INTERNATIONAL HELLENIC UNIVERSITY
School of Science and Technology

Virtual Labs #4: "Topology Control in Wireless Sensor Networks": Documentation

Referring to Courses:
Wireless Sensor Networks,
MSc in Information and Communication Technology Systems

prepared by

Liaskos Christos
Computer Architecture and Communications Lab
Department of Informatics
Aristotle University of Thessaloniki

Dr. Koutitas George
School of Science and Technology
International Hellenic University

September 2010

Table of Contents

List of Figures

1.0 Preface

This report constitutes the documentation of the 4^{th} virtual laboratory environment that was developed for the "Mobile Communication Networks" and "Sensor Networks" courses of the MSc in Information and Communication Technology Systems, International Hellenic University.

Purpose of the present work is to provide a tool for the study of topology control and maintenance in Wireless Sensor Networks, with respect to energy efficiency per node an per battery model. The application heavily relies on the "atarraya" simulator by Pedro M. Wightman and Miguel A. Labrador. The presented application is a modified version of the original simulator, specially adapted for educational purposes in the context of the "Wireless Sensors" course. For commercial use of the software, permission by both the I.H.U and the original authors is required.

This documentation heavily relies on [1], the only known official documentation source of the original simulator. It consists of four main parts:

The fist section outlines the basic theoretical concepts and problem of topology control.The following three sections provide usage information for the simple user (student of an MSc program in Telecommunications), the supervisor (academic assistant with some knowledge of programming in MATLAB) and the programmer (expert in generic object oriented programming and particularly in MATLAB). The provided information in this leaflet intends to familiarize the simple user with the graphical interface and its potential, endow the supervisor with enough knowledge to create custom exercise scenarios and program expansions, and provide a blueprint for the programmer that has been bestowed with the task of heavily modifying/expanding the original program.

September 10, 2010

Liaskos Christos, Electrical Engineer, Dpt. of Informatics, A.U.Th.

2.0 Introduction: Theoretical concepts of topology control in WSN

In this application the user must be familiar with the following additional theoretical concepts, besides the ones presented in VLAB2.

2.1 Topology control and maintenance

Topology control is a technique used mainly in wireless ad hoc and sensor networks in order to reduce the initial topology of the network to save energy and extend the lifetime of the network. The main goal is to reduce the number of active nodes and active links, preserving the saved resources for future maintenance.

Topology control has been divided into two subproblems: topology construction, in charge of the initial reduction, and topology maintenance, in charge of the maintenance of the reduced topology so characteristics like connectivity and coverage are preserved.

This is the first stage of a topology control protocol. Once the initial topology is deployed, specially when the location of the nodes is random, the administrator has no control over the design of the network; for example, some areas may be very dense, showing a high number of redundant nodes, which will increase the number of message collisions and will provide several copies of the same information from similarly located nodes. However, the administrator has control over some parameters of the network: transmission power of the nodes, state of the nodes (active or sleeping), role of the nodes (Clusterhead, gateway, regular), etc. By modifying this parameters, the topology of the network can change.

Upon the same time a topology is reduced and the network starts serving its purpose, the selected nodes start spending energy: The "optimal" reduced topology stops being it at the

first second of full activity. After some time being active, some nodes will start to run out of energy. Especially in wireless sensor networks with multihoping, it is a fact that nodes that are closer to the sink spend higher amounts of energy that those farther away due to packet forwarding. The network must restore the reduce network periodically in order to preserve connectivity, coverage, density and any other metric that the application requires.

