How to Extend NS-2 for Wireless Sensor Networks Localization

Abdelhady M. Naguib

Abstract— Localization for wireless sensor networks (WSN) is concerned with determining the location of sensor nodes and it is very important for many WSN applications. The cost and difficulties of building a WSN in a real world makes simulation an essential tool for analyzing and studying the performance of these networks. Little work in this area has been using NS-2, so that this paper aims to present detailed steps for modifying and adding new modules for NS-2 to implement a localization algorithm. In addition, this paper evaluate the performance of a localization algorithm under certain performance metrics.

Index Terms— Centroid, Localization, NS-2, Simulation, WSN.

1 Introduction

IT can be stated that WSNs are distributed networks with composed of many tiny-sized, low-cost, battery-powered nodes. Due to these features, WSNs have a widespread using for many applications in human life such as smart homes, industry controlling, habitat monitoring and disaster recovery [1-3]. However, the location of these nodes is unknown and many applications needs the knowledge of this location such as data-centric storage [4]. Other WSNs applications require precise knowledge of nodes' location, such as the geographical routing technique [5, 6], network security [7] and energy efficient management [8]. Localization has many benefits other than determining location, such as improving WSN efficiency [9].

Due to the alternation of sensor node modes (e.g. wake up and sleep) and the change of environment parameters such as channel noise, solving a localization problem mathematically is very difficult and gives inaccurate results. Hence, simulation models considered another technique to solve a localization problem. This done by changing simulation inputs and observing the resulting outputs. In order to design and develop new localization models for WSNs, there is a strong need to use an effective simulation tool. MATLAB is considered a powerful package, but fails to provide outputs close to realistic results; Which can be obtained through proper implementations and good simulation tools [10]. Many works in [11-14] provide localization models, that developed and deployed using GloMosim, OMNET++ and OPNET Modeler. However, a little work has done using NS-2 [15].

The capability that nodes can self-determine their locations considered one of the basic services of this self-localization method [16]. Localization algorithms for wireless sensor networks divided into two categories: range-based and range-free localizations. Range-based algorithms require precise information on the distances and angles between nodes. However, according to wireless sensor networks, this is not suitable because of high cost needed to obtain such information. In contrast, range-free localization schemes use estimated distances between anchors (known location either by GPS or manually) and unknown nodes, resulting in much lower costs than for range-based schemes. Although the localization error in this case is a little large, it can still meet the requirements for most applications. Thus, range-free algorithms are now the most common type used for localization [17].

The aim of this paper is to provide the following contributions: first, extending NS-2 to implement a localization algorithm for wireless sensor networks. Second, give a detailed description for the process of modeling centroid localization. Finally, evaluate its performance using simulation-based analysis. The rest of this paper is organized as follows: section 2, provides an overview of centroid algorithm. Section 3, explains how NS-2 can be extended to implement a localization algorithm for wireless sensor networks. Section 4, describes the modeling process of class hierarchies added to NS-2. Section 5, evaluates the performance of centroid algorithm using the extended NS-2. Finally, section 6 concludes this paper.

2 Centroid Algorithm Overview

The Centroid localization algorithm [18] was proposed by N. Bulusu et.al. This algorithm assumes that a set of anchor nodes (A_i, $1 < i < n$), with overlapping regions of coverage, exist in the deployment area of the WSN. The main idea is to average the position of anchor nodes, located at (X_i, Y_i) according to the following formula:

$$\left(X_{est}, Y_{est}\right) = \left(\frac{x_1 + x_2 + \ldots + x_i}{N}, \frac{y_1 + y_2 + \ldots + y_i}{N}\right) \quad (1)$$

Where (X_{est},Y_{est}) is the estimated position of sensor node and N is number of anchor nodes. An example of how the Centroid algorithm works is shown in Fig. 1, where a sensor node N_k (unknown position) is within communication range to four anchor nodes, A_1~A_4. The node N_k localizes itself to the centroid of the quadrilateral $A_1A_2A_3A_4$ (for the case of a quadrilateral, the centroid is at the point of intersection of the

• *Abdelhady Mohammad Naguib is currently Assistant Professor, Systems and Computers Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt, PH-20-01060660785. E-mail: a_naguib@azhar.edu.eg*

bi-medians (the lines connecting the middle points of opposite sides). Sensor node N_k is located at point (X_a, Y_a) and receives four packet from anchors A_1, A_2, A_3 and A_4 and its estimated location is given by (X_{est}, Y_{est}). As shown in figure there is a gap between actual location and estimated location of sensor node. This gap is called localization error LE which can be calculated according to the following formula:

$$LE = \sqrt{(X_{est} - X_a)^2 + (Y_{est} - Y_a)^2} \qquad (2)$$

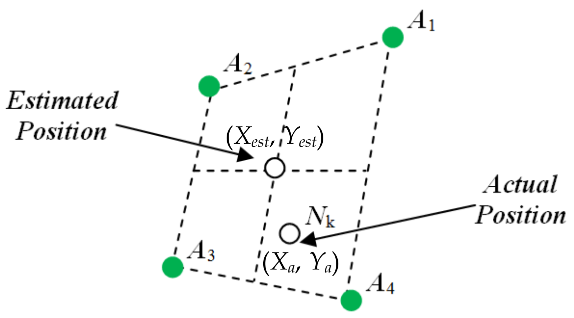

Fig. 1. Centroid Localization.

The following section introduces an introduction to NS-2 languages and presents detailed steps of how to extend and adding new modules to NS-2 for implementing a localization algorithm finally, a description for every class added.

3 Extending NS-2

NS-2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While C++ defines the internal mechanism (i.e., a backend) of the simulation, OTcl on the other hand sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend) [19]. The C++ and the OTcl are linked together using TclCL. After simulation, NS-2 outputs text-based simulation results known as trace file. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used. To analyze a particular behavior of the network, users can extract a relevant subset of text-based data and transform it to a more conceivable presentation.

