ns-2 Extension to Simulate Localization System in Wireless Sensor Networks

Adnan M. Abu-Mahfouz[1]
[1]Departement of Electrical, Electronic and Computer Engineering, University of Pretoria
Pretoria, South Africa
Adnan.Mahfouz@up.ac.za

Gerhard P. Hancke[1,2]
[2]ISG Smart Card Center, Royal Holloway, University of London
London, United Kingdom
Gerhard.hancke@rhul.ac.uk

Abstract—**The ns-2 network simulator is one of the most widely used tools by researchers to investigate the characteristics of wireless sensor networks. Academic papers focus on results and rarely include details of how ns-2 simulations are implemented, and if published the modules tend to be scheme specific. The availability of generic and reusable modules for ns-2, which supports specific research areas while allowing great scope for customization, are limited. Researchers new to ns-2 are therefore often required to build simulations largely from first principles. This paper presents an extension to the current version of ns-2, which enables a normal user, who has basic knowledge of ns-2, to implement and simulate any custom localization system within a wireless network. The technical content of this paper would be beneficial to researchers who want to implement new or existing localization algorithms and anyone new to ns-2 who wish to know more about how a simulation project is built and structured.**

Keywords-simulator; localization; ns-2; sensor's position

I. Introduction

Location information plays a critical role in wireless sensor networks (WSN). Most of the WSN applications and techniques require that the sensor nodes' positions must be determined. Localization algorithms (e.g. [1-5]) follow several approaches to estimate sensor nodes' position. One approach is to use special nodes called beacons, which know their location, e.g. through a GPS receiver or manual configuration. The other nodes that do not know their location, sometimes referred to as unknowns, use different techniques to compute their own position based on the location information of the beacons and the measured distance to these beacons. Once the unknown obtains its position, it could act as a reference for other unknowns. One way to estimate node's position is by using the multilateration method [6], a node within the range of at least three beacons (or references) can estimate its position by minimizing the differences between the measured distances and the estimated Euclidean distances in order to obtain the minimum mean square estimate (MMSE) from the noisy distance measurements.

The difficulties of setting up a WSN with real nodes and the infeasibility of analysis make simulation an essential tool to study WSNs. Simulation is widely used in system modeling for applications ranging from engineering research, business analysis, manufacturing planning and biological science experimentation [7]. There are several simulation packages that can be used to simulate WSN, such as ns-2 [8], OMNET++ [9] and TOSSIM [10].

ns-2 is an open-source event-driven simulator designed specifically for research in networks. ns-2 was developed in C++ and uses Object-oriented Tool command Language (OTcl) as a configuration and script interface (i.e., a front-end). Each language has two types of classes. The first type includes the standalone C++ and OTcl classes that are not linked together. The second type includes classes which are linked between the two languages. These C++ and OTcl classes are called compiled hierarchy and interpreted hierarchy, respectively. These two hierarchies are linked together using an OTcl/C++ interface called TclCL [11].

ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. Recently, Morávek [12] investigated the capabilities of ns-2 with regards to its suitability to model localization in wireless networks. This investigation concluded that ns-2 had all the basic properties to support simulations of different localization techniques (such as time of arrival and received signal strength (RSS)) contained in several ns-2 tools and modules, and that ns-2 allowed researchers a high level of independence from the designed framework of the simulator, allowing developers to modify existing modules or create new modules. This freedom in development, however, means a developer is faced with a large number of implementation options, which requires widespread modification of modules and an in-depth knowledge of the inner workings of ns-2.

In this paper, the current version of ns-2 is extended to simulate localization system in wireless sensor networks (Section II). This extension collects together and leverages ns-2 existing properties for localization modeling, offering a simplified and clearly defined development route for implementing new schemes. New modules are added and a base class (called Position), which enables the sensor nodes to estimate their position using the general multilateration method, is created. New classes derived from "Position" class can be created to implement other localization algorithms. A simulation is conducted in Section III to evaluate the performance of Position class. Results are presented for five localization algorithms (which are derived form Position class)

in terms of localization error, number of references used and remaining energy.