2.2 Topology Construction Algorithms

There are many ways to perform topology construction:

Change the transmission range of the nodes Turn off nodes from the network Create a communication backbone Clustering, etc. Some examples of topology construction algorithms are:

Tx range-based

- Geometry-based: Gabriel graph (GG), Relative neighborhood graph (RNG), Voronoi diagram
- Spanning Tree Based: LMST, iMST
- Direction Based: Yao graph and Nearest neighbor graph, Cone Based Topology Control (CBTC), Distributed RNG
- Neighbor based: KNeigh, XTC
- Routing based: COMPOW

Hierarchical

- CDS-based: A3, EECDS, CDS-Rule K

- Cluster-based: Low Energy Adaptive Clustering Hierarchy (LEACH), HEED

Graphical Examples

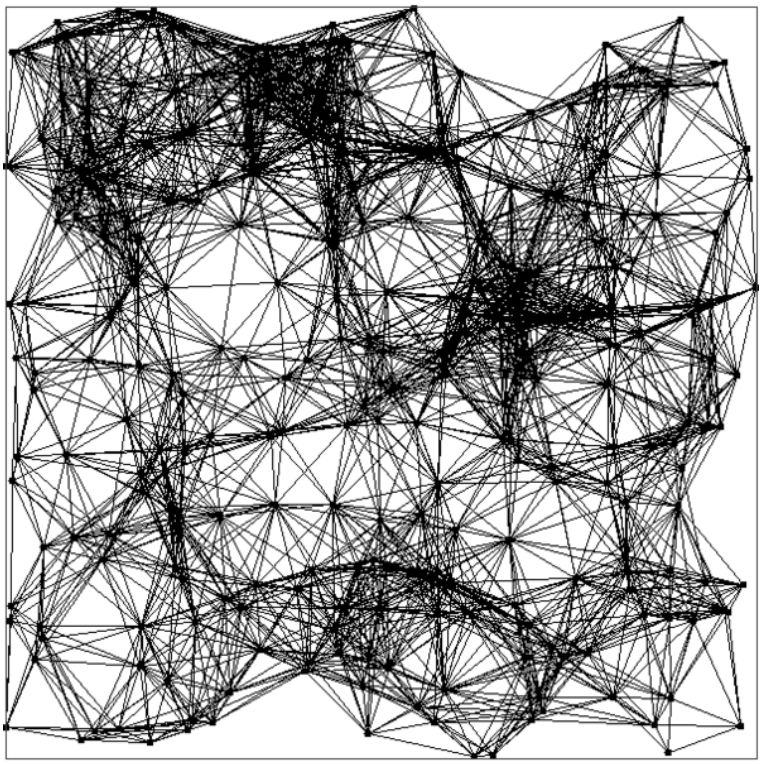

Figure 1: Full power network topology.

2.3 Topology maintenance algorithms

In the same manner as topology construction, there are many ways to perform topology maintenance:

- Global Vs. Local
- Dynamic Vs. Static Vs. Hybrid
- Triggered by time, energy, density, random, etc.