To create an executable file, all C++ codes need to be compiled and linked. Since the body of NS-2 is large, the time of compilation is not negligible. On the other hand, OTcl is an interpreted programming language, not a compiled one. Any change in an OTcl file can be executed without compilation. Each line needs more execution time in OTcl, because it does not convert the codes into machine language. C++ is a language suitable for running a large simulation since it is fast to run, but it is slow to change. On the other hand, OTcl is suitable for parameter configurator because it is fast to change, but it is slow to run. Hence, NS-2 is constructed by combining the advantages of these two languages.

In this paper NS-2.34 is extended and new modules are added so as to implement a localization algorithm (e.g. Centroid). The new classes that were added to the NS-2 are shown in Fig. 2. These classes can be divided into two types. The first one, standalone classes, which are, **RepData**, **Place** and **Centroid** classes. These classes are used only from the C++ domain. The second type is, compiled hierarchy classes, which are **LoczApplication**, **LoczReqAgent** and **LoczRepAgent** classes. In order to access these classes from the OTcl domain, they should be linked to the corresponding interpreted hierarchy classes as follows, **Application/LoczApplication**, **Agent/LoczReq** and **Agent/LoczRep**, respectively. In the following, a description for every module added are stated.

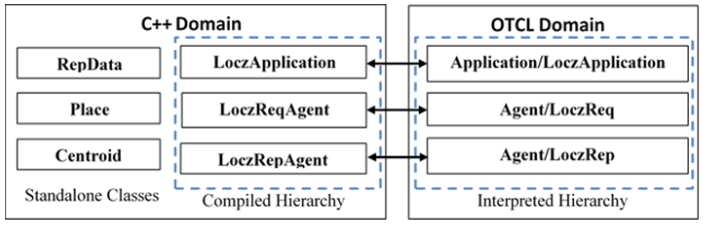

Fig. 2. Extending NS-2 by adding new modules.

3.1 RepData Class

This class is used to capture the data that is required by the localization algorithm. Generally this data include the *address* and *location* of the anchors/reference that send the localization reply packets, and the *power* at which these packets are received.

3.2 Place Class

This class is the base class for any localization algorithm that will be implemented. This class contains functions that are responsible for the following; calling a localization algorithm (e.g. centroid), extracting the required data from the received information of the anchors/references and making sure that the estimated location is within the boundary of the topography.

3.3 Centroid Class

This class is responsible for implementing centroid algorithm by applying the steps stated in section 2.

3.4 LoczReqAgent Class

It is derived from the **Agent** class built into NS-2. **LoczReqAgent** is responsible for creating a localization request packet (**PT_LOCZREQ**) and then broadcasting it to neighboring nodes. This agent should be attached only to unknown sensor nodes because anchors already know their position.

3.5 LoczRepAgent Class

LoczRepAgent class is considered a child of base class **Agent**. It is responsible for receiving packets from neighboring sensor nodes. This agent should be attached to all nodes (unknowns and anchors/references). Two types of packets could be received. The first is a localization request packet (**PT_LOCZREQ**). If this type of packet is received by an anchor or reference node it constructs a localization reply packet (**PT_LOCZREP**) that includes location information and then sends it to the requesting node. Unknown nodes receiving this type of packet simply de-allocate it. The second is a localization reply packet (**PT_LOCZREP**). The requesting sensor node that receives this packet sends it to the application layer (**LoczApplication**), which processes the included location information to estimate the node's position.

3.6 LoczApplication Class

This class is derived from the **Application** class. Each sensor node in the network attaches an object from this class to its agent(s). The **LoczApplication** class has several functions, such as periodically broadcast a localization request packet (**PT_LOCZREQ**) by invoking the broadcast function of **LoczReqAgent**, Extracting the localization information from the received localization reply packet (**PT_LOCZREP**) then store it in a **RepData** vector and finally, estimating the node location.

3.7 Interpreted hierarchy

In order to access the newly compiled hierarchy classes **LoczApplication**, **LoczReqAgent** and **LoczRepAgent** from the OTcl domain, these classes were mapped and linked to corresponding OTcl classes, which are **Application/LoczApplication**, **Agent/LoczReq** and **Agent/LoczRep**, 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 lrep [new Agent/LoczRep]** will create a new object of class **LoczRepAgent**. The following section illustrates the modeling process of class hierarchy of the new classes added to NS-2.

4 Modeling New Class Hierarchies Added to NS-2

For simplifying the process of modeling classes added to NS-2, only the new classes were included. These classes are: the one derived from (i.e. parent classes) and the classes used by these new classes. According to collaboration diagrams, solid lines represent the inheritance between classes; for example **Class1** → **Class2** means **Class1** is derived from **Class2**. When a class is using a method and/or member of another class, dotted lines are used.

4.1 Collaboration Diagram for Centroid Class

The collaboration diagram for **Centroid** class is shown in Fig. 3. **Centroid** class uses the **ReferenceNode** structure, which consists of *location* variable to store the location of anchor/reference neighboring nodes.

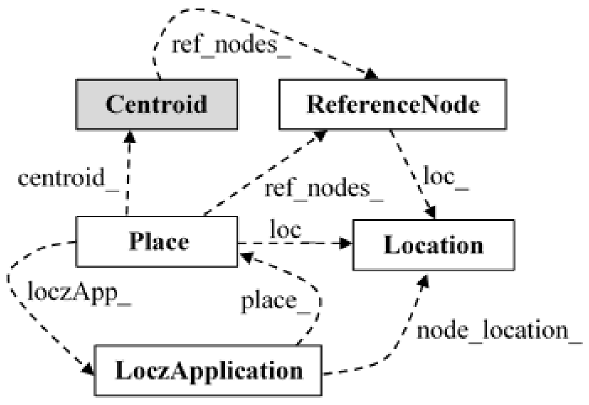

Fig. 3. Collaboration diagram for Centroid class.