II. Localization extension structure

ns-2 contains several flexible modules for energy-constrained wireless ad-hoc networks, which encourages researchers to use ns-2 to investigate the characteristics of WSNs. However, to implement and evaluate localization algorithms, the current ns-2 version (ns-2.34) should be extended and new modules should be added. This section describes the class and file structure of the localization extension presented in this paper. The main purpose of this extension is to provide researchers with a simplified development path when evaluating localization schemes. Although the entire structure is presented for the sake of completeness, not all the classes and files need to be modified to implement a new scheme. Only the classes and files shaded in yellow in Fig. 1, Fig. 2 and Fig. 3 need to be customized.

Fig. 1 shows the new classes that were added to the ns-2. These classes can be divided into two types. Firstly, there are standalone classes, such as the Position class. These classes are used only from the C++ domain. Secondly, there are compiled hierarchy classes, such as the LocDisApp class. In fact, no OTcl modules were created. However, in order to be able to access the newly compiled hierarchy classes LocDisApp, LocReqAgent and LocResAgent from the OTcl domain, these classes were mapped and linked to corresponding OTcl classes, which are Application/LocDis, Agent/LocReq and Agent/LocRes, respectively. In this way the users are able to create an object of the compiled hierarchy classes from the OTcl domain. For example, the OTcl command "set lreq [new Agent/LocRec]" will create a new object of class LocReqAgent.

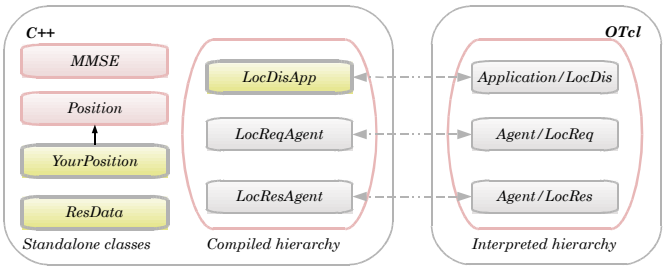

Figure 1. The new modules added to the ns-2

A. Class Hierarchy

The Doxygen documentation system [13] was used to illustrate the class hierarchy of the new classes as shown in Fig. 2. For the sake of simplicity, only the new classes, the classes they are derived from (i.e. parent classes) and the classes used by these new classes were included. Solid lines show where a class is inheriting from another class, for example $A \rightarrow B$ means class A is derived from class B. Dotted lines show where a class is using a method and/or member of another class.

1)MMSE Class

This class is responsible for all the mathematical matrices operations associated with the MMSE estimation. Instead of using general matrix multiplication and matrix inverse, optimized methods dedicated mainly to MMSE were implemented. These optimized methods require less computation and shorter execution time.

2)Position Class

Position class represents the general multilateration method to estimate the node position. This method uses all of the available references, does not distinguish between beacons and references, does not weigh the references used and performs the estimation only once. This class is the base class for any localization algorithms implemented, represented in the text and figures as the YourPosition class. For the sake of simplicity Fig. 1, Fig. 2 and Fig. 3 include only these two classes, while another localization algorithms are also implemented that will be mentioned later.

3)YourPosition Class

This class is derived from Position class and is the core class for defining a new localization scheme. Compared with the Position class, the YourPosition class will include functions specific to the scheme being implemented. For example, for an implementation of ALWadHA [1], one of the schemes tested in Section III, this class would include scheme specific methods such as the smart reference-selection method, specified number of references and termination criterion.

As shown in Fig. 2, the Position class uses the MMSE and LocDisApp classes and the ReferenceNode structure, which consists of two members, location variable and double variables to store the measured distance. The YourPosition class uses an array of this structure (ref_nodes_) to store the location information (location and distance) of neighboring references. The Location class represents the X, Y and Z coordination of sensor nodes.

4)LocReqAgent and LocResAgent Classes

These classes are derived from the Agent class. LocReqAgent constructs and broadcasts a "location request" packet. This agent should be attached only to unknown nodes because beacons already know their location. LocResAgent is responsible for handling the packets received. This agent should be attached to all nodes (unknowns and beacons). Two types of packets could be received. The first is a "location request" packet. If this type of packet is received by a beacon or reference node it constructs a "location response" packet that includes location information and then sends it to the requesting node. Unknown nodes receiving this type of packet simply deallocate it. The second is a "location response" packet. The requesting node that receives this packet sends it to the application layer (LocDisApp), which processes the included location information to estimate the node's position.

