Pymote 2.0: A Simulation Framework designed for Wireless Ad-hoc and Sensor Networks

Farrukh Shahzad
Department of Computer Engineering
King Fahd University of Petroleum and Minerals, Dhahran, KSA.
Email: farrukhshahzad@kfupm.edu.sa

Simulation has always been very popular among network-related research. A large number of simulators have been proposed in literature in which algorithms for mobile ad hoc networks (MANET) or wireless sensor networks (WSNs) can be implemented and studied. These simulators have different design goals and largely vary in the level of complexity and included features. They support different hardware and communication layers assumptions, focus on different distributed networks implementations and environments, and come with a different set of tools for modeling, analysis, and visualization. Classical simulation tools include NS-2, OMNeT++, J-Sim, OPNET, TOSSIM, and others [1][2][3].

The development of the larger and complicated wireless network requires that the design concepts are checked and optimized using simulation [4]. The simulation environment can either be an adaptive development or a new development. The adaptive development includes simulation environments that already existed before the idea of WSNs emerged. These simulation environments were then extended to support wireless functionality and adapted for the use with WSNs. In contrast, new developments cover new simulators, which were created solely for simulating WSNs, considering sensor specific characteristics from the beginning [1].

Simulators can be divided into three major categories based on the level of complexity:

- algorithm level,
- packet level, and
- instruction level.

Some algorithm level simulators are [5], [6], [7] and [8].

The motivation for this work is based on the author's quest for a simulation tool for his PhD research. The idea was to find a easy to use tool which is expendable as needed. After some research, we have concluded that a Python-based tool called Pymote [9] somewhat fulfills our requirements.

I. Introduction to Pymote

We decided to use Pymote, a high level Python library for event based simulation of distributed algorithms in wireless adhoc networks [9]. The library allows the user to make implementation of their ideas using Pythona popular, easy to learn, full featured, object oriented programming language. Functionalities provided by the library are implemented without additional layer of abstraction, thus harnessing full power of Python's native highly expressive syntax. Using the library, users can quickly and accurately define and simulate their algorithms.

Following is a simple python script for simulating 'Flood' message among few nodes using Pymote. Figure 1 shows the generated topology.

```
1  from pymote import *
2  net_gen = NetworkGenerator(degree=2, n_count=10)
3  net = net_gen.generate_homogeneous_network()
4  from pymote.algorithms.broadcast import Flood
5  net.algorithms = ( (Flood, {'informationKey':'I'}), )
6  some_node = net.nodes()[0]
7  some_node.memory['I'] = 'Hello'
8  sim = Simulation(net)
9  sim.run()
10 net.savefig(fname=__name__)
```

Listing 1. A simple python script for simulating 'Flood' message

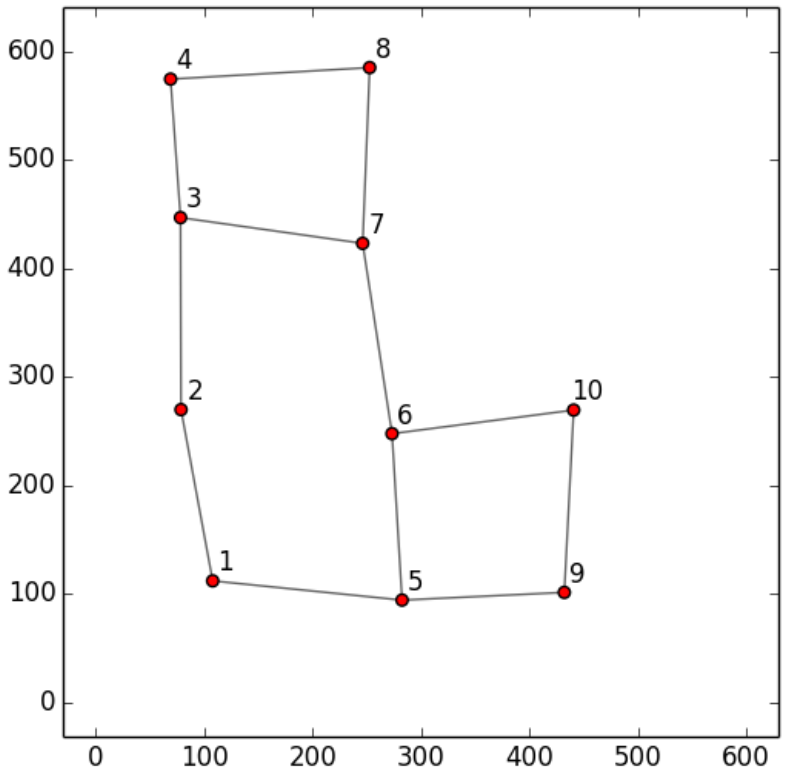


Figure 1. A simple 10-nodes WSN topology using Pymote

A. *Why Python*

1) Easy to learn, well documented and quick prototyping;
2) Full featured object oriented language.;
3) Simple and highly expressive: keeps the code clean and minimal, making its usage straightforward;
4) Support for interactive mode allow experimentation and analysis (Ipython);
5) Introspection features, namely, easy programmatic inspection of all defined objects properties;
6) Rich scientific functions library and strong support for scientific computing and calculations (NumPy and SciPy, matplotlib);
7) Promote open source reproducible research.

B. *Pymote Features*

1) Pymote is a Python package for event based simulation and evaluation of distributed algorithms [9].
2) Pymote is designed to allow rapid interactive testing of new algorithms, their analysis and visualization while minimizing developers time.
3) Current implementation is targeted for wireless sensor network.
4) It supports both interactive algorithm simulation and automation of experiments and provides visualization tools for both.
5) It has been deliberately kept simple, easy to use, and extensible.
6) The source code along with the documentation is available as an open source project at https://github.com/darbula/pymote.

II. Pymote 2.0 Extension

In this section, we explain one of our contribution related to this research which is the design and implementation of packet level modules for propagation, energy consumption and mobility models to extend the Pymote framework. We also added graphing and data collection modules to enhance the Pymote base functionality and modified existing modules for node, network, algorithm and logging.

Table I
PROPAGATION MODEL PARAMETERS

Description	Parameter	Default
Transmit Antenna gain	G_TX	1
Receive Antenna gain	G_RX	1
System Loss (¿=1.0)	L	1
Min. Received signal power threshold	P_RX_THRESHOLD	-70 dbm
Frequency	FREQ	2.4 Ghz
Path loss exponent	BETA	2.5
Gaussian noise standard deviation	SIGMA_DB	4 dbm

A. Propagation Model

We implemented two basic radio propagation models and the commonly used shadowing model for WSN in the Pymote framework. These models are used to predict the received signal power of each packet. At the physical layer of each wireless node, there is a receiving threshold (P_RX_THRESHOLD). When a packet is received, if its signal power is below the receiving threshold, it is marked as error and dropped by the MAC layer. The free space propagation model assumes the ideal propagation condition that there is only one clear line-of-sight path between the transmitter and receiver, while the two-ray ground reflection model considers both the direct path and a ground reflection path which gives more accurate prediction at a long distance than the free space model. However, in reality, the received power is a random variable due to multipath propagation or fading (shadowing) effects. The shadowing model consists of two parts: path loss component and a Gaussian random variable with zero mean and standard deviation DB, which represent the variation of the received power at certain distance. Table I lists parameters available for propagation module. The propagation model type (free space, two-ray ground or shadowing) is a network level attribute, which should be selected before starting the simulation.

B. Energy consumption Model

In our extended framework, the energy model object is implemented as a node attribute, which represents the level of energy in a node. Each node can be configured to be powered by external source (unlimited power), Battery (default) or energy harvesting (EH) sources. The energy in a node has an initial value which is the level of energy the node has at the beginning of the simulation. It also has a given energy consumption for every packet it transmits and receives which is a function of packet size, transmission rate and transmit (receive) power. The model also supports idle or constant energy discharge due to hardware/ micro-controller consumption and energy charge for energy harvesting based WSN. During simulation, each nodes available energy is recomputed every second based on the charging and/or discharging rate. If it drops below minimum energy required to operate (Emin) then that node assumed to be dead (not available for communication) until energy reaches above Emin again later in simulation (for EH nodes). Table II lists parameters available for energy module which can be set differently for each node. The energy object keeps track of the energy available (for battery-operated or energy harvested nodes) and total energy consumption.

C. Mobility Model

Our extended framework allows nodes to be mobile during simulation. Each node can be configured as fixed or mobile. The mobility module support three types of motion as summarized in Table III [10][11]. During simulation, each mobile node location is recomputed every second.