The localization information (*location*) of neighboring anchor/reference nodes is stored into array of this structure (**ref_nodes_**). **Place** class uses the **Centroid**, **LoczApplication** and **Location** (NS-2 built in class) classes and **ReferenceNode** structure. The **X**, **Y** and **Z** coordinates of sensor node are represented by **Location** class.

4.2 The Timer Classes

Timer class is a module that can be used to delay actions. After Timer has been started for a given period of time (i.e., at the expiration), it takes actions unless cancelled or restarted. As shown in Fig. 4, three timer classes were created, which are the **OutputTimer**, **ReqTimer**, and **EstimateTimer** classes. The parent class **TimerHandler** is the parent class for these timer classes. As shown in Fig. 4, there is a collaboration between these classes **LoczApplication** class so as to schedule tasks during the execution time.

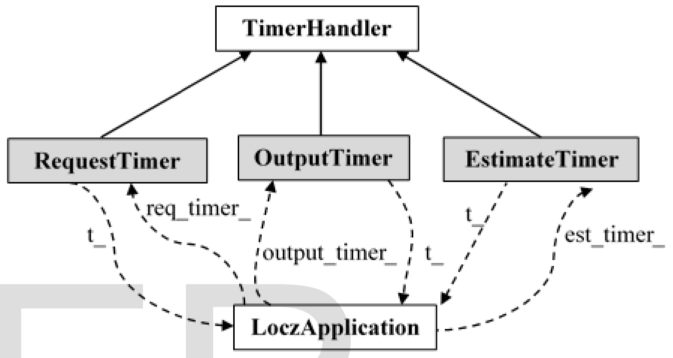

Fig. 4. Collaboration diagram for Timer classes.

The main purpose of **OutputTimer** is to schedule the action of recording the result to the trace file. The result printed to trace file is as follows: *estimated position*, *localization error*, *number of references used*, *actual node position* and *remaining energy*. In order to moderate how frequently sensor nodes broadcast a location request packet, the **ReqTimer** is used. The **EstimateTimer** is used to schedule the estimation process after a specific delay (i.e. after sending the location request packet). This time delay is required to give location reply packets enough time to receive from neighboring anchor/reference nodes.

4.3 LoczReqAgent and LoczRepAgent Classes

In NS-2 packets consist of packet header and packet payload. NS-2 rarely uses data payload in simulation. To avoid any complication it is suggested not to use data payload in simulation. In order to develop new protocol, a protocol-specific header (**PSH**) type is developed to be used in packets. Doing so allows a new protocol implementation to avoid overloading already-existing header fields. Protocol-specific header consists of two levels; the first one allocates spaces for PSHs and uses an offset concept, the second level consists of the attributes of a **PSH,** use C++ *struct* data type and main field is offset_ which refers to the distance between the beginning of packet header and that of a protocol-specific header. In the C++ domain, protocol-specific headers are declared but not instantiated. Therefore, NS-2 uses a *struct* data type (rather than a class) to represent protocol-specific headers. No constructor is required for a protocol-specific header.

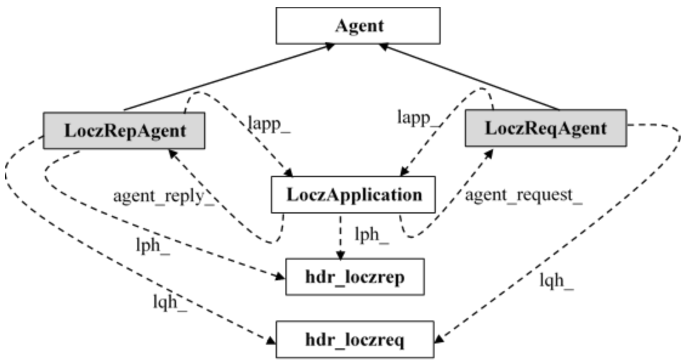

Fig. 5. Collaboration diagram for LoczReqAgent & LoczRepAgent classes.

LoczReqAgent and **LoczRepAgent** classes are derived from the **Agent** class (built in NS-2) as shown in Fig. 5. Localization request packet (**PT_LOCZREQ**) is constructed and broadcasted by **LoczReqAgent**. In order to handle the packets received, which could be localization request or localization reply packets, **LoczRepAgent** is responsible for this. Two new types of packet were created: the first one, a localization request packet (**PT_LOCZREQ**), which uses the new **PSH** defined in the structure **hdr_loczreq**, and the second one, a localization reply packet (**PT_LOCZREP**), which uses the new **PSH** defined in the structure **hdr_loczrep**. **LoczReqAgent** uses only **hdr_loczreq** to construct localization request packets. According to **LoczRepAgent** it uses **hdr_loczreq** so as to access and process the received localization request packets; also it uses the **hdr_loczrep** to construct the localization reply packets.

4.4 LoczApplication Class

As shown in Fig. 6, the **LoczApplication** class is derived from the **Application** class (which is built in NS-2). **LoczApplication** class collaborates with many classes: three timers, two agents, **Location** and **Place** classes. In order to gain access to the received localization reply packets, **LoczApplication** class use the **hdr_loczrep** header structure so as to process the included localization information and estimate the node position.

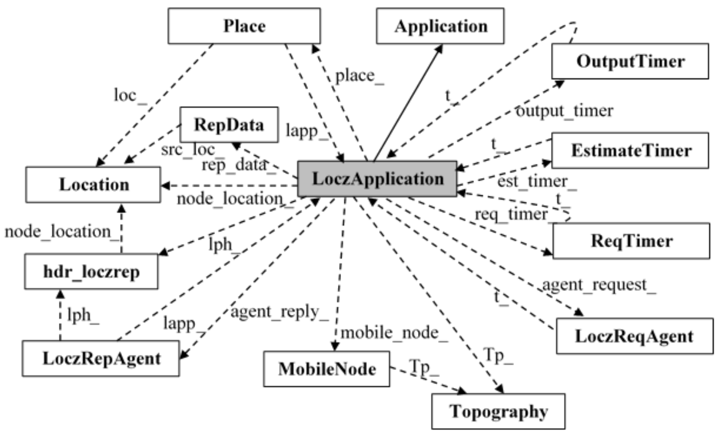

Fig. 6. Collaboration diagram for LoczApplication class.