Two new types of packet were created: firstly, "location request" packet (PT_LOCREQ), which uses the new protocol-specific header (PSH) defined in the structure hdr_locreq, and secondly, "location response" packet (PT_LOCRES), which uses the new PSH defined in the structure hdr_locres. LocReqAgent uses only hdr_locreq to construct "location request" packets. LocResAgent uses both of the headers' structure, it uses hdr_locreq to gain access and to process the

received "location request" packets, while it uses the hdr_locres to construct the "location response" packets.

5)LocDisApp Class

The LocDisApp class is derived from the Application class. Each node in the network uses an object from this class by attaching it to its agent(s). The LocDisApp class performs several functions, such as invoking the broadcast method of LocReqAgent periodically to broadcast a "location request" packet, processing the received "location response" packet and estimating the node location. As shown in Fig. 2, this class collaborates with several classes, which are the three timers, the two agents, Location and Position classes. It uses the hdr_locres header structure to gain access to the received "location response" packets in order to process the included location information and estimate the node location. LocDisApp uses a vector of class ResData, which is used to store the location information received from neighboring references.

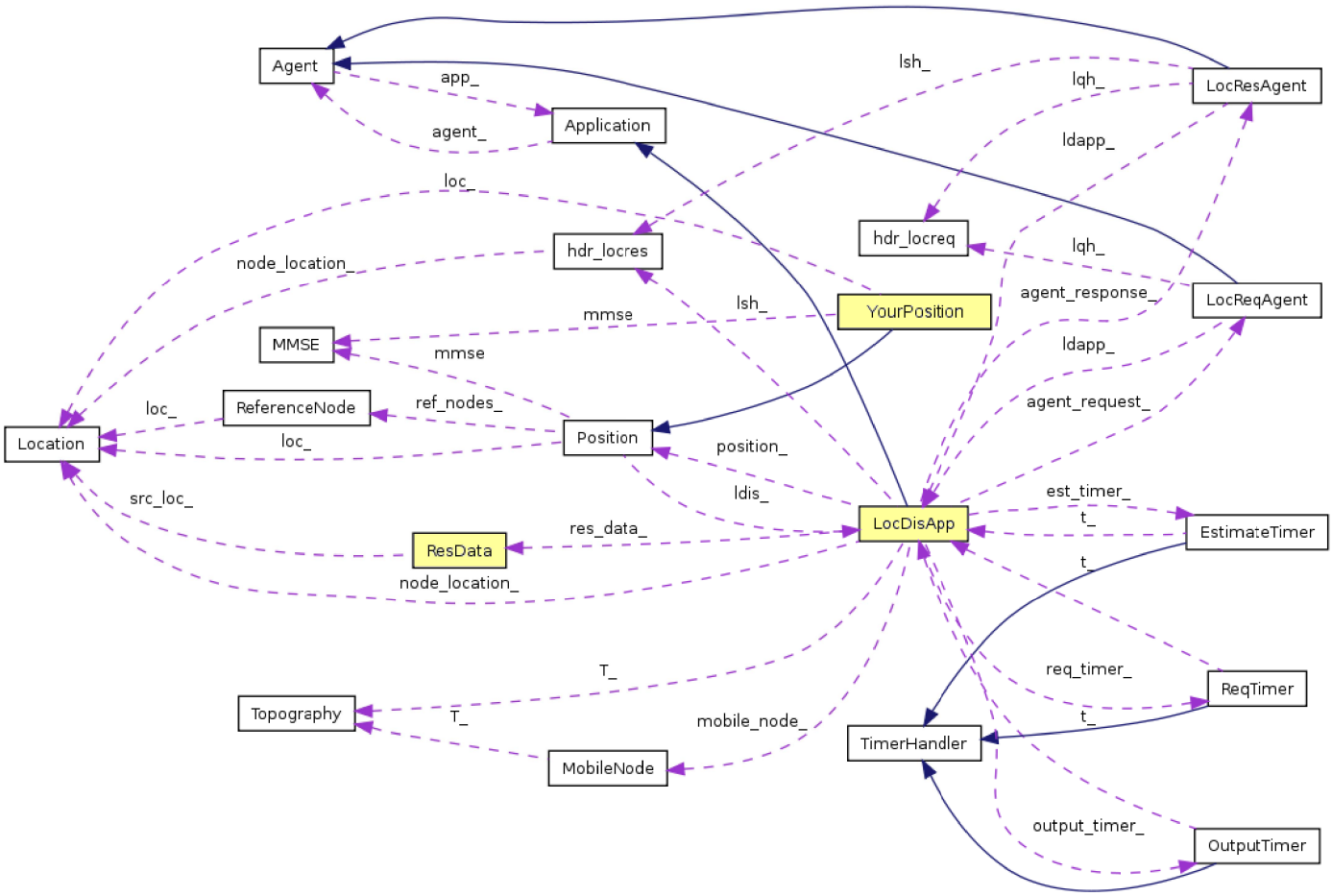

Figure 2. Collaboration diagram for the new classes

6)The Timer Classes

Three timer classes were created, which are the ReqTimer, EstimateTimer and OutputTimer classes. These classes are derived from the TimerHandler class. These timer classes collaborate with LocDisApp to schedule several tasks during the run time. The ReqTimer is used to moderate how frequently sensor nodes broadcast a "location request" packet. After sending the "location request" packet, the EstimateTimer is used to schedule the estimation process after a specific delay, which is required to give "location response" packets enough time to receive from neighboring references. The OutputTimer is used to schedule the action of recording the result, such as location error, number of references used and remaining energy, to the trace file.

7)ResData Class

This class is responsible for storing and retrieving the location information included in the "location response" packets received from the neighboring references. This information includes the address, the location and the power with which the packet is received.

B. The File Structure of the ns-2 Extension

Fig. 3 shows the file structure of the new ns-2, where the files under the "location" directory represent the new files that were added to ns-2, while the other files (left-hand side) are the modified files. The new classes that were discussed in the previous section are implemented in the files under the

"location" directory. In addition to these new files, some other files were modified as follows:

- *common/packet.h:* Two packet types were created in the locationpacket.h file using two structures (hdr_locreq and hdr_locres). In order to use these two types of packet, their corresponding packet types (PT_LOCREQ and PT_LOCRES) were defined in the packet.h file.
- *common/location.h:* This file contains the Location class, which is used to represent the location coordination (X, Y and Z) of nodes. Some methods were added to this class, such as the getter and the setter of an individual coordinate, is_equal( ) method to check if two locations are the same and distance( ) method to find the distance between two locations.

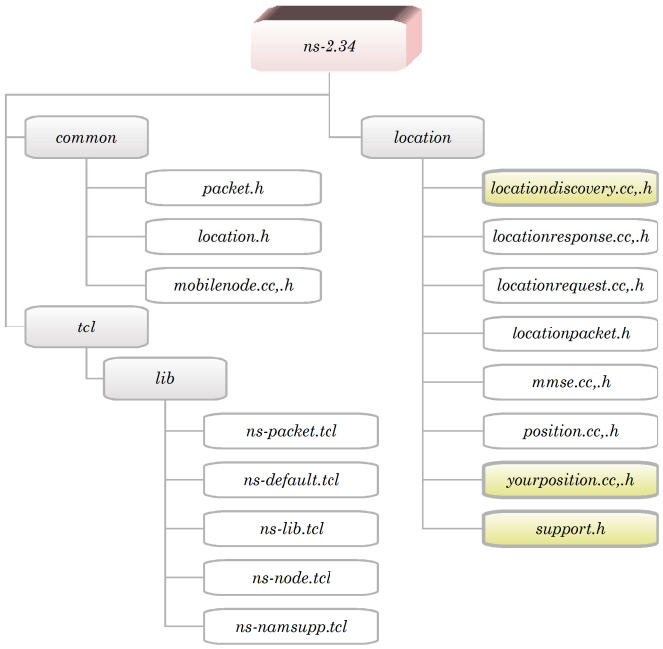

Figure 3. The structure of the new ns-2, showing the new files added to ns-2 (right-hand side) and the modified files (left-hand side)