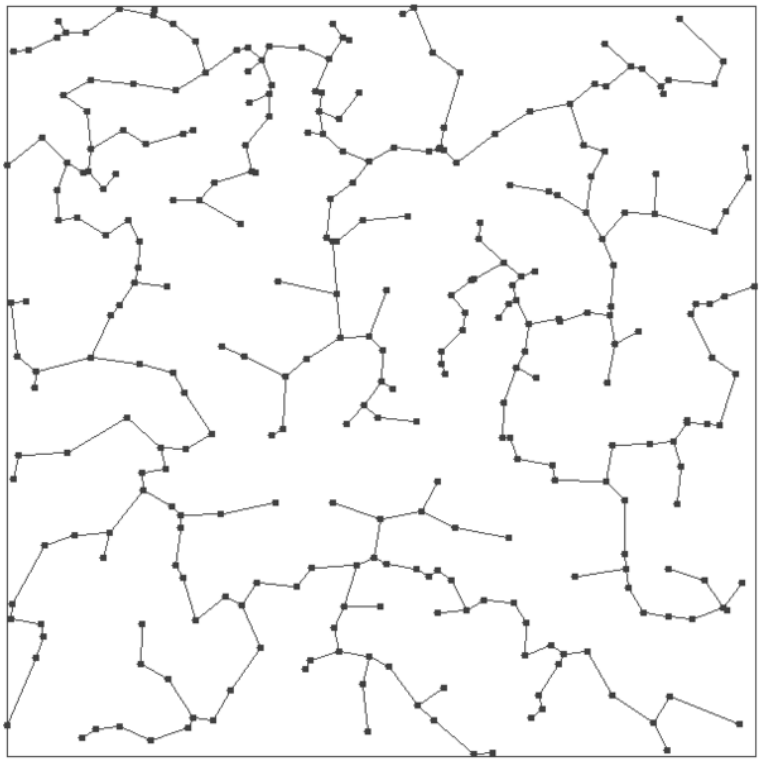

Figure 2: Reduced network topology via Minimal Spanning Tree (Change in Tx Range).

Some examples of topology maintenance algorithms are:

Global

- DGTRec (Dynamic Global Topology Recreation): Periodically, wake up all inactive nodes, reset the existing reduced topology in the network and apply a topology construction protocol.

- SGTRot (Static Global Topology Rotation): Initially, the topology construction protocol must create more than one reduced topology (hopefully as disjoint as possible). Then, periodically, wake up all inactive nodes, and change the current active reduced topology to the next, like in a Christmas tree.

- HGTRotRec (Hybrid Global Topology Rotation and Recreation): Work as the SGTRot, but when the current active reduced topology detects a certain level of disconnection,

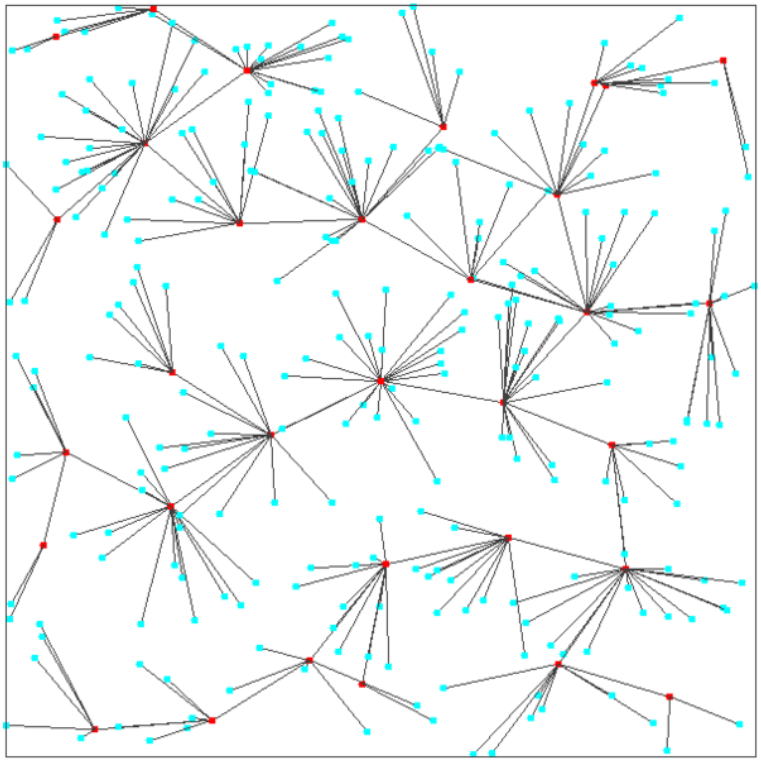

Figure 3: Reduced network topology via Connected Dominating Set (Select a subset of nodes that cover all the network and turn off non-selected nodes).

reset the reduced topology and invoke the topology construction protocol to recreate that particular reduced topology.

Local

- DL-DSR (Dynamic Local DSR-based TM): This protocol, based on the Dynamic Source Routing (DSR) routing algorithm, recreates the paths of disconnected nodes when a node fails.

2.4 Further reading

See references [2, 3, **?**] for Fresnel zones and [1]. The above content was based on corresponding entries of "*wikipedia, the online encyclopedia*".