A vector of class **RepData** is used to store the localization information received from neighboring anchors/references. Therefore, **LoczApplication** class uses this vector. This information stored in this vector are the *address*, *location* and the *power* with which the packet was received.

4.5 The Complete Procedure of the Localization Process

In the following, the overall steps for performing the localization process using the new modules added to NS-2 stated as follows:

1. The **OutputTimer** is scheduled with a specific delay (e.g. 1.0 second) to record the result into a trace file using **LoczApplication** class.
2. **LoczApplication** also schedules the **ReqTimer** by invoking the timer member function **sched(delay_)**, which is used to specify when to start broadcasting localization request either directly or after a random time.
3. The **LoczReqAgent**'s method called **broadcast( )** is invoked at the expiration time of **ReqTimer** by the **LoczApplication** class in order to broadcast a localization request packet, the **EstimateTimer** is scheduled to start position estimation after a specific delay finally **LoczApplication** reschedules the **ReqTimer** so as to repeat the process again.
4. A localization request packet (**PT_LOCZREQ**) is constructed by **LoczReqAgent** and then it broadcasts the packet to the neighboring anchor/reference nodes.
5. When **LoczRepAgent** of the anchor/reference nodes receives the localization request packet, it requests from the **LoczApplication** the localization information of the requesting node (i.e. the node's location). After that, a new localization reply packet (**PT_LOCZREP**) is constructed by **LoczRepAgent**, which includes this information, and sends it back to the requesting node.
6. After the requesting node (unknown location) receives the localization reply packets from neighboring anchors/references through the **LoczRepAgent**, it sends them upstream to **LoczApplication** class.
7. After that, **LoczApplication** class will extract the required information from the packet received, namely the *address*, *location* and the *power* with which the packet is received, and then stores this information in a **RepData** vector.
8. The **LoczApplication** will invoke the **estimate( )** method of the **Centroid** class, at the expiration time of **EstimateTimer**.
9. Finally, **Centroid** class estimates the node position using the data stored in the **RepData** vector, by applying the steps stated in section 2.

4.6 The New Structure of NS-2

By now, the new structure of NS-2 is composed of two file categories. The first category, files that added to NS-2 and the second category is NS-2 built in files, which modified only.

Fig. 7 illustrates these categories as follows: the files under the localization directory represent the new modules that added to NS-2, while the other files (on the left-hand side) are the modified ones. Under the *localization* directory resides the implemented files of the new classes that discussed in the previous sections.

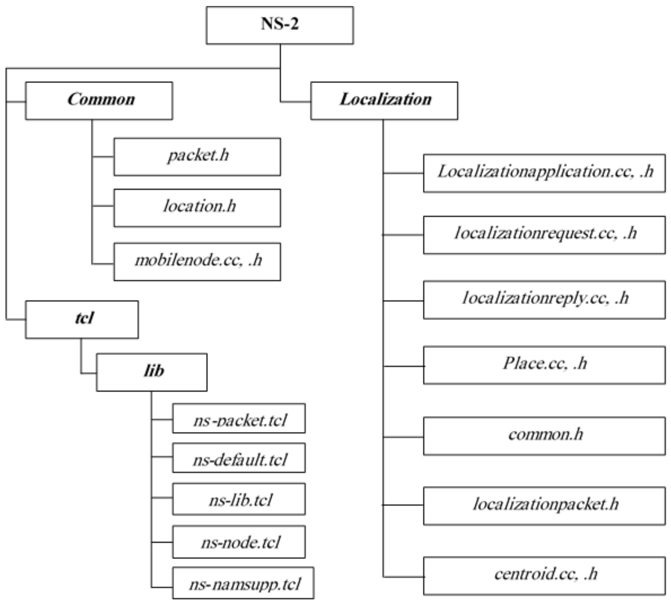

Fig. 7. New Structure of NS-2

Beside these new files, some other files modified under the common and tcl directories as follows:

- ***common/packet.h***: In the **localizationpacket.h** file, two new packet types were created using two structures (**hdr_loczreq** and **hdr_loczrep**). In order to use these two types of packet, their corresponding packet types (**PT_LOCZREQ** and **PT_LOCZREP**) were defined in the **packet.h** file.
- ***common/location.h***: This file contains the **Location** class, which is used to represent the position coordinates (**X, Y** and **Z**) of nodes.
- ***common/mobilenode.cc, .h***: Two methods were added to these files. The first one was added to *mobilenode.h* so as to get the topography of mobile node. The second is to log localization information into the trace file as follows: L (stands for localization event), time, node energy, number of used anchors/references, localization error (percentage of radius) finally (x, y) coordinates of the estimated and actual positions.
- ***tcl/lib/ns-packet.tcl***: two new packet headers added to this file, **LoczReq** and **LoczRep** in order to activate the new header structures **hdr_loczreq** and **hdr_lozrep**.
- ***tcl/lib/ns-node.tcl***: a new OTCL *instvar*, called *nodeAttribute_*, and a new OTCL *instproc*, called *attribute* created in order to specify the type of nodes (anchor, reference or unknown) and to get the node attribute respectively.
- ***tcl/lib/ns-lib.tcl***: within the *tcl* file created for configuring the wireless sensor network and running simulation, the following statement: ***$node set nodeAttribute_ $attribute_*** is written so as to specify node attribute (anchor, reference or unknown). Hence, an *instproc* was created into *ns-lib.tcl* to set the *instvar attribute_* enables the simulator to deal with the node attribute.
- ***tcl/lib/ns-namsupp.tcl***: so as to change node's color from blue to red (i.e. this node estimated its position), the Node instproc *color* was modified within this file to enable changing the node's color after running the simulator.
- ***tcl/lib/ns-default.tcl***: this file is used to initialize the bound variables of both interpreted and compiled hierarchies. Many variable were bound such as, *packetSize_* for both **LoczReq** and **LoczRep** agents, *reqFreq_* which is used to specify the frequency of node broadcasting a localization request packet and *showColor_* to enable(1) or disable(0) showing the color of the nodes according to their attribute (anchor, reference or unknown).