- *common/mobilenode.cc, .h:* Two methods were added to these files. The first is to get an object to the topography. The second is to record the result in the trace file. The result could be the location error, number of references used and remaining energy.
- *tcl/lib/ns-packet.tcl:* In order to activate the new header classes, the OTcl class names were included in this file. The new OTcl header classes are "PacketHeader/LocReq" and "PacketHeader/LocRes", and so only "LocReq" and "LocRes" were added to the active protocol list.
- *tcl/lib/ns-node.tcl:* As mentioned before, from the localization perspective, the node could be a beacon, reference or unknown. In order to specify the type of nodes, a new instvar, called "nodeAttribute_", and a new instproc to get the node attribute, called "attribute", were created.
- *tcl/lib/ns-lib.tcl:* In order to enable the simulator to deal with the node attribute, an instproc was created to set the instvar "attribute_". Within the "Simulator instproc create-wireless-nodes" the node is allowed to set its attribute ($node set nodeAttribute_ $attribute_).
- *tcl/lib/ns-namsupp.tcl:* During the simulation, when the unknown nodes estimate their position they change their color (for instance to red). In order to enable the nodes to change their color after running the simulator, the "Node instproc color" was modified within this file.
- *tcl/lib/ns-default.tcl:* Sometimes it is necessary to bind some variables in both hierarchies (i.e. interpreted and compiled hierarchies). The default value of these bind variables is initialized in the ns-default.tcl file. Several variables were bound, such as the packet size, the request frequency (reqFreq_), the showColor_ variable to enable showing the color of the nodes based on their attribute and the subset_ variable to specify which localization algorithm should be applied.

C. Localization Procedures

The complete procedures of the localization process are as follows:

- LocDisApp schedules the OutputTimer with a specific delay (e.g. 1.0 second) to record the result into a trace file.
- LocDisApp schedules the ReqTimer with a specific delay, which determines how frequently the node broadcasts a "location request" packet.
- At the expiration time of ReqTimer, the LocDisApp invokes the LocReqAgent's method called broadcast( ) in order to broadcast a "location request" packet, schedules the EstimateTimer to start location estimation after a specific delay and reschedules the ReqTimer.
- LocReqAgent constructs a "location request" packet, and then it broadcasts the packet to the neighboring nodes.
- The LocResAgent of the reference nodes that received the "location request" packet requests the location information of the node from the LocDisApp. LocResAgent constructs a new "location response" packet, which includes this information, and sends it back to the requesting node.
- The LocResAgent of the requesting node receives the "location response" packets from neighboring references and then sends them to LocDisApp for more processing.

- LocDisApp extracts the required information from the packet received, namely the address and location of the sending reference node and the power with which the packet is received, and then stores this information in a ResData vector.
- At the expiration time of EstimateTimer the LocDisApp invokes the Position's method (or one of its child classes method) called estimate( ).
- Position estimates the node location using the data stored in the ResData vector based on the multilateration method.

D. Guidelines for Runing the Simulation

Using the ns-2 extension does not require new knowledge or writing a specific code to run the simulator. Normal users who have the basic knowledge to run a simple wireless network using ns-2 are able to write a simple OTcl script to simulate the proposed localization system. This section gives some guidelines for configuring the localization simulation. It assumes that the reader is familiar with setting up wireless mobile network simulations in ns-2. Therefore, it will not explain the entire simulation procedures, rather it will show how to configure nodes to simulate localization. However, the reader is referred to [8] for some tutorials about configuring wireless networks.

At the beginning of the simulation there are only two types of nodes, beacons and unknowns. Each of them has a different configuration, as shown in Fig. 4. Beacons already know their location, so they should be attached only with LocResAgent. Unknowns should be attached with LocReqAgent in order to allow them to broadcast location request messages, also they should be attached with LocResAgent to handle the recipient packets (which are "location request" and "location response" packets). These two types of agents should be attached to LocDisApp.