3.0 User's Manual

3.1 Requirements

- Java Runtime Environment 1.6.5 or newer.
- Dual core CPU with 2GB RAM. Quad core CPU with 4GB RAM ensures optimal performance.
- 5MB of hard disk space are required to hold the application files.

3.2 Installation Notes

No installation is required. Simply copy the application folder to a desired location.

3.3 Application execution and usage

To start the application, simply execute the provided .JAR file. Execution way may be OS specific, but is defined by the JRE provider.

Upon execution the main form appears:

The application offers a variety of options for designing and experimenting with topology control algorithms. This section provides an brief user guide on how to use the simulation tool and all its available options. The first step is to have a clear understanding of the simulation scenarios to be run. Here, the user needs to know in advance which protocols he or she wants to use; whether the experiment is just a preliminary test or an exhaustive performance evaluation, what is the nature of the topologies that she is planning to use, what type of statistics are needed, and so forth. In this section, these questions are answered in

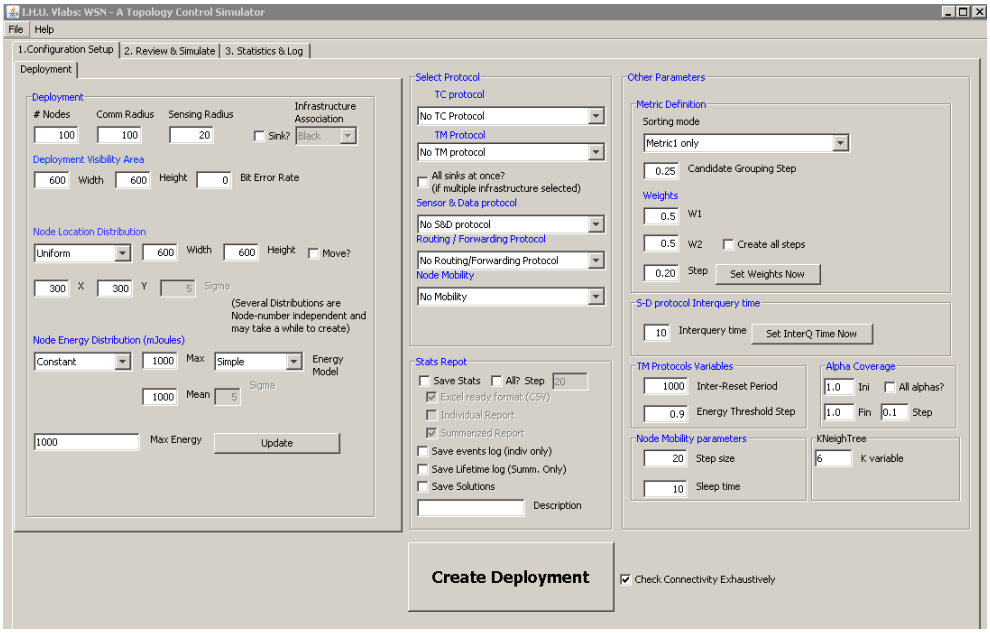

Figure 4: The application's main form.

order that a user may create and run successful simulations with the application.

Selection of the protocols

The application includes four types of protocols: Topology construction, topology maintenance, sensor-data, and communication protocols. the application can be set to work in either of the following two modes related to topology control: Topology construction only, or All protocols. The first mode is designed to test a specific topology construction algorithm and measure the initial reduced topology that the algorithm produces. In order to select this mode, the user only needs to select a topology construction protocol.

The second mode is intended to test not only the reduced topology but the lifetime of the network, based on the combination of all the protocols, i.e. topology construction and topology maintenance. In order to select the second mode the user needs to select a protocol in each of the protocol categories, i.e. select a topology construction and a topology maintenance protocol, otherwise the application will not allow the user to run the experiment.