5 Performance Evaluation of Centroid Localization using Extended NS-2

This section evaluates Centroid algorithm under extended NS-2 environment. Performance metrics used in evaluating performance are as follows: looking at the effects of nodes' deployment, nodes' density and network size on the localization error and the number of anchors used. It will examine the impact of increasing number of anchor nodes on the accuracy of the centroid algorithm and the number of localized sensor nodes (coverage).

5.1 Average Localization Error

The average localization error (*ALE*) at a specific time *t* is equal to the summation of the localization error of all localized sensor nodes, divided by the number of these localized nodes. After that, divided by the transmission range as show in the following formula:

$$ALE = \left(\frac{1}{n}\sum_{i=1}^{n} LE_i\right)\frac{1}{R}*100 \qquad (\%) \qquad (3)$$

Where n, number of localized sensor nodes, LE_i is localization error for sensor node i computed from formula (2) and R is the transmission range for sensor node.

5.2 Setup and Environment

Centroid localization algorithm evaluated using the extended NS-2. The deployment field divided into several regions; in order to satisfy the applicability of the centroid algorithm in large-scale WSNs. The size of region is 50x50 m^2 and each region contains the same number of unknown sensors and anchors without ignoring the randomness of node distribution. All nodes had a limited transmission range (R) of 50 m.

At each experiment the simulation was run 10 times; the duration of each run was 200 sec (the total duration was 2000 sec), and at each run nodes were redistributed randomly in different places (using a different seed value). Two types of deployment were used: random and grid deployment using either static or mobile sensors which creates four scenarios. The characteristics of all scenarios are shown in table 1 and simulation parameters are shown in table 2.

TABLE 1

CHARACTERISTICS OF ALL SCENARIOS

Parameter	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Topology Type	Random	Random	Grid	Random
Topology Size	100 x 100	200 x 200	200 x 200	320 x 320
Mobility Model	Static	Static	Static	Random Waypoint
Number of regions	4	16	---	---
Rows x Columns	---	---	10 x 8	---
Number of unknown nodes	100	208	72, 68, 64, 60, 56, 52, 48 and 44	100
Number of anchors/region	4, 5, 6, 7 and 8	2, 3, 4, 5, 6 and 7	---	---
Total number of anchors	16, 20, 24, 28 and 32	32, 48, 64, 80,96 and 112	8, 12, 16, 20, 24, 28, 32 and 36	10, 20, 30, 40, 50, 60, 70 and 80

TABLE 2

SIMULATION PARAMETERS

Parameter	Value
Channel type	Channel/WirelessChannel
Radio-propagation model	Propagation/FreeSpace
Network interface type	Phy/WirelessPhy
MAC type	Mac/802_11
Interface queue (ifq) type	Queue/DropTail/PriQueue
Link layer type	LL
Antenna model	Antenna/OmniAntenna
Max packets in interface queue	50
Routing protocol	AODV
Energy model	EnergyModel
Transmitting power in watts	0.281838
Receiving power in watts	0.281838
Initial energy in Joules	20.0
Receive sensitivity threshold	7.69113e-08 watt
Carrier sense threshold	5.3352e-06 watt
Transmitter antenna gain	1
Receiver antenna gain	1

5.3 Simulation Results and Discussions

Simulations presented in the following sub-sections are based on two performance metrics: average localization error and coverage. According to average localization error, it is computed using formula (3) as a percent of transmission range (%R). The second metric is coverage; it is computed as the number of localized sensor nodes. Sensor node is said to be localized when it receives localization reply packets from more than or equal to 3 anchor nodes (i.e. **PT_LOCZREP ≥** 3).

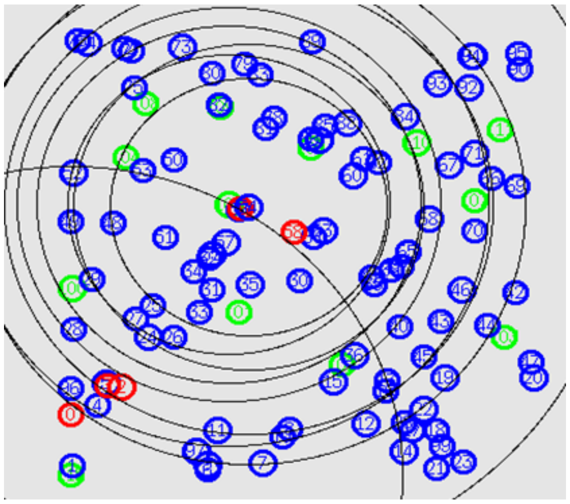
Fig. 8. First scenario (100x100)

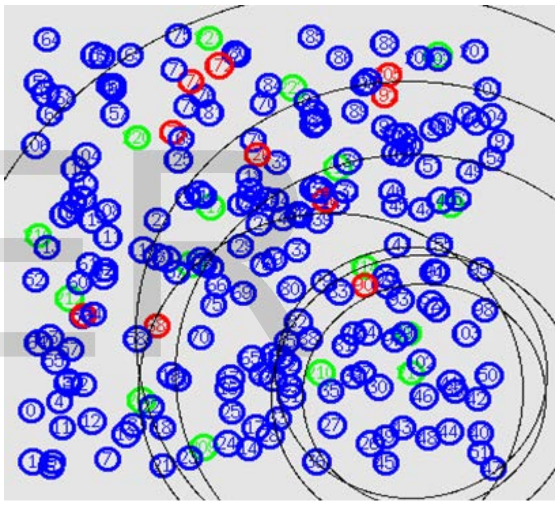
Fig. 9. Second scenario (200x200)