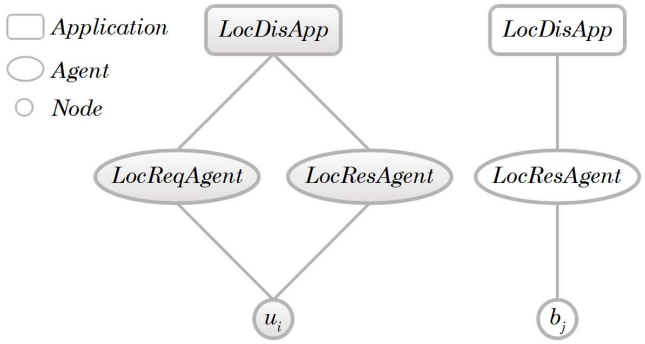

Figure 4. Node configuration, where u_i is an unknown node and b_j is a beacon node

Before creating the nodes it is required to specify the nodes' configuration, such as the routing protocol, MAC type and so on. In addition to these configurations it is also required to specify the attribute of the nodes (either BEACON or UNKNOWN), which can be done with the help of the command "node-config". For example, the following code shows how to create beacon nodes:

```
set val(nn)     10      ;# Number of nodes
set val(nb)     3       ;# Number of beacons
# Beacon nodes:
# Nodes configuration
set val(attr)   BEACON  ;# Node attribute
$ns_ node-config -attribute $val(attr)
# Nodes creation
for {set i 0} {$i < $val(nb)} {incr i} {
        set node_($i) [$ns_ node]
}
```

The next step is to create the required agents and application based on the configuration shown in Fig. 4.

```
# Beacon nodes have only response agent
for {set i 0} {$i < $val(nb)} {incr i} {
        # Setup the response agent
        set lres_($i) [new Agent/LocRes]
        $ns_ attach-agent $node_($i) $lres_($i)
        # Setup the location discovery application
        set ldis_($i) [new Application/LocDis]
        $ldis_($i) attach-agent $lres_($i)
}
```

Finally, the location discovery applications should be started at a specific time:

```
# Start the locdis applications
for {set i 0} {$i < $val(nn)} {incr i} {
        $ldis_($i) set random_ 1
        $ldis_($i) set subset_ 3
        $ldis_($i) set showColor_ 1
        $ns_ at 0.0 "$ldis_($i) start"
}
```

If the user does not want to use the default value of the bind variables he can change the setting of these variables before starting the applications. For instance, in the previous code the random_ variable was set to 1 to all the LocDis applications to start broadcasting the location request messages at a random time instead of starting immediately (random_ = 0). The subset_ variable specifies the localization algorithm that should be applied to estimate the node location; the value of three refers to the ALWadHA algorithm. The variable showColor_ is used in all the unknowns to change their color after they estimate their location; setting this variable to zero disables this feature.

E. Implementing New Localization Algorithms

Implementing new localization algorithms using the ns-2 extension is a straightforward process. Assuming that one need to implement a new algorithm called "Your", then the following procedures should be followed:

- Create two new files "yourposition.h" and "yourposition.cc" under the "location" directory (see Fig. 3), and then modify the ns-2 "Makefile" by adding the corresponding object file name of the new module (-I./location/yourposition.o)
- Derive a new class called YourPosition from the Position class, and then implement the new functions and features which not provided by the Position class.

- In the file "support.h" assign a number for the new algorithm (e.g. #define YOUR 10). If the new algorithm requires extra information for localization, then this information should be added to ResData vector within this file.
- To enable the user to run the new algorithm by specifying the value of the subset_ variable to "10" (as explained in Section II.D) add these few lines to the switch case in "locationdiscovery.cc" file:

```
switch (subset_)
{
        ........
        case YOUR:
           position_ = new YourPosition(this);
           break;
}
```

- Finally recompile ns-2.

III. Simulation Result

To evaluate the effectiveness of the presented ns-2 extension, six localization algorithms have been implemented for the performance comparison, using the same assumptions. Two algorithms based on basic multilateration method were implemented. The first one is based on the single-estimation approach (called M_Single), while the second one is implemented based on the successive refinement approach (called M_Refine). The other algorithms are Nearest [3], CRLB (Cramer-Rao-Lower-Bound) [4], NDBL (node distribution-based localization) [5] and ALWadHA [1].