D. Plotting and Data collection

These modules allow real-time plotting and data collection during and after simulation for interactive analysis and comparisons of useful information. The modules implements generic helper methods and utilizes the python Matplotlib package [12]. The simulation script is responsible for utilizing these methods to plot/chart and collect/log appropriate information as required by the simulated algorithm and application scenario. The output files are managed by utilizing separate folder for each type of files within the current working path (TableIV). Also for

Table II
ENERGY MODEL PARAMETERS

Description	Parameter	Default
Transmit power	P_TX	84 mW
Receive power	P_RX	73 mW
Initial Energy	E_INIT	2.0 Joules
Min. Energy required for operation	E_MIN	0.5 Joules
Charging rate (EH nodes)	P_CHARGING	2 mW/sec
Discharging rate	P_IDLE	0.1 mW/sec
Transmission rate	TR_RATE	250 kbps

Table III
MOBILITY PARAMETERS

Type	Parameters	Default
0: Fixed	None	
1: Mobile (uniform velocity)	VELOCITY HEADING	20 m/s, 45 deg
2: Mobile (uniform velocity, random heading)	VELOCITY	20 m/s
3: Mobile (random motion)	MAX_ RANDOM_ MOVEMENT	30 m

each simulation run, a separate folder, prefixed with the current date time is used for all files created during that simulation run. The output format includes CSV, PNG, and high quality SVG and PDF which can directly by inserted into Latex and other publishing applications. HTML files are also created with embedded JavaScript for interactive plotting needed for presentation and on-line content. It utilizes the innovative charting library provided by Highsoft [13], which is free to use for personal and academic purposes.

E. Modified Node module

Enhanced framework requires significant modification in the Node module. The Node object now contains node type, energy model object and mobility object. The modified send and receive methods check before transmission or reception whether node has enough energy to perform the operation. Also the propagation model dictates whether a packet is received without errors (i.e. when received signal power is greater than the threshold based on the distance between the sender and receiver nodes). The object also keeps track of number of messages transmitted, received, or lost.

III. A COMPLETE SIMULATION EXAMPLE

In this section we will provide a compete simulation example with line-by-line explanation of the python script created for the simulation.

Table IV
FILE MANAGEMENT

Type	Folder Name	Examples
Data files	/data	CSV files with energy consumption,etc.
Charts/plots	/charts	Line and/or bar charts of energy levels, etc.
Topology	/topology	Topology used before/after simulation
Logging	/logs	Simulation run and module level logging
Combined	/yyyy-mm-ddThh-mm-ss	All files generated during a specific run

A. System requirements

1) Step 1: Download and install the latest Python 2.7.x from https://www.python.org/download/releases/2.7/ (don't install 3.x release). Windows, Linux and Mac are supported.
2) Step 2: Download and install any suitable IDE (Recommended Pycharm from https://www.jetbrains.com/pycharm/downloa
3) Step 3: Fork or download the latest Pymote framework from https://github.com/farrukhshahzad/pymote2.0). Open the pymote in the IDE.

B. Simulation script

The simulation script is shown below. The script includes the comments for each step and provides a guide to write scripts for other type of simulations. Some of the output from this script are shown in Fig. 2 - 6.

```
"""
    This is a python script to show a complete simulation example.
    In this example, the famous range-free localization algorithm called 'DV-hop'
    is simulated. Follow the comments in the script,
"""

'''First we need to include required packages'''

# built-in packages
import time

# external packages
import scipy as sp
import numpy as np
from numpy.random import seed
from numpy import sqrt

# Internal packages
from pymote import *
from pymote.conf import global_settings
from pymote import propagation
from toplogies import Toplogy
from pymote.utils import plotter
from pymote.utils.filing import get_path, load_metadata,\
     DATA_DIR, TOPOLOGY_DIR, CHART_DIR, DATETIME_DIR

from pymote.simulation import Simulation
from pymote.sensor import TruePosSensor
from pymote.networkgenerator import NetworkGenerator
from pymote.algorithms.niculescu2003.dvhop import DVHop
from pymote.algorithms.niculescu2003.trilaterate import Trilaterate

'''Start of Script'''

# get the pymote version and print it
meta = load_metadata()
print meta['name'], meta['version']

seed(123)  # to get same random sequence for each run so that simulation can be reproduce

# Network/Environment setup
global_settings.ENVIRONMENT2D_SHAPE = (600, 600)  # desired network size for simulation
net = Network()  # Initiate the network object
h, w = net.environment.im.shape  # get the network width and height(should be as above)

n = 100  # total no of nodes
p_anchors = 20  # No. of anchors in %age
c_range = 100  # communication radii of each node
degree = 10   # Desired degree or connectivity (how many nodes are in range)


# Start out with empty arrays(list) to be filled in later after simulation ends
```

```
xpositions = []
xestpositions = []
deltapos = []
esterror = []
positions = []
newpos = []
anchpositions = []
message_stats = []
position_stats = []
consume = []
energy = []

# Network Topology setup
Node.cid = 1  # start node id

net_gen = NetworkGenerator(n_count=n, degree=degree)
net = net_gen.generate_homogeneous_network()  # A random homogeneous topology

#net_gen = Toplogy(n_count=n, degree=degree, n_max=n, n_min=n, connected=False)
#net = net_gen.generate_grid_network(randomness=0.2)

# Computes no. of anchors
f_anchors = (int)(100 / p_anchors)
n_anchors = (int)(n *  p_anchors/100.0)

# Set some nodes as anchor based on number of desired anchors
# Two arrays are populated with location of nodes to be plotted later
for node in net.nodes():
    xpositions.append(net.pos[node][0])
    if (node.id % f_anchors==0):  # anchor nodes
        node.compositeSensor = (TruePosSensor,)
        node.type = 'C'  # Anchors
        anchpositions.append({'x': net.pos[node][0], 'y': net.pos[node][1],
                              'name': str(node.id), 'color': 'red',
                              'marker': {'symbol': 'circle', 'radius': '8'}})
    else:
        positions.append({'x': net.pos[node][0], 'y': net.pos[node][1],
                          'name': 'Node: ' + str(node.id)})


avg_deg = round(net.avg_degree())

# set the network name based on no. of Nodes, degree and Comm. Range
net.name = "%s - Nodes=%s, Avg degree=%s, Range=%s" \
           % (net_gen.name, net.__len__(), int(avg_deg), int(node.commRange))

# Save the network topology as a PNG image (other options are pdf, svg)
net.savefig(fname=get_path(DATETIME_DIR, net.name),
            title=net.name, format='png',
            x_label="X", y_label="Y", show_labels=False)


# Now select the algorithm for simulation
net.algorithms = ((DVHop, {'truePositionKey': 'tp',
                                  'hopsizeKey': 'hs',
                                  'dataKey': 'I'
                                  }),
                  (Trilaterate, {'truePositionKey': 'tp',
                                        'hopsizeKey': 'hs',
                                        'positionKey': 'pos',
                                        'dataKey': 'I'}),
                   )

start_time = time.time()
# simulation start
sim = Simulation(net)
```

```
sim.run()
# simulation ends

end_time = time.time() - start_time
print("Execution time:  %s seconds ---" % round(end_time,2) )

# Now data capturing/analysis and visualization
k=0
err_sum=0.0
total_tx = 0  # Total number of Tx by all nodes
total_rx = 0  # Total number of Rx by all nodes
total_energy = 0  # Total energy consumption by all nodes

for node in net.nodes():
    total_tx += node.n_transmitted
    total_rx += node.n_received
    err=0
    message_stats.append([node.id, node.n_transmitted,
                              node.n_transmitted_failed_power,
                              node.n_received,
                              node.n_received_failed_power,
                              node.n_received_failed_loss
                              ])
    consume.append(round(node.power.energy_consumption * 1000.0, 2))
    energy.append(node.power.energy * 1000.0)
    total_energy += node.power.energy_consumption

    if node.type == 'N' and 'pos' in node.memory:
        act = node.get_dic()['Info']['position']
        est = node.memory['pos']
        newx = est[0]
        newy = est[1]
        err = sqrt(sum(pow(act - est, 2)))
        print node.id, node.type, act, est, err
        position_stats.append((node.id, newx, newy, err))
        err_sum += err
        k += 1
        newpos.append({'x': newx, 'y': newy,
                  'name': 'Node: ' + str(node.id)})
    elif node.type == 'C':
        newx = net.pos[node][0]
        newy = net.pos[node][1]
    else:
        newx = -1
        newy = -1
    xestpositions.append(newx)
    esterror.append(err)
    deltapos.append(net.pos[node][0] - newx)

# Summary of simulation result/Metrics
comments = "Anchors: " + str(n_anchors) + " = " +str(p_anchors) +"%"+ \
           ",  Runtime(sec): "+ str(round(end_time,2)) + \
           ",  Tx/Rx per node: " + str(round(total_tx/net.__len__())) + "/" + \
                                       str(round(total_rx/net.__len__())) + \
           ",  Energy/node (mJ): " + str(round(1000*total_energy/net.__len__(), 3)) + \
           ",  Avg. error: " + str(round(err_sum/k, 2))


print ("%s nodes localized, %s" % (k, comments))
sim.reset()


# Create CSV, plots and interactive html/JS charts based on collected data
np.savetxt(get_path(DATETIME_DIR, "message_stats-%s.csv" % n),
           message_stats,
           delimiter=",", fmt="%8s", comments='',
```

```
            header="Node,transmit,transmit_failed_power,"
                   "received,received_failed_power,"
                   "received_failed_loss")

sp.savetxt(get_path(DATETIME_DIR, "localization_error-%s.csv" % n),
           position_stats,
           delimiter=",", fmt="%8s", comments='',
           header="Node,actual,estimated,error")

sp.savetxt(get_path(DATETIME_DIR, "energy-%s.csv" %n),
               list(zip(range(1, n+1), energy, consume)),
               delimiter="\t", fmt="%s",
               header="Nodes\tEnergy Left (mJ)\tConsumed", comments='')

# Create html/JS file for network Topology
plotter.gethtmlScatter(xpositions, [anchpositions, positions, newpos],
                fname="Topology-"+net.name, folder=DATETIME_DIR,
                xlabel="X", ylabel="Y", labels=['Anchor','Regular','Localized'],
                title="Topology-"+net.name, open=1, range={'xmin':0, 'ymin':0, 'xmax': w, 'ymax': h},
                comment=comments, show_range=str(int(node.commRange)),
                plot_options=["color: 'red', visible: false,", "color: 'blue',", "color: 'green', visi

plotter.gethtmlLine(range(1,len(xpositions)), [xpositions, xestpositions, deltapos, esterror],
                fname="X-"+net.name, folder=DATETIME_DIR,
                xlabel="Node", ylabel="meters", labels=['Actual', 'Estimated', 'X-Error', 'Est. Error'
                title="X-"+net.name, open=1,
                comment=comments,
                plot_options=["color: 'red',", "color: 'blue',", "type: 'areaspline', color: 'grey', v

plotter.gethtmlLine(range(1,len(xpositions)), [consume, [row[1] for row in message_stats]],
                fname="Power-"+net.name, folder=DATETIME_DIR,
                xlabel="Node", ylabel="mJ", labels=['Energy','No. of Tx'],
                open=1, comment=comments, plot_type='bar',
                plot_options=["color: 'red',", "color: 'blue',", "type: 'areaspline', color: 'grey', v
```

