zv 4 -Zz 0 Xc 0 Xi 0 Xs 0 -Ss 1
r -t 0.000050008 -Zt 0 -Zi 1 -Zs 200 -Zd -1 -Zc 0 -Zq 2 -Zv 4 -Zz 0 Xc 0 Xi 0 Xs 0 -Ss 1
r -t 0.000050008 -Zt 0 -Zi 2 -Zs 200 -Zd -1 -Zc 0 -Zq 2 -Zv 4 -Zz 0 Xc 0 Xi 0 Xs 0 -Ss 1
s -t 0.001000000 -Zt 0 -Zi 0 -Zs 200 -Zd -1 -Zc 1 -Zq 2 -Zv 4 -Zz 1 Xc 0 Xi 1 Xs 0 -Ss 1
r -t 0.001050005 -Zt 0 -Zi 3 -Zs 200 -Zd -1 -Zc 1 -Zq 2 -Zv 4 -Zz 1 Xc 0 Xi 1 Xs 0 -Ss 1
r -t 0.001050006 -Zt 0 -Zi 4 -Zs 200 -Zd -1 -Zc 1 -Zq 2 -Zv 4 -Zz 1 Xc 0 Xi 1 Xs 0 -Ss 1
r -t 0.001050008 -Zt 0 -Zi 1 -Zs 200 -Zd -1 -Zc 1 -Zq 2 -Zv 4 -Zz 1 Xc 0 Xi 1 Xs 0 -Ss 1
```

In the `show.sh` file, you can see `nodes.awk` file. This is the script for analyzing the result file.

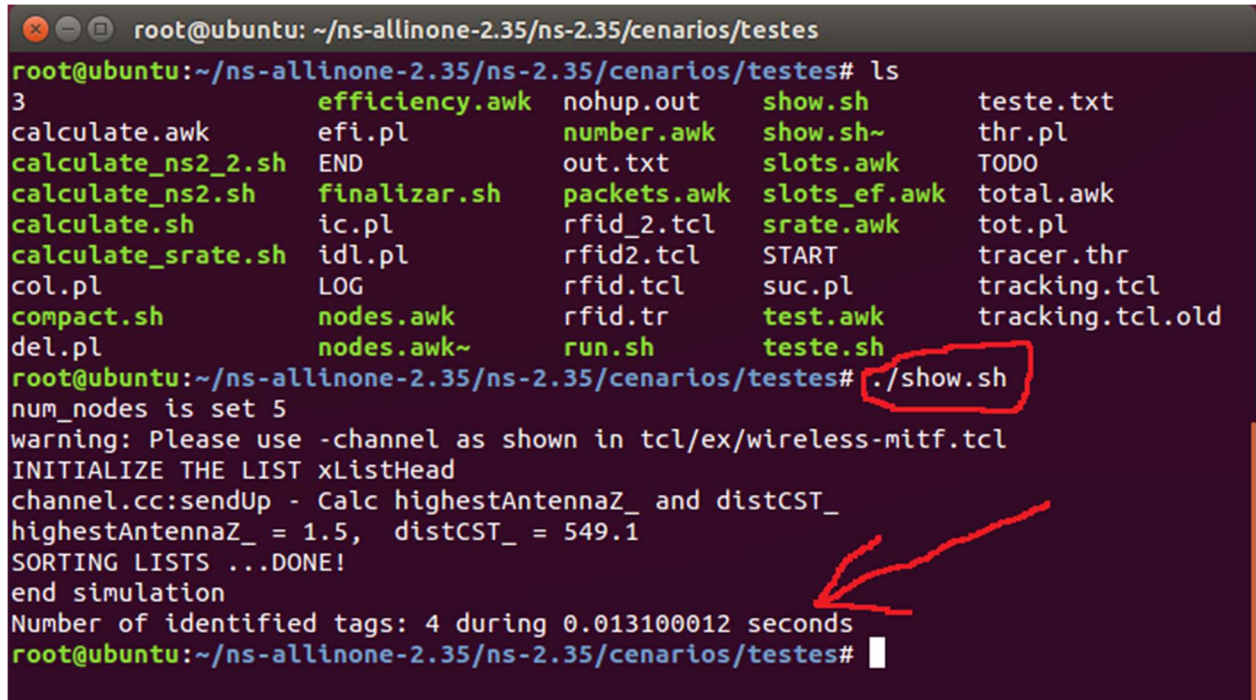
Nodes.awk file content:

```
BEGIN{
    i=0;
    b=0;
    k=0;
}
{
    action=$1;
    cl1=$7;
    cl2=$3;
    cl3=$13;
    if( (action=="r")&&(cl1==0)&&(cl3==5) ){
        c1[i]=action;
        c2[i]=cl1;
        c3[i]=cl2;
        c4[i]=cl3;
        i++;
    }
}

END {
    for(a=1; a in c2;a++)
    {
        b++;
    }
    for (j=1; j in c1; j++)
    {
        k++;
    }
    printf("Number of identified tags: %d during %.9f seconds \n", j, c3[b-1]);
}
```

Now we will run `show.sh` to start simulation.

```
cd ~/ns-allinone-2.35/ns-2.35/cenarios/testes# ./show.sh
```



```
root@ubuntu: ~/ns-allinone-2.35/ns-2.35/cenarios/testes
root@ubuntu:~/ns-allinone-2.35/ns-2.35/cenarios/testes# ls
3                efficiency.awk  nohup.out       show.sh         teste.txt
calculate.awk    efi.pl          number.awk      show.sh~        thr.pl
calculate_ns2_2.sh END            out.txt        slots.awk       TODO
calculate_ns2.sh  finalizar.sh    packets.awk     slots_ef.awk    total.awk
calculate.sh      ic.pl          rfid_2.tcl     srates.awk      tot.pl
calculate_srates.sh idl.pl        rfid2.tcl      START          tracer.thr
col.pl           LOG           rfid.tcl       suc.pl          tracking.tcl
compact.sh       nodes.awk      rfid.tr        test.awk        tracking.tcl.old
del.pl           nodes.awk~     run.sh         teste.sh
root@ubuntu:~/ns-allinone-2.35/ns-2.35/cenarios/testes# ./show.sh
num_nodes is set 5
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5, distCST_ = 549.1
SORTING LISTS ...DONE!
end simulation
Number of identified tags: 4 during 0.013100012 seconds
root@ubuntu:~/ns-allinone-2.35/ns-2.35/cenarios/testes#
```

That is it. Now let's analyze the result and look how this simulator work.

When we run `show.sh`, it call's the other four files respectively. Most important file is `rfid.tcl` in here. It setting simulator for simulation and start simulation. After that, results are being written to `rfid.tr` and analyzed by `nodes.awk`. That is the basic concept of this simulator

3.