Twelve beacons and 64 unknowns were distributed randomly in a $200\,m \times 200\,m$ filed. Several experiments were performed. At each experiment the simulation was run 100 times; the duration of each run was 600 sec (the total duration was 60 000 sec) and at each run nodes were redistributed randomly in different places (using a different seed value). RSS was used to measure the distance between nodes. However, to simulate noise, each measured distance was disturbed by a normal random variant with the following settings: a mean of 0.1% of the measured distance and a standard deviation of 1% of the measured distance. The implemented algorithms are evaluated in terms of the location error, number of references used and remaining energy.

A. Localization Error

The mean error is estimated every second for all knowns as a ratio of transmission range. The mean error at a specific time t is equal to the summation of the location error of all knowns divided by the number of these knowns and then it is divided by the transmission range (r_{tx}) as follows:

$$Mean\ error_t = \left(\frac{1}{n} \sum_{i=1}^{n} \| \hat{z}_i - z_i \| \right) \frac{1}{r_{tx}} 100\% \quad , \qquad (1)$$

where n is the total number of knowns at a specific time t, z_i is the actual node's location and $\hat{z}_i$ is the estimated node's location. Fig. 5 shows the mean error of the implemented algorithms.

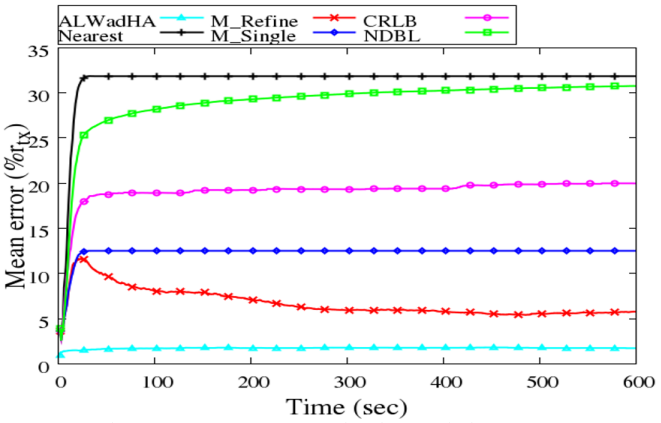

Figure 5. Mean error as a ratio of transmission range

B. Number of References

Increasing the number of references could enhance the accuracy of the position estimation. On the other hand it increases the complexity of computations. The multilateration method uses all the available references. The number of references for the Nearest and CRLB algorithms is predefined manually by three references. The ALWadHA and NDBL algorithms specify the number of references dynamically at each iteration, based on a specific criterion. The average number of references used at a specific time t is calculated as follows:

$$\#\,of\ References_t = \frac{1}{n} \sum_{i=1}^{n} C(S_i) \quad , \qquad (2)$$

where n is the total number of knowns at a specific time t, S_i is the subset of references used to estimate the location of node i and $C(Si)$ is the cardinality of set S_i. Fig. 6 shows the average number of references used by each algorithm.

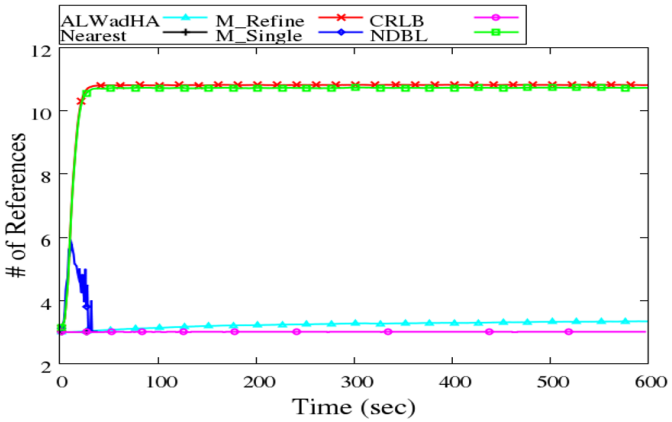

Figure 6. Average # of references

C. Remaining Energy

At the beginning of the simulation each node has 2.0 joule. Fig. 7 shows the average remaining energy vs time considering only energy consumption due to communication.