Listing 2. An example script for simulation of DV-hop localization algorithm

IV. Another Simulation Scenario

We consider Internet of Things (IoT) application scenario where an energy harvesting WSN (EHWSN) node is installed/embedded within the Thing (object that need to be monitored) [14][15]. Several of such objects with EHWSN nodes form a cluster (in virtual star topology) around a high power coordinator node (or cluster head). EHWSN nodes can only communicate to its own coordinator (when they have enough energy). Coordinators are special wireless nodes which have sufficient power available and can send data to base station directly or via other coordinators (multi-hop) in a typical converge-cast application as illustrated in Fig. 7. These objects are mobile and can move around its neighborhood or move to another neighborhood (within the range of a different coordinator). The coordinators are installed at strategic fixed locations throughout the facility.

Figure 8 illustrates the scheme on time scale and can be summarized as follows:

- The coordinator periodically (period = $T_c = t_4 - t_0$) broadcasts a beacon pulse with 10% duty cycle. The pulse contains the MAC address (48 or 64 bytes) of the radio, which is universally unique, and the number of registered nodes. After transmitting the beacon, it goes in the listening mode to receive messages from any EHWSN mode which have any packets to send.
- The neighboring EHWSN nodes (which have enough power to operate) periodically wake up (period = $T_n > T_b$) with a certain duty cycle to receive the beacon pulse from the coordinator (t_1 in Figure 8). If the received coordinator's MAC address is different than the last communicated coordinator or the node has not communicated recently, then the node will send a registration message containing its MAC address and the power status (t_2 in Figure 8); otherwise node will go back to sleep or send the data packet if any. The destination address will be set to the coordinator address so that any other neighboring nodes which are listening will ignore it.

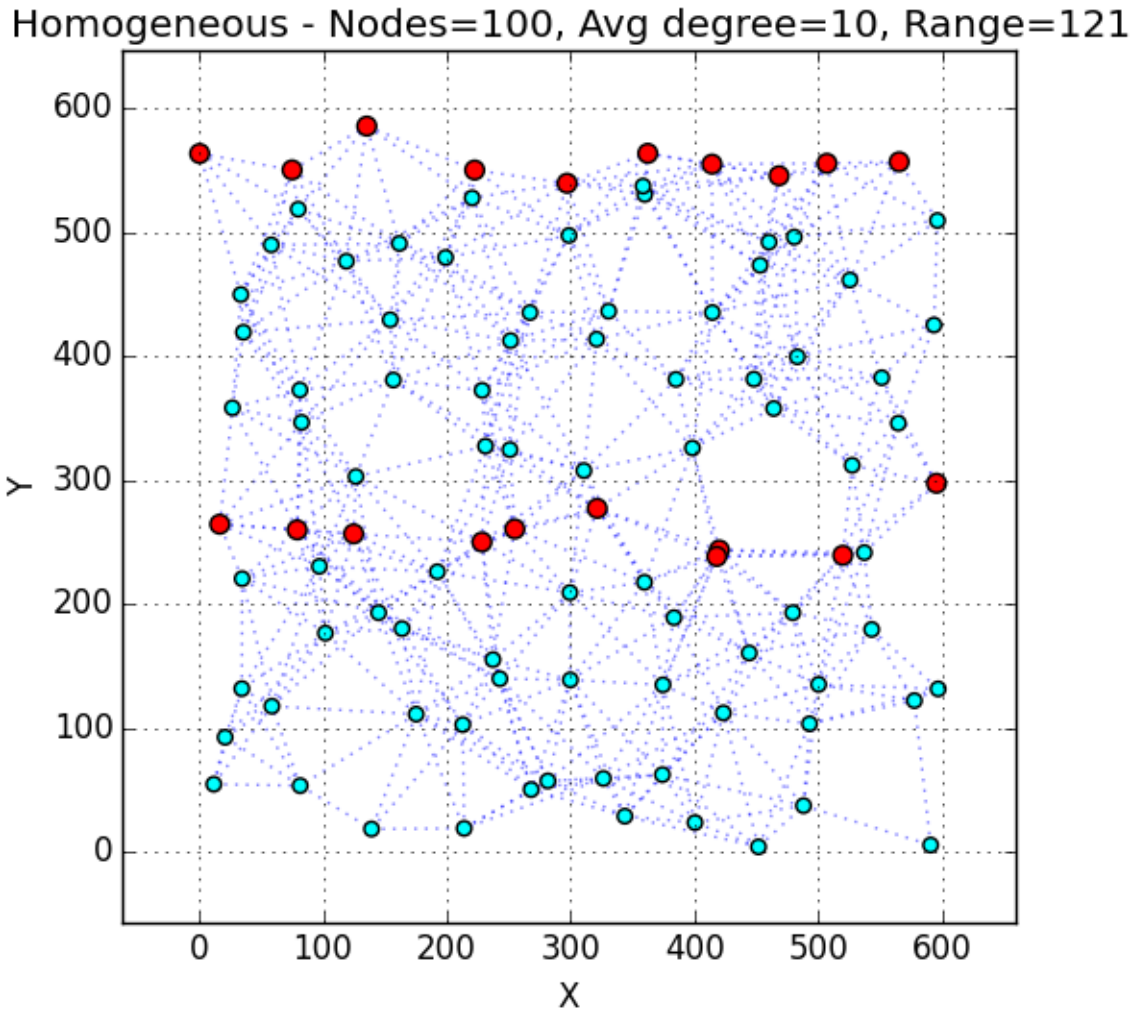
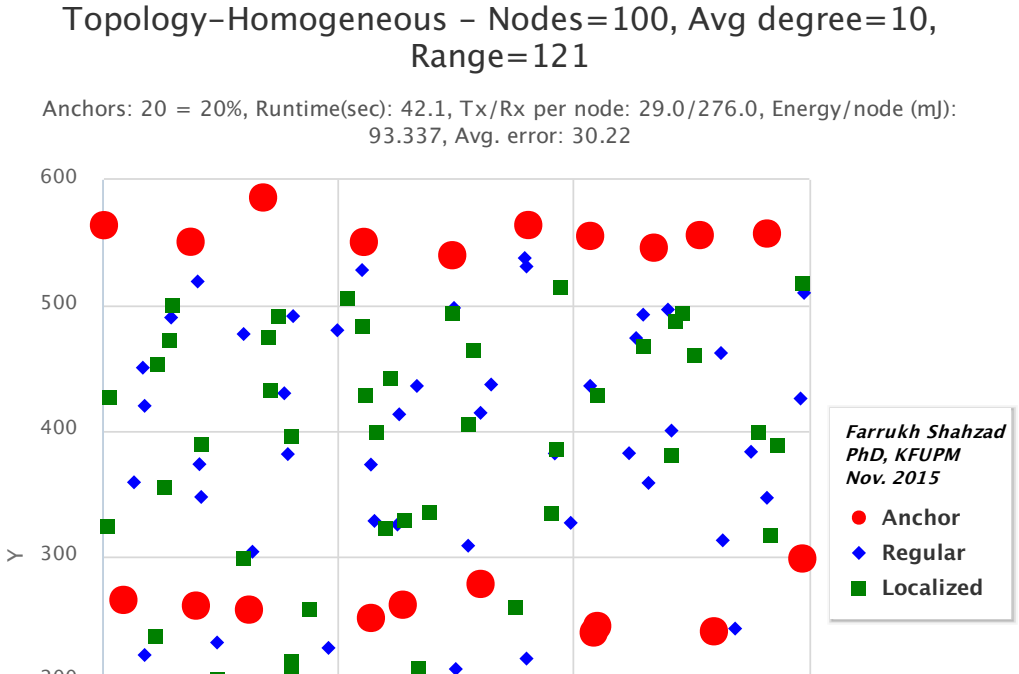