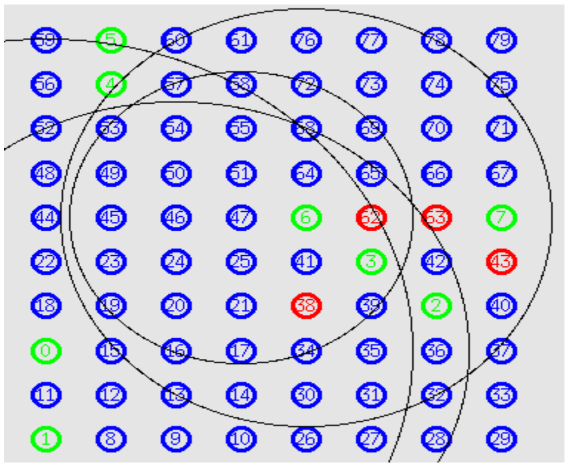
Fig. 10. Third scenario (Grid)

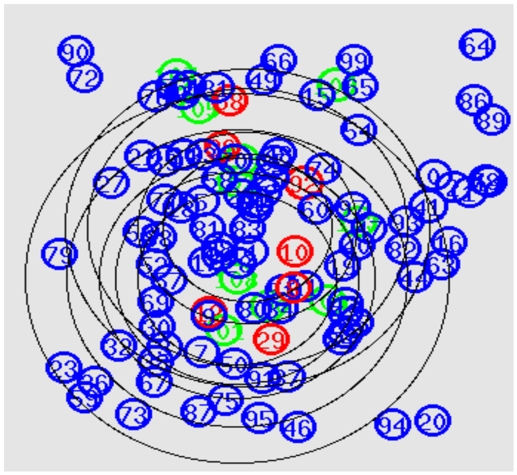
Fig. 11. Fourth scenario (Mobile 320x320)

There are three types of sensor nodes: unknown, anchor and localized nodes, by using NAM as a visualization tool there are three colors used for each node type as shown in figures 8, 9, 10 and 11. Unknown nodes have a blue color; anchor nodes have a green color and localized nodes have a red color. In the following sub-sections, four scenarios are simulated and evaluated as follows.

5.3.1 First Scenario

This scenario has the following characteristics: Random static deployment of nodes, area size is 100x100, 4 regions (each 50x50) and 100 unknown sensor nodes. The impact of varying number of anchor nodes on localization accuracy and coverage is shown in the following figures.

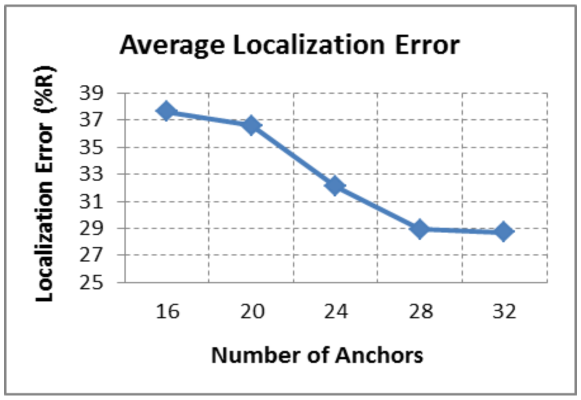

Fig. 12. Average Localization Error

As shown in Fig. 12, as the number of anchor nodes increases the localization error decreases. That is due to the reduction in the size of the possible region for unknown nodes to be located. For example, when number of anchors equals 16 (i.e. 4 anchors per region), the localization error is 37.6 % and when number of anchors equals 32 (i.e. 8 anchors per region), the localization error drops to 28.7 %. The impact of varying number of anchors on the number of localized sensor nodes is shown in Fig. 13. As the number of anchor nodes increase, number of localized sensor nodes increases also.

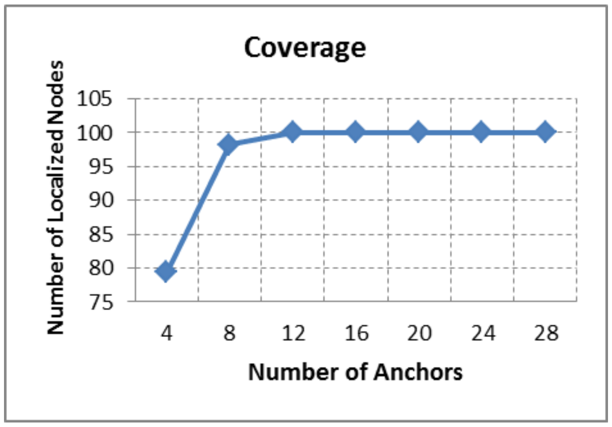

Fig. 13. Coverage

5.3.2 Second Scenario

The characteristics of this scenario are as follows: Random static deployment of nodes, area size is 200x200, 16 regions (each 50x50) and 208 unknown sensor nodes. The impact of varying number of anchor nodes on localization accuracy and coverage are shown in the following figures.