The topology construction protocols included in the application are based on algorithms presented in published papers. Currently, the application includes the A3 , EECDS, and CDSRule-K algorithms. Although all types of topology construction protocols might be implemented, such as those based on changing the transmission range of the nodes, hierarchical protocols, cluster-based protocols, etc., the current version of the application includes only those based on the Connected Dominating Set concept. In addition, and for the sake of comparison with a wireless sensor network without topology control, the application offers a protocol that does not reduce the topology at all, called JustTree. The only service that this protocol provides is the creation of a forwarding tree to implement the constant gateway forwarding protocol.

The application also includes all the topology maintenance techniques included in , i.e., static, dynamic and hybrid topology maintenance techniques. They are generic algorithms that work with any of the topology construction protocols. The simplest topology maintenance protocol included in the application is the No Topology Maintenance protocol, which does nothing to maintain the initial topology, but monitors it until it dies. The protocol is in charge of informing the application when this event happens. In the application, the termination policy is defined as the moment at which the sink stops receiving information from the nodes.

The sensor-data management protocol models the behavior of the sensors, regarding variables like data transmission frequency, data aggregation policies, etc. the application provides a simple protocol for sending and receiving messages without data aggregation. Nodes transmit data packets at predefined times, and forward every received data packet based on the forwarding policy explained in the following paragraph.

Although communication or routing protocols are not the focus of this simulator, some kind of routing procedure is necessary so packets can reach the sink. Given that the topology control protocols implemented in the application are designed to produce a tree-like reduced

topology, the tool provides a very simple forwarding algorithm that allows packets to reach the sink node: The constant gateway forwarding protocol. In this protocol, packets are always forwarded to a default gateway unless the destination of the packet is a direct neighbor, in which case it will be sent directly to that node. In a tree-like topology, the gateway of a node is simply its parent node. In this fashion, the packet will finally reach the sink node, since it is the root of the tree.

The application does not include any routing protocol, but defines a data structure that models a routing table. This data structure allows users to develop more advanced routing protocols than the one currently implemented. In addition, the routing table can store a limited amount of packet sequence numbers (events have a field for this purpose) that allow the implementation of other forwarding algorithms like flooding-based protocols, or save the last versions of the routing protocol information packets, like routing tables on a Distance Vector protocol, or the last neighborhood information in a Link-State protocol.

In the applications graphical user interface, the panel named the application presents the available protocols. In addition, this panel contains the controls for the simulation agent, the report configuration options, simulation events and statistics panels, and the batch simulations controls.

Other protocols

The application has been used to validate analytically derived equations and show special efiects or behaviors of topology control algorithms. This section describes five tools included in the application that are very well suited for educational purposes, as they allow users to:

Calculate the Critical Transmission Range (CTR) based on the formulas of Penrose-Santi and Santi-Blough .
Reproduce the experiment to obtain the Giant Component figures: Greatest Connected

Component and Ratio of Connected Topologies.

Reproduce the experiment that proves connectivity of the CTR formula of Santi-Blough for 2 dimensional deployments.

Calculate the Minimal Spanning Tree on a given graph and provide the sum of the selected edges.

Save the neighborhoods of a graph in a file.

The Minimal Spanning Tree of a graph is a classical tool for graph analysis Prims algorithm was implemented. Regarding the last tool, the information provided by it can be used to define and solve linear programming optimization problems on graphs, like finding a minimal set cover of the graph.

Energy and communications model

In the design of the application simplicity and focus on reaching a better understanding of the behavior of the topology con- trol algorithms was embraced. As such, several assumptions were made to make the simulator simpler while still good enough for achieving its main objective. These assumptions are related to the energy and the communication models utilized in the tool.

The energy model included in the application is based on the following formulas:

$$E_{TXbit} = E_{elect} + (E_{amp} \cdot (\pi r^2))$$

$$E_{RXbit} = E_{elect}$$

In addition, the application also makes the following assumptions:

- During the idle time, a node does not spend energy. Even though this assumption has

been proven untrue because being idle might be as costly as receiving data, this is still an assumption that can be done in some experiments, for example, the delay time for a topology construction protocol is very short compared to the total amount of time the network is going to be active, so the energy spent on idle listening is negligible. This function is being implemented for the next version of the simulator.