Figure 2. Homogeneous Random Topology



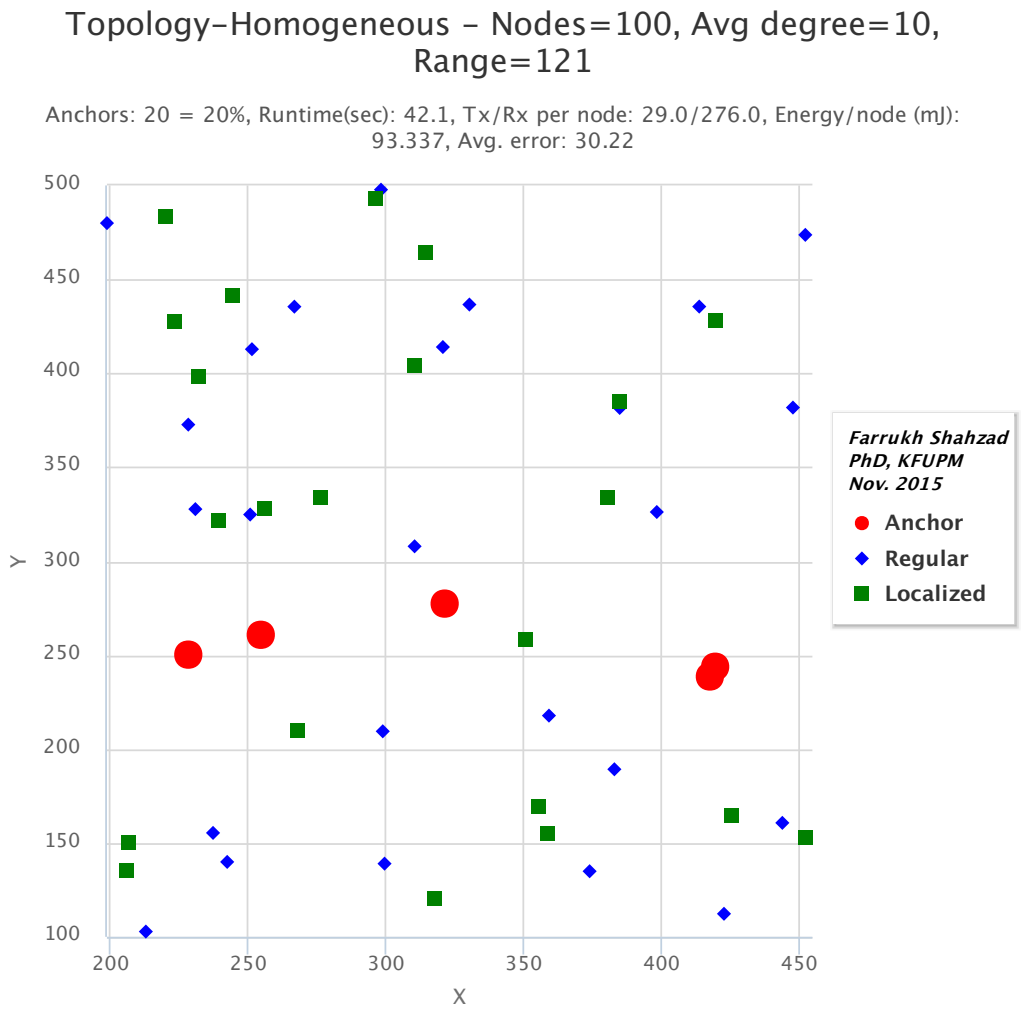
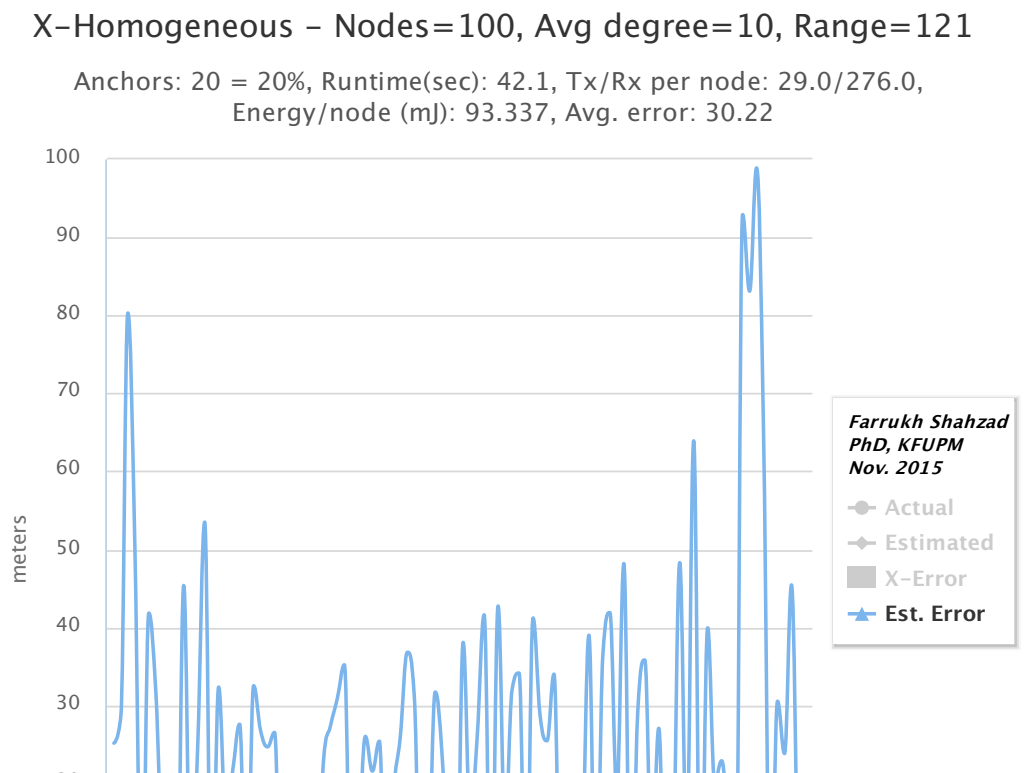


Figure 4. Topology image from interactive html/JavaScript chart (Zoomed in view)



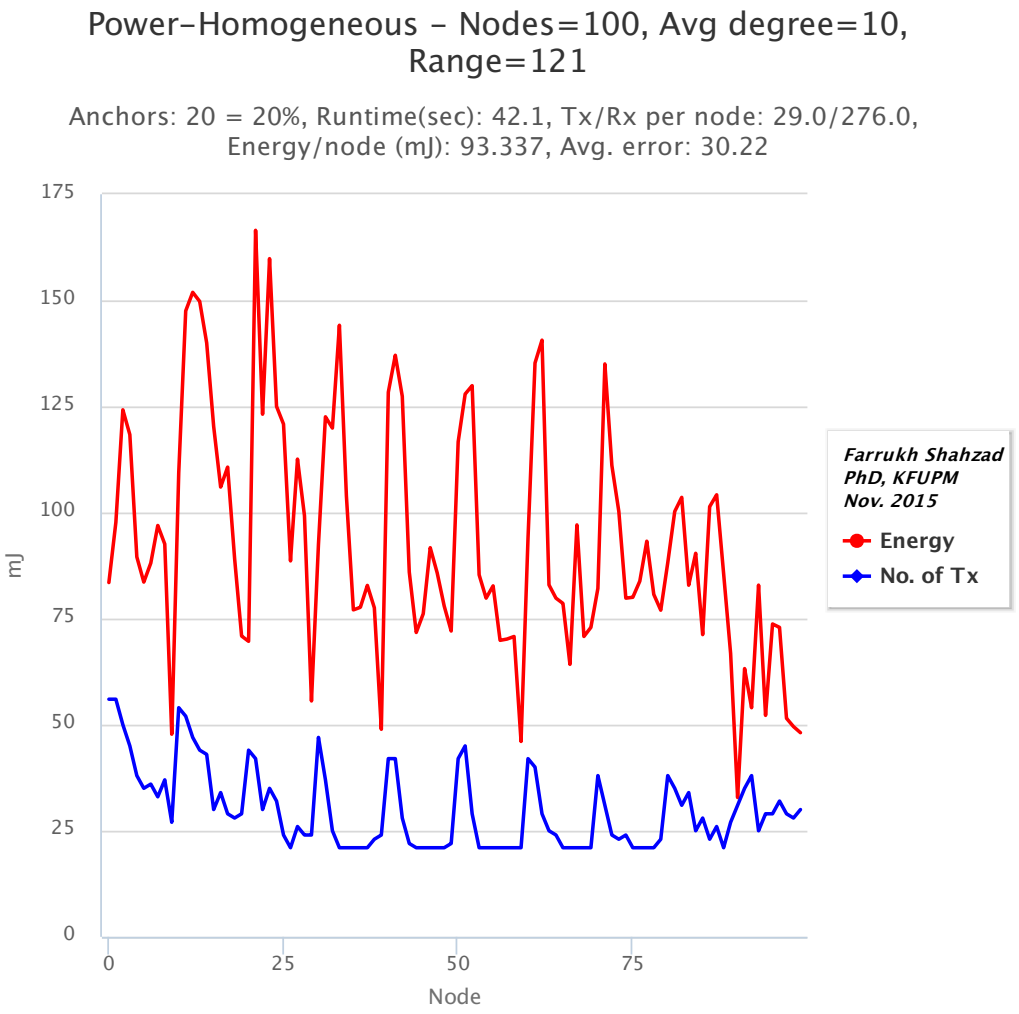


Figure 6. Power consumption and No. of Transmission for each node (interactive chart)

- On receiving the node registration message, the coordinator record the registration information and increment the number of registered nodes (t_3 in Figure 8). If coordinator receives a data message then it will buffer it for future transmission to base station or for aggregation. The coordinator will acknowledge the received packet in both cases.
- The case of multiple child nodes transmitting in response to the same beacon pulse is also shown in Figure 8 during the second beacon period. The contention is avoided by using a random back off time before transmission. The wining node will transmit first (node A transmits at t_7 in Figure 8) and other nodes will wait for the channel (e.g. node B transmits at t_9 in Figure 8).