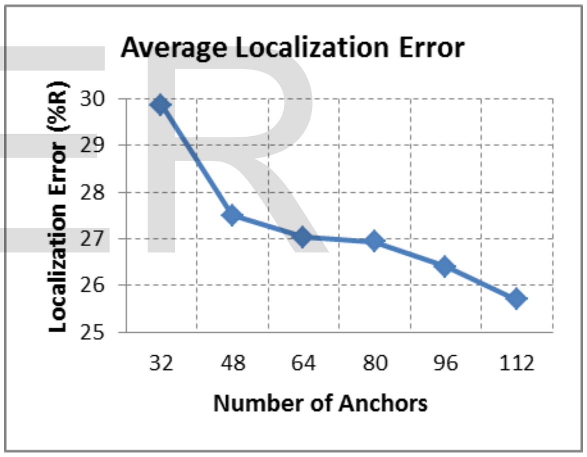

Fig. 14. Average Localization Error

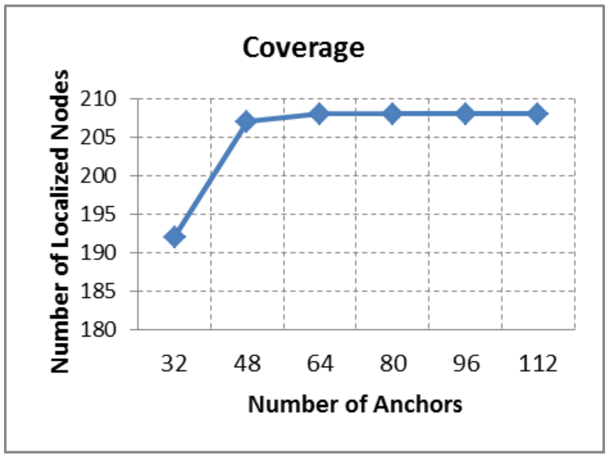

Fig. 15. Coverage

As shown in Fig. 14, when number of anchors equals 32 (i.e. 2 anchor/region), the localization error is 29.9 % and when increasing it to 112 anchors (7 anchor/region), the localization error drops to 25.7 %. Fig 15 shows the impact of varying number of anchors on the number of localized sensor nodes. From this figure when number of anchor nodes equal 32 (2 anchor/region), number of localized sensor nodes equals 192 out of 208 and when number of anchor nodes increases to 112 (7 anchor/region), the number of localized sensors increases to 208 (full localization).

5.3.3 Third Scenario

This scenario has the following characteristics: Grid static deployment of nodes, area size is 200x200 and all nodes are deployed into a grid of 10x8 (rows x columns) yielding 80 nodes. The influence of varying number of anchor nodes and number of unknown nodes on localization accuracy and coverage are shown in the following figures.

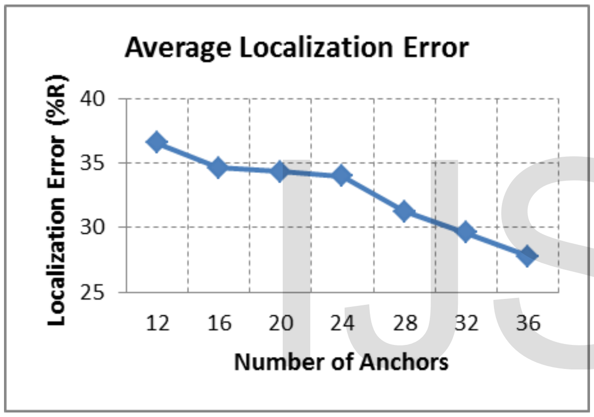

Fig. 16. Average Localization Error

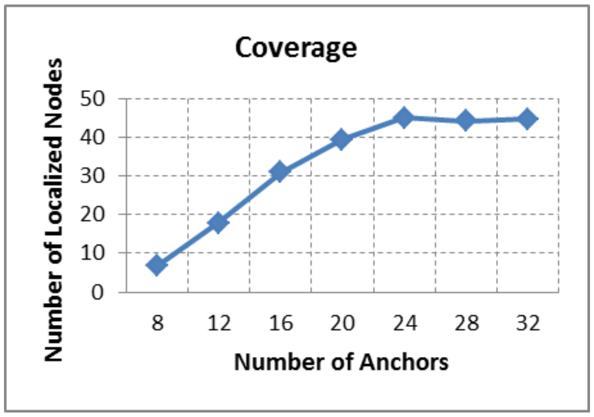

Fig. 17. Coverage

As shown in Fig. 16, when number of anchor nodes increase, the localization error decreases. For example when number of anchor nodes equals 12 the localization error = 36.6 % and when number on anchor nodes increases to 36 the localization error drops to 27.8 %. According to Fig. 17, when number of anchor nodes = 8, number of localized sensor nodes = 7 and when number of anchor nodes increases to 32, number of localized sensor nodes increases to 45.

5.3.4 Fourth Scenario

The characteristics of this scenario are as follows: Random mobile deployment of nodes, area size is 320x320, nodes moves according to random waypoint mobility model generated by BonnMotion [20] scenario generator. It has been observed that with the Random Waypoint model, nodes have a higher probability of being near the center of the simulation area [20]. So that they are initially uniformly distributed over the simulation area by cutting off the initial phase (i.e. additional seconds at the beginning of the scenario should be skipped). The impact of varying number of anchor nodes on localization accuracy and coverage are shown in the following figures.

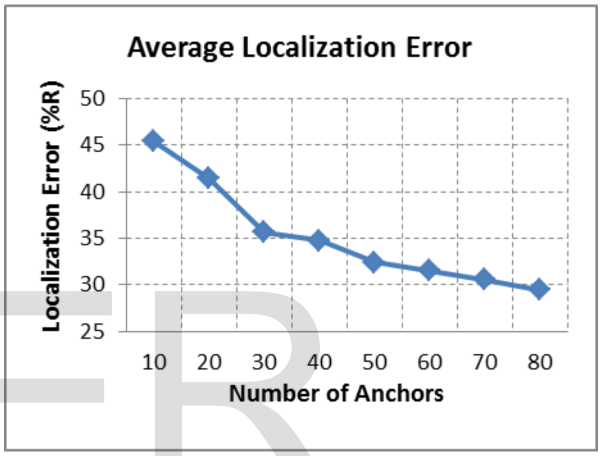

Fig. 18. Average Localization Error

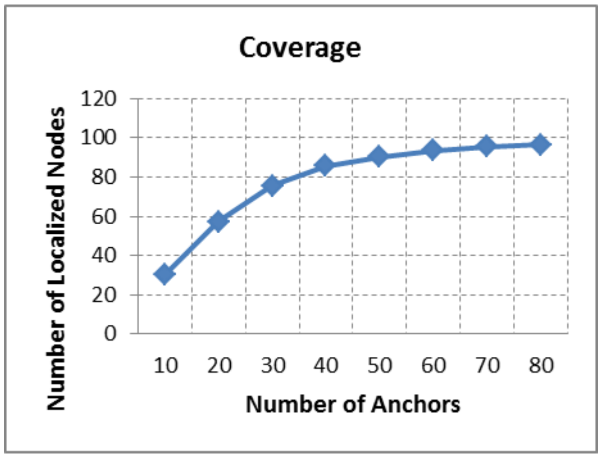

Fig. 19. Coverage

As shown in Fig. 18, when number of anchor nodes equals 10, the localization error = 45.4 % and when number of anchor nodes increases to 80, the localization error drops to 29.4 %. From Fig. 19, when number of anchors = 10, the number of localized sensor nodes = 30 and when number of anchor nodes reaches 80, the number of localized sensor nodes increases to 98.