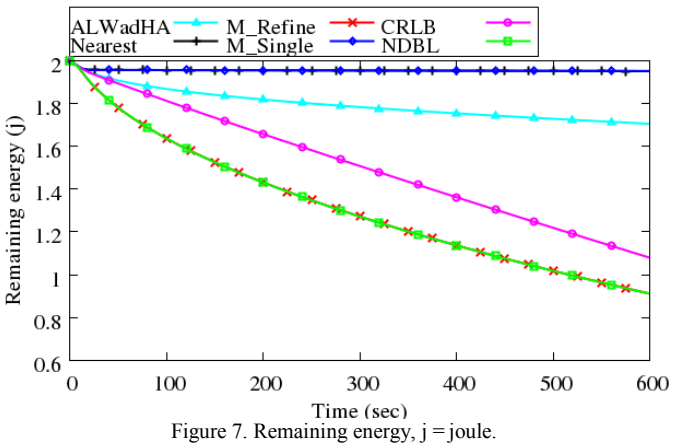

Figure 7. Remaining energy, j = joule.

IV. Conclusion

This paper presents an extension to the current version of ns-2. This extension allows normal users with a basic ns-2 knowledge to simulate localization system in wireless sensor networks, acting as a simple starting point for implementing new localization algorithms. As an example, the presented ns-2 extension is used to implement and evaluate several localization algorithms. The content of this paper should be of interest to researchers who want to implement new or existing localization algorithms. As a secondary contribution, considering the limited academic publications available, this paper should also be of interest to anyone who is new to ns-2 and who wishes to know more about how a simulation project is built and structured. We also hope that this paper will encourage practitioners in other areas of WSN research will develop similar ns-2 extensions.

References

[1] A. M. Abu-Mahfouz and G. P. Hancke, "An efficient distributed localization algorithm for wireless sensor networks: Based on smart reference-selection method," submitted for publication.

[2] S. Chinnappen-Rimer and G. P. Hancke, "Perimeter echo algorithm for network localization," in *Proceedings of the IEEE AFRICON 2009,* September 23 - 25, Nairobi, Kenya, 2009, pp. 1-5.

[3] K. Y. Cheng, V. Tam and K. S. Lui, "Improving aps with anchor selection in anisotropic sensor networks," in *Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services — ICAS-ICNS '05,* October 23-28, Papeete, Tahiti, 2005, pp. 49-54.

[4] D. Lieckfeldt, J. You and D. Timmermann, "An algorithm for distributed beacon selection," in *Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communication — PerCom '08,* March 17-21, Hong Kong, China, 2008, pp. 318-323.

[5] S. Han, S. Lee, S. Lee, J. Park and S. Park, "Node distribution-based localization for large-scale wireless sensor networks," *Wireless Networks,* vol. 16, no. 5, pp. 1389-1406, 2010.

[6] A. Savvides, C. C. Han and M. B. Strivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking — MobiCom '01,* July 16-21, Rome, Italy, 2001, pp. 166-179.

[7] T. Issariyakul and E. Hossain, *Introduction to Network Simulator (NS2).* New York, NY, USA: Springer, 2009.

[8] "The network simulator - ns-2," 12 February 2010, http://nsnam.isi.edu/nsnam/index.php/User_Information. Last accessed on 26 November 2010 .

[9] "OMNET++ simulator," 09 March 2011, http://www.omnetpp.org/. Last accessed on 02 April 2011 .

[10] P. Levis, N. Lee, M. Welsh and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st International Conference on Embedded Networked Sensor System — SenSys '03,* November 05-07, Los Angeles, CA, USA, 2003, pp. 126-137.

[11] K. Fall and K. Varadhan, "The Ns Manual," 9 May 2010, http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf.

[12] P. Morávek, "Ns-2 simulator capabilities in nodes localization in wireless networks, " 2009, http://www.feec.vutbr.cz/EEICT/2009/sbornik/03-Doktorske %20projekty/01-Elektronika%20a%20komunikace/06-xmorav08.pdf. Last accessed on 26 November 2010 .

[13] "The doxygen documentation system," 12 October 2010, http://www.stack.nl/~dimitri/doxygen/. Last accessed on 26 November 2010 .