A. Simulation setup

We only need to simulate the communication performance of one cluster formed by a coordinator and its children EHWSN nodes. The coordinator is placed in middle of n randomly deployed EHWSN nodes over a 600 m by 600 m area. We assumed that these nodes are constantly being charged during simulation and nodes are mobile (type=2, see Table III). We consider beacon, registration and data packet sizes of 100 bytes while the acknowledgment packet size of 15 bytes. We used default parameters for different modules as listed in Tables I - III. Some other parameters are shown in Table V. The simulation script utilizes the plotting and data collection modules to generate image and data files for easy visualization and analysis of simulation results (Figure 9).

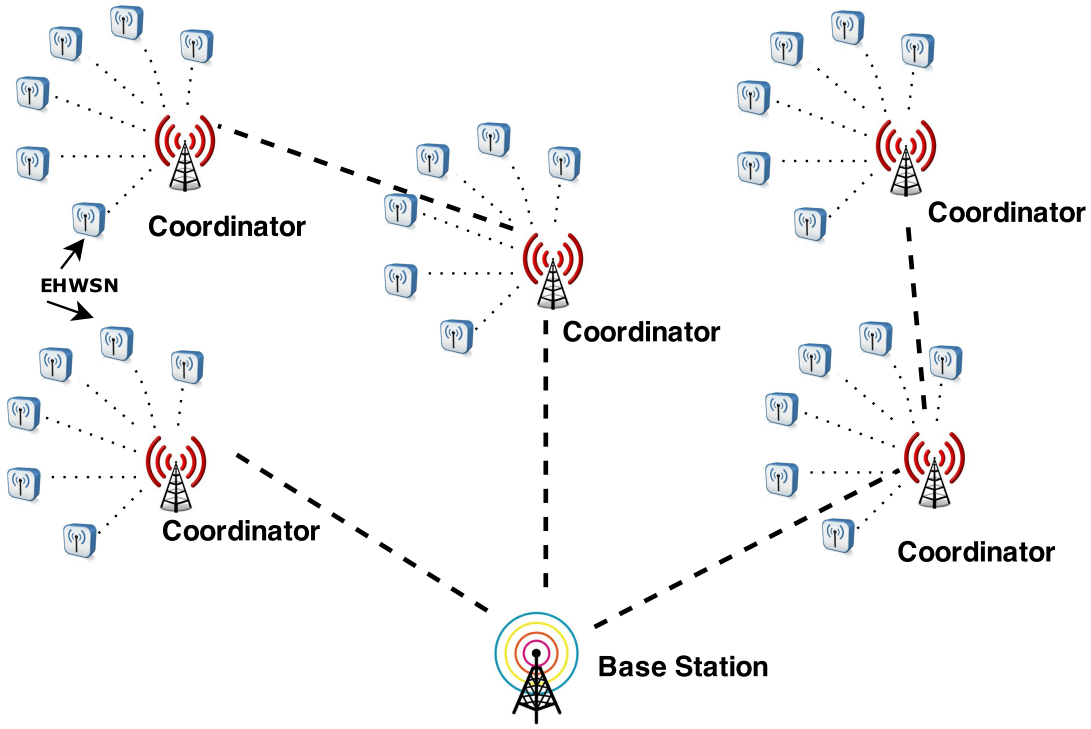


Figure 7. Proposed deployment scheme for EHWSN system

Table V
SIMULATION PARAMETERS

Parameter	Name	Value
Tb	Beacon period	1 sec
b	Beacon duty-cycle	10%
n	No. of nodes	5 - 100
Sd	Data packet size	100 bytes

B. Simulation Results

Figure 10 shows a simple topology generated for simulation using the Pymote. The center node (#1) acts as the coordinator for the EHWSN nodes (numbered 2 to 26). The node in lighter color means that its available energy is below E_MIN (=0.5 J). We arbitrarily selected node 5 and node 10 as borderline in terms of energy available (i.e. the initial energy at start of simulation). Node 5 doesn't have enough energy to transmit in the beginning but charged up above E_MIN (Table II) during the simulation and start communicating. On the other hand, Node 10 just has enough energy to send few messages before its energy level dropped below E_MIN. We set the charging rate to 0 for Node 10. The energy level change during the simulation run is shown in Figure 11.

Figure 12 shows the net node displacement during the simulation as they move around with constant speed but in random direction. Figure 13 illustrates the energy consumption of all EHWSN nodes. We can notice that some nodes never communicated due to low energy (like node 2, 4, 11, 15, 20, 21 and 26) whereas node 5 and 10 were only active during some part of the simulation as we discussed earlier. Finally Figure 14 shows the location of nodes at the end of simulation. Secondly, we vary the number of EHWSN nodes in the network from 10 to 100 in the increment of 5. The extended framework generates simulation output files for each iteration. The output files also include the overall summary. Figure 15 shows the generated topology for 100 EHWSN nodes (2 to 101). Figure 16 shows energy consumption plots for coordinator and other nodes combined (sum of energy consumption for all EHWSN nodes). The chart also shows number of messages (packets) received and lost at coordinator for each iteration.