6 Conclusion

To inspect the characteristics of WSNs, there are several flexible features contained into NS-2, which encourage researchers to use it. However, to implement and evaluate localization algorithms, the current NS-2 version should be extended and new modules should be added. Hence, this paper focus on how to extend NS-2 in order to simulate a localization algorithm for wireless sensor networks. The extended NS-2 is ready to combine any localization algorithm by merging its files with existing modules added and modified.

In addition, this paper illustrates the steps for implementing centroid algorithm by combining its files with the added and modified NS-2 modules. Finally, this paper evaluates the performance of centroid algorithm using four different scenarios. According to results obtained, under different topology types, size and different deployment of sensor nodes (Random, Grid and Mobile), as the number of anchor nodes increase, the localization accuracy increase and number of localized sensor nodes increase also.

References

[1] J. P. Sheu, P. K. Sahoo, C. H. Su and W. K. Hu, “Efficient path planning and data gathering protocols for the wireless sensor network,” *Computer Communications*, vol. 33, no.3, pp. 398-408, 2010.

[2] N. Xiong, M. Cao, A. Vasilakos, L. Yang and F. Yang, “An energy-efficient scheme in next-generation sensor networks,” *International Journal of Communication Systems*, Volume 23, pp. 1189-1200, Numbers 9-10, 2010.

[3] F. Bai, K. Munasinghe and A. Jamalipour, “Accuracy, latency, and energy cross-optimization in wireless sensor networks through infection spreading,” *International Journal of Communication Systems*, Volume 24, Number 5, pp. 628-646, 2011.

[4] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, “GHT: A geographic hash table for data-centric storage,” *In Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications — WSNA* '02, Atlanta, Georgia, USA, pp. 78-87, 28 September 2002.

[5] B. Karp and H. Kung, “GPSR: Greedy perimeter stateless routing for wireless networks,” *In Proceedings of the 6th Annual International Conference on Mobile Computing and Networking – MobiCom '00*, Boston, Massachusetts, USA, pp. 243-254, 06-11 August 2000.

[6] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, “A scalable location service for geographic ad hoc routing,” *In Proceedings of the 6th Annual International Conference on Mobile Computing and Networking – MobiCom '00*, Boston, Massachusetts, USA, pp. 120-130, 06-11 August 2000.

[7] Y. C. Hu, A. Perrig, and D. B. Johnson, “Packet leashes: A defense against wormhole attacks in wireless networks,” *In Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies – INFOCOM '03*, San Francisco, California, USA, vol. 3, pp. 1976-1986, 30 March – 03 April 2003.

[8] Y. Xu, J. Heidemann, and D. Estrin, “Geography-informed energy conservation for ad hoc routing,” *In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking – MobiCom '01*, Rome, Italy, pp. 70-84, 16-21 July 2001.

[9] A. Srinivasan and J. Wu, “Wireless sensor networks (WSNs): Secure localization,” *Encyclopedia of Wireless and Mobile Communications*, pp. 1545-1571, April 2008.

[10] S. Sinha, Z. Chaczko, and R. Klempous, “SNIPER: A Wireless Sensor Network Simulator,” Springer-Verlag Berlin Heidelberg, 2009.

[11] A. Molina and G. Alba, “Location discovery in Wireless Sensor Networks using meta-heuristics,” *Applied Soft Computing 11*, 1223–1240, 2011.

[12] A. Caballero, F. Merino, L. Gil, P. Maza and I. Ollero, “A probabilistic framework for entire WSN localization using a mobile robot,” *Robotics and Autonomous Systems 56*, 798–806, 2008.

[13] D. Nezhad, A. Miri and A. Makrakis, “Location privacy and anonymity preserving routing for wireless sensor networks,” *Computer Networks 52*, 3433–3452, 2008.

[14] H. Wang and H. Cheng, “Simulation and Modeling for Centroid Algorithm Using OPNET in Wireless Sensor Networks,” *CCIS 238*, pp. 1–10, Springer-Verlag Berlin Heidelberg, 2011.

[15] The Network Simulator, NS–2 [Online]. Available: http://www.isi.edu/nsnam/ns/, 2016.

[16] S. Liang, L. Liao, and Y. Lee, “Localization Algorithm based on Improved Weighted Centroid in Wireless Sensor Networks,” *Journal of Networks*, vol. 9, no. 1, January 2014.

[17] X. Lai, S. X. Yang, G. Zeng, J. She and M. Wu, “New Distributed Positioning Algorithm Based on Centroid of Circular Belt for Wireless Sensor Networks,” *International Journal of Automation and Computing, 04(3)*, 315-324, DOI: 10.1007/s11633-007-0315-x, July 2007.

[18] N. Bulusu, J. Heidemann and D. Estrin, “GPS-less Low-cost Outdoor Localization for Very Small Devices,” *IEEE Personal Communications*, vol. 7, no. 5, pp. 28–34, 2000.

[19] T. Issariyakul, E. Hossain, *Introduction to Network Simulator NS2*, Springer New York Dordrecht Heidelberg London, 2012.

[20] BonnMotion, A Mobility Scenario Generation and Analysis Tool, University of Osnabr¨uck, July 8, 2013.