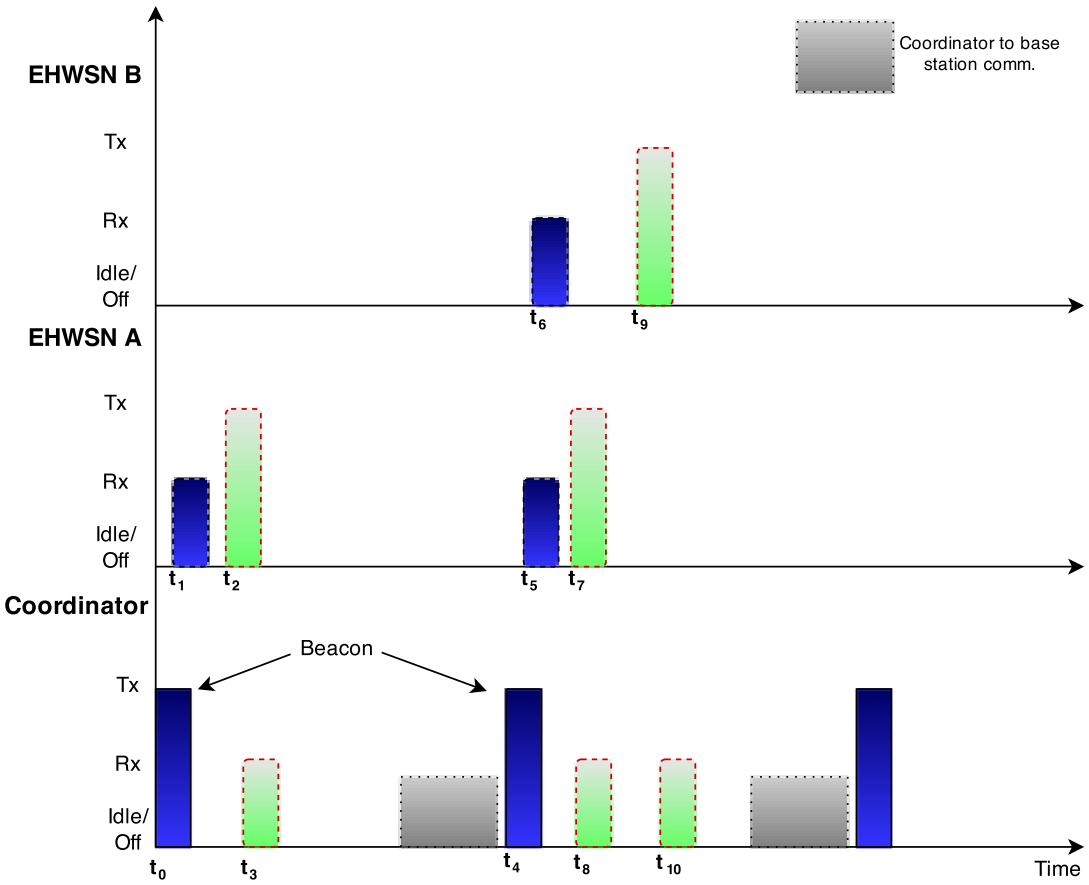


Figure 8. Timing diagram of coordinator's beaconing and node's transmission

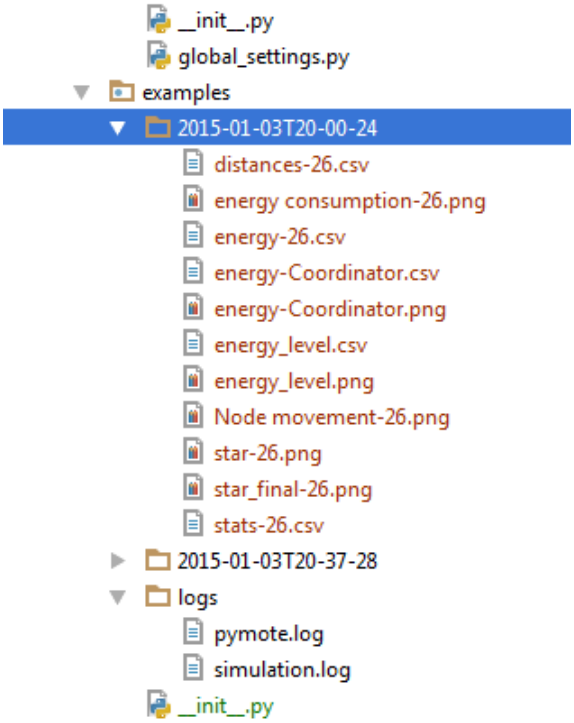


Figure 9. Simulation output files

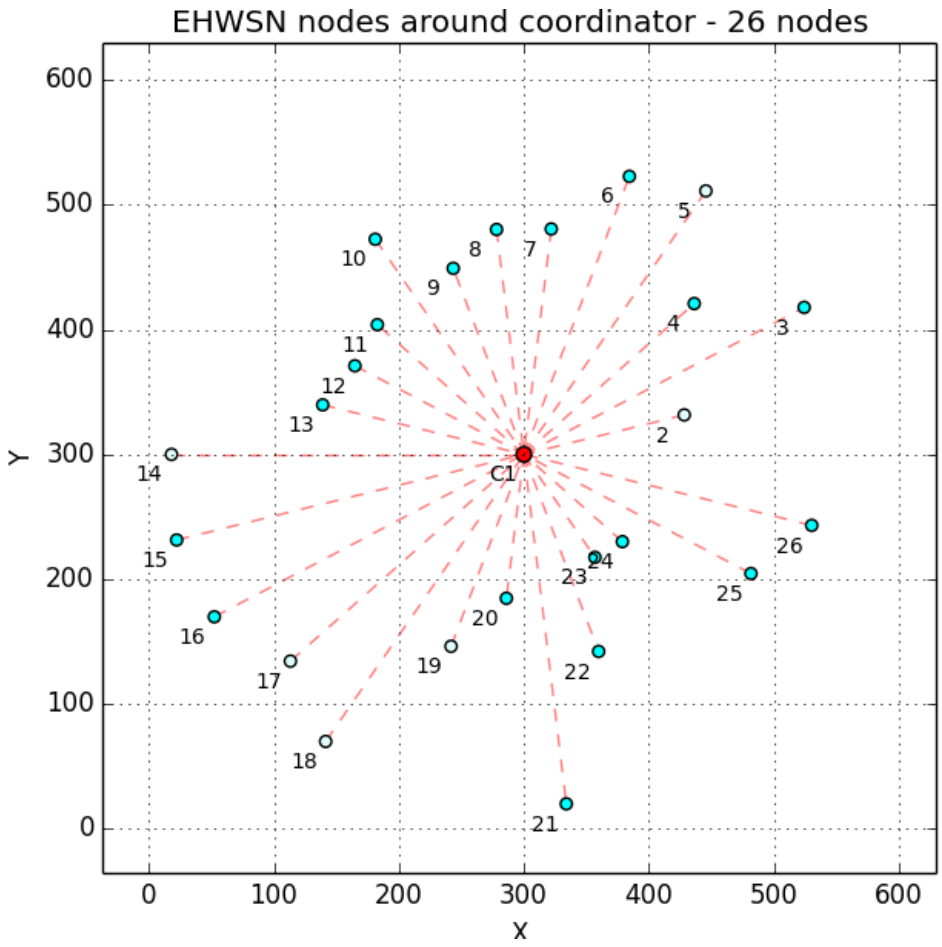


Figure 10. 25 EHWSN nodes around a coordinator

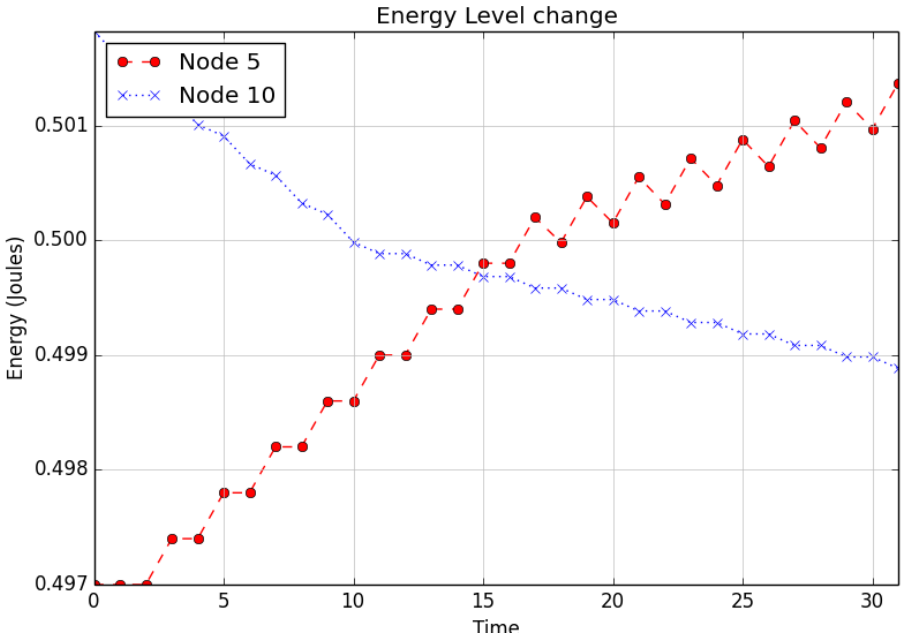


Figure 11. Energy level change for Node 5 and 10

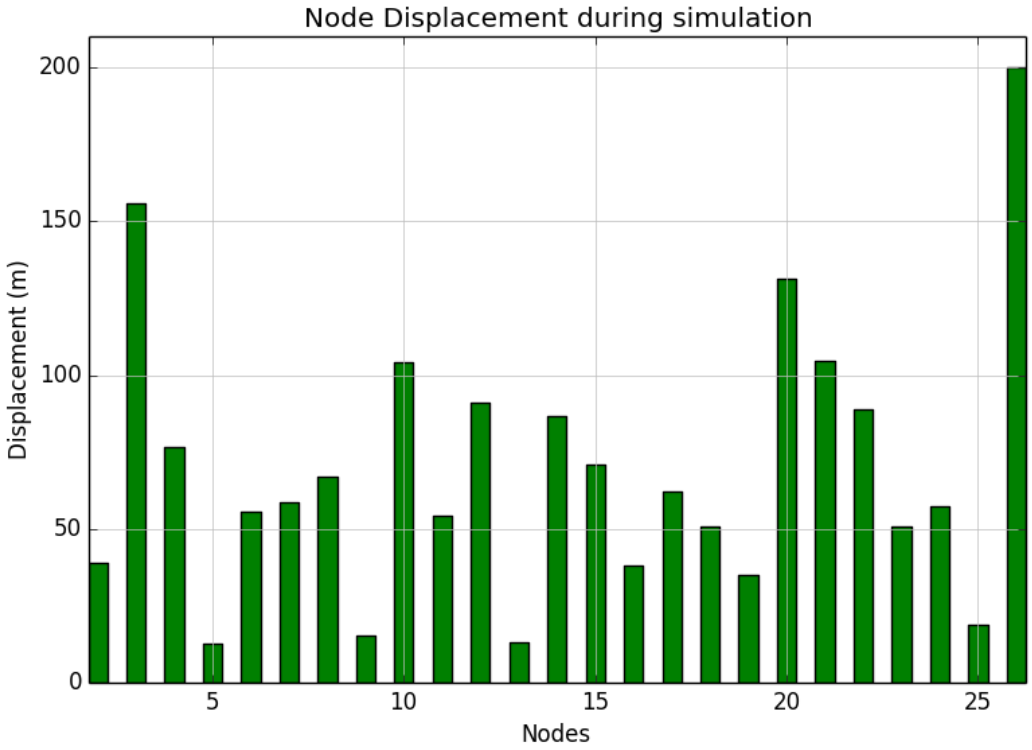


Figure 12. Node displacement during the simulation

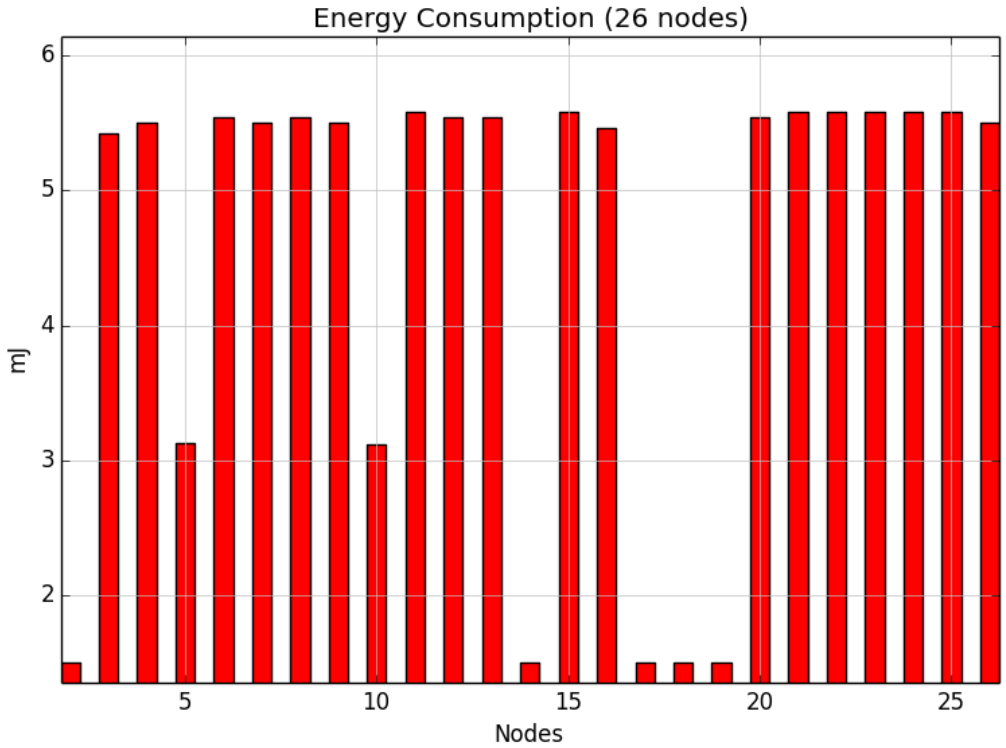


Figure 13. Energy consumption during the simulation

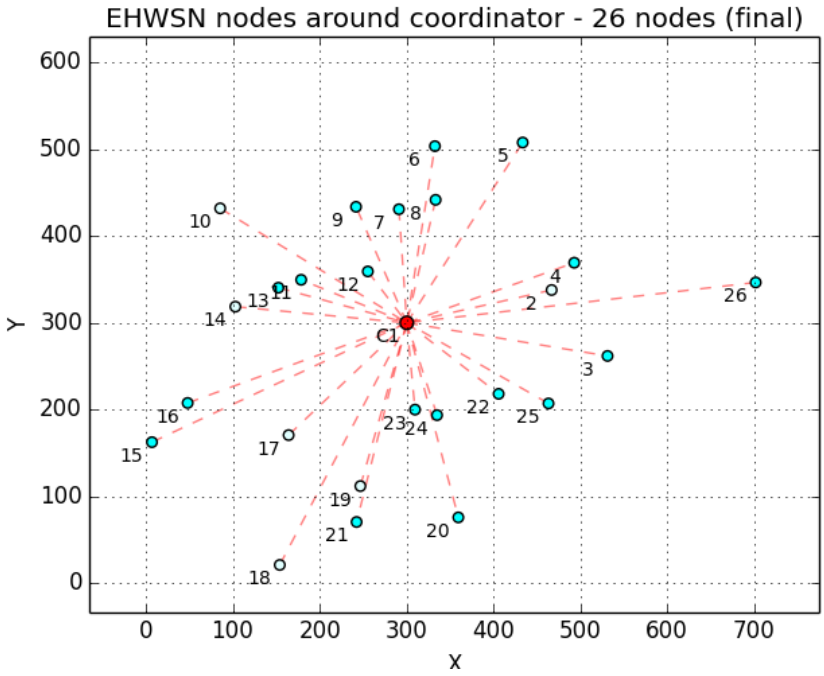


Figure 14. Final position of nodes at the end of simulation

Table VI
SIMULATION STATISTICS

Nodes	Coordinator Energy consumption(mJ)	Total EHWSN consumption(mJ)	Received Packets at Coordinator	Lost Packets at Coordinator
11	15	45	44	18
16	38	104	110	64
21	48	134	98	124
26	65	178	237	65
31	73	203	213	125
36	92	253	254	176
41	105	290	265	229
46	122	335	306	268
51	126	349	409	181
56	139	386	432	222
61	142	401	398	272
66	180	491	652	194
71	159	454	502	248
76	186	521	464	414
81	192	541	495	412
86	203	573	594	365
91	240	662	699	435
96	244	678	689	464
101	264	729	866	380

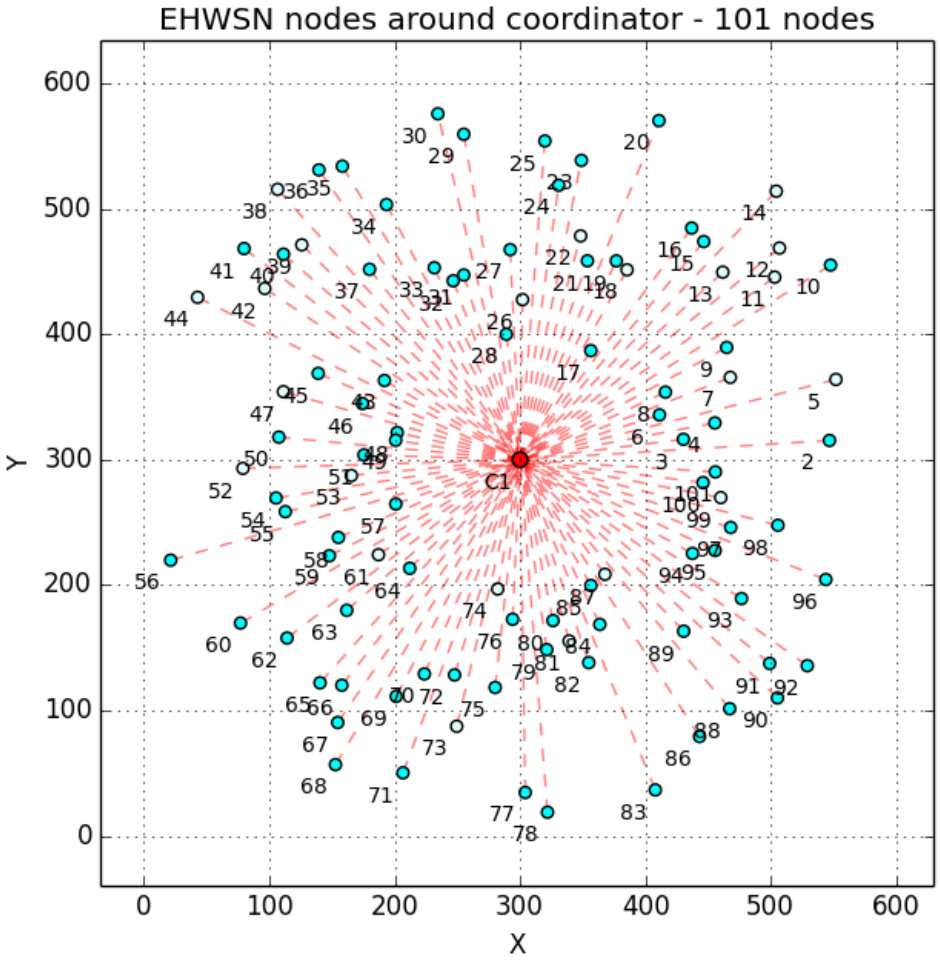


Figure 15. 100 EHWSN nodes around a coordinator

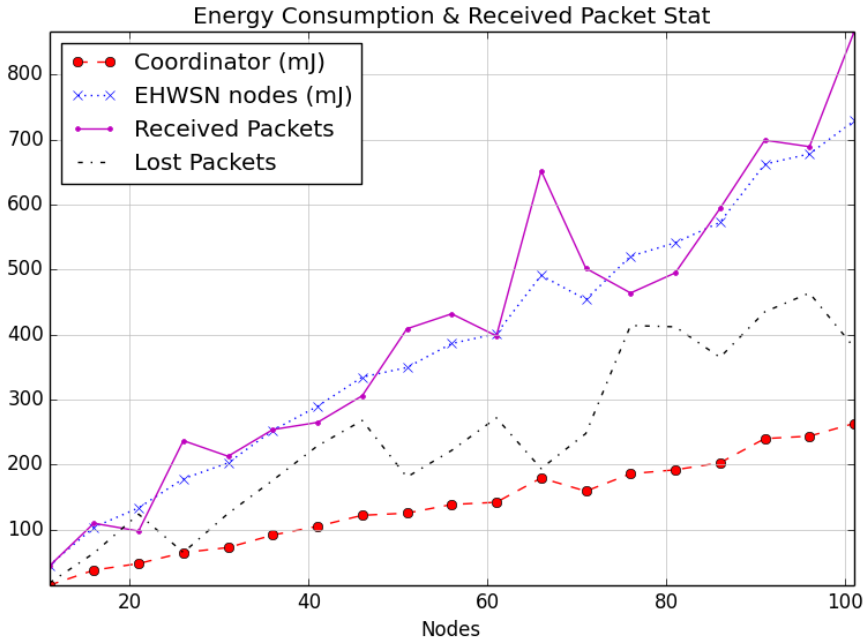


Figure 16. Overall summary

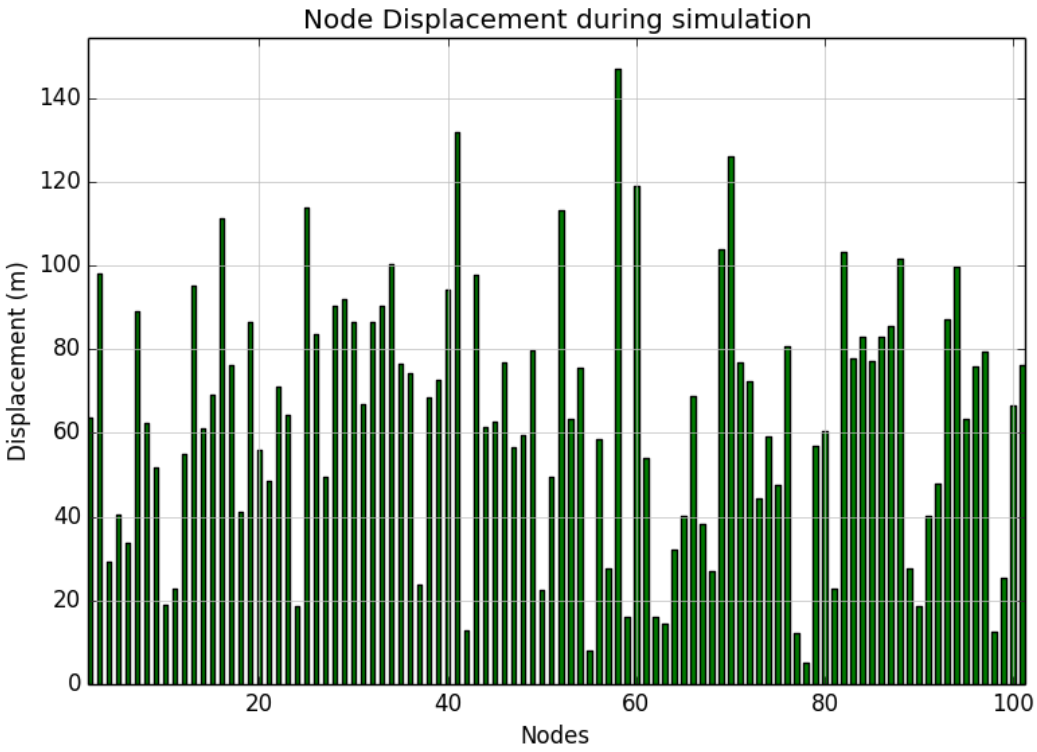


Figure 17. Node displacement during the simulation for 100 nodes

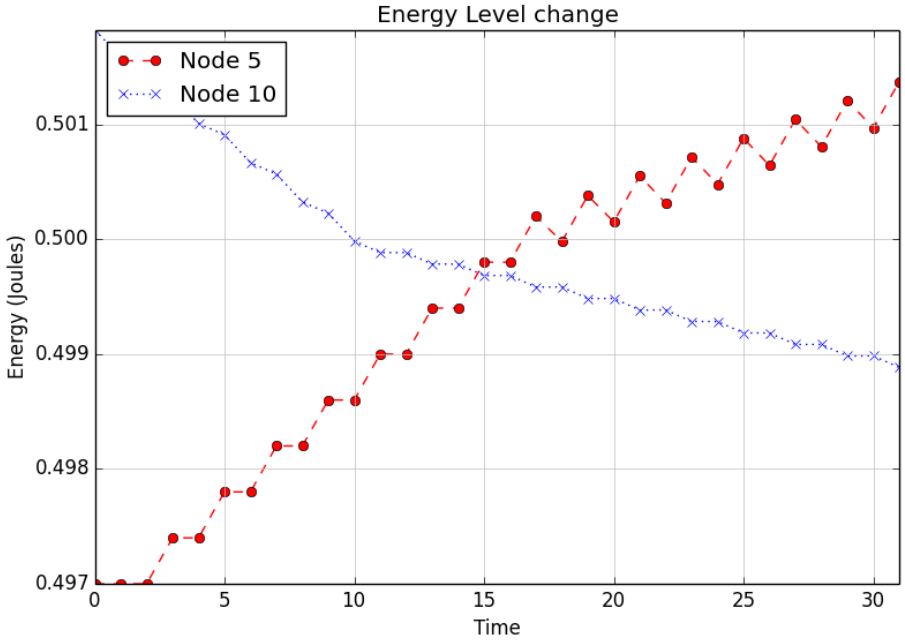


Figure 18. Energy level change for Node 5 and 10 with network size = 100

Table VI presented the overall simulation summary for all iterations. Node displacement and energy level change during the simulation for 100 nodes are shown in Figure 17 and Figure 18 respectively.

V. Conclusion

The development of a reliable and robust large-scale WSN system requires that the design concepts are checked and optimized before they are implemented and tested for a specific hardware platform. Simulation provides a cost effective and feasible method of examining the correctness and scalability of the system before deployment.

We utilized and extended the Python based Pymote framework to allow packet level simulation. We implemented modules for propagation, energy consumption and mobility models. We also added graphing and data collection modules to enhance the Pymote base functionality and modified existing modules for node, network, algorithm and logging to support the extended framework. We provided a comprehensive example with Python script (including line by line comments) and showed the corresponding results in the forms of plots and charts. Finally, we performed an example simulation for a scheme to efficiently utilize EHWSN in an IoT application. The simulation results presented include topology maps, plots for available energy, bar charts for node displacement and energy consumption and comparison of received and lost packets at the coordinator node.

References

[1] Q. Ali, A. Abdulmaowjod, and H. Mohammed, "Simulation & performance study of wireless sensor network (WSN) using MATLAB," pp. 307–314, 2010.

[2] A. Sobeih, "J-Sim: A Simulation and emulation environment for wireless sensor networks," *IEEE Wireless Communications*, vol. 13, no. 4, pp. 104–119, Aug. 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1678171

[3] Y. Tselishchev, A. Boulis, and L. Libman, "Experiences and Lessons from Implementing a Wireless Sensor Network MAC Protocol in the Castalia Simulator," in *2010 IEEE Wireless Communication and Networking Conference*. IEEE, Apr. 2010, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5506096

[4] A. Abuarqoub, F. Alfayez, M. Hammoudeh, T. Alsboui, and A. Nisbet, "Simulation Issues in Wireless Sensor Networks: A Survey," in *SENSORCOMM 2012 , The Sixth International Conference on Sensor Technologies and Applications*, Aug. 2012, pp. 222–228.

[5] L. Shu, M. Hauswirth, H.-C. Chao, M. Chen, and Y. Zhang, "NetTopo: A framework of simulation and visualization for wireless sensor networks," *Ad Hoc Networks*, vol. 9, no. 5, pp. 799–820, Jul. 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870510001435

[6] A. Kroeller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer, "Shawn: A new approach to simulating wireless sensor networks," p. 10, Feb. 2005. [Online]. Available: http://arxiv.org/abs/cs/0502003

[7] F. J. . A. Marculescu, "AlgoSenSim - Overview [TCS-Sensor Lab]," 2006. [Online]. Available: http://tcs.unige.ch/doku.php/code/algosensim/overview

[8] "Sinalgo." [Online]. Available: http://dcg.ethz.ch/projects/sinalgo/

[9] D. Arbula and K. Lenac, "Pymote: High Level Python Library for Event-Based Simulation and Evaluation of Distributed Algorithms," *International Journal of Distributed Sensor Networks*, vol. 2013, no. 797354, p. 12, 2013.

[10] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, Aug. 2002. [Online]. Available: http://doi.wiley.com/10.1002/wcm.72

[11] G. Han, J. Chao, C. Zhang, L. Shu, and Q. Li, "The impacts of mobility models on DV-hop based localization in Mobile Wireless Sensor Networks," *Journal of Network and Computer Applications*, vol. 42, pp. 70–79, Jun. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804514000824

[12] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. [Online]. Available: http://scitation.aip.org/content/aip/journal/cise/9/3/10.1109/MCSE.2007.55

[13] T. Hø ns, "Highcharts, Highstock and Highmaps documentation — Highcharts," 2013. [Online]. Available: http://www.highcharts.com/docs

[14] P. Nintanavongsa, M. Y. Naderi, and K. R. Chowdhury, "Medium access control protocol design for sensors powered by wireless energy transfer," in *2013 Proceedings IEEE INFOCOM*. IEEE, Apr. 2013, pp. 150–154. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6566753

[15] S. Basagni, M. Y. Naderi, C. Petrioli, and D. Spenza, "Wireless sensor networks with energy harvesting," *Mobile Ad Hoc Networking: The Cutting Edge Directions*, pp. 701–736, 2013.