- The nodes are assumed to have one radio for general messages and a second radio for control messages: The main radio is used in all operations when the node is in active mode, and the second one (low power cheaper radio) is used to send and receive control packets to wake up the main radio. Only the main radio can be turned off, which means no messages will be received and no energy will be used. The secondary radio is assumed to use half the energy of the main radio.

- The sink node has a infinite source of energy. In general, the sink node is assumed to be powered from an external source of energy.

The communication model used in the application is based on the following assumptions:

- The communication range of the nodes is a perfect symmetric unit disk. If $d_{x,y} \leq r_x \rightarrow$ x and y can see each other.

- A constant bit error rate (BER) is defined for the whole network. This is a simple implementation of an error model. Whenever a packet is going to be sent, a random number is generated and compared to the message error rate (that depends on the size of the message). If the random number is greater, the message is re ceived, otherwise it is lost. The default value for the BER is 0, which means there is no packet loss. No sum of partially received packets will build a complete packet.

- the application assumes that there exists a Data Link Layer that deals with packet losses and retransmissions, but it does not model this. In order to model some of the

consequences of the operations of the MAC layer, the packets are delayed a random amount of time in order to model delays occurring due to retransmissions, contention, etc. The variable that defines the maximum delay value can be found in the constants interface, by the name of MAX TX DELAY RANDOM. The default value for this variable is 0.2 time units.

Type of experiments

The application offers two types of experiments: single topology based, that uses the visual representation of the topology, and the batch execution mode that simulates a large set of topologies. The single topology based type is good for protocol design and debugging. The batch type is made for full scale evaluation and analysis. During the protocol design phase, it is very important to have the capability to run a small number of controlled topologies one at a time to be able to compare the results of several runs on a known scenario. This is a debugging phase where many changes are introduced in the protocol until it behaves as intended. During this process, a visual representation of the topology and the performance of the topology control algorithm is very helpful. Once the protocol has reached a stable version that has worked well in several single topologies, the protocol needs a more exhaustive test with a larger number of topologies in order to analyze its average behavior. The batch execution mode allows the researcher to run simulations with hundreds of random topologies. In this case, the visual representation of the topologies is not necessary; actually, it would slow down the simulation process.

Structure of a topology

A topology in the application is composed of four basic elements: Deployment area, regular nodes, sink nodes, and virtual network infrastructures or VNI. The deployment area is an abstract concept, which is useful for visualization purposes. It is a rectangle in which the user deploys the nodes of the network. In order to define the deployment area, the user

needs to define its width and height.

The set of regular nodes is usually the biggest set of elements in the topology. They are in charge of monitoring the environmental variable or variable of interest, sending this information to the sink and routing packets. As with any wireless sensor devices, regular nodes are very limited in terms of resources. The sink nodes are special nodes that, in most scenarios, are included to receive the information from all active nodes in the network. They serve as bridges between the wireless sensor network and any type of external network used to transport the sensed data to its final destination somewhere else in the Internet. In some cases they are also in charge of initiating, executing, and/or controlling the topology construction and maintenance protocols, routing protocols, etc. Despite the fact that in real life the hardware configuration of the sink nodes is different compared to a regular wireless sensor device, the application uses the same data structure to model both nodes, with the main difference that sink nodes are considered to have an unlimited amount of energy. the application also support several topologies at the same time through the VNI feature. It identifies the different VNIs with numbers from 0 to 6, and visually with colors: Black, Red, Blue, Green, Orange, Pink, and Yellow. Each node is assumed to have a separated data structure for each VNI, so the information of each one is independent from each other. Regular nodes have no affiliation with a particular VNI, while each sink node is associated with a VNI to which it serves as a sink node. All sink nodes are assumed to be regular nodes by the VNI, that are not associated with them, keeping their characteristic of unlimited energy. The following list contains all the available options to define the network topology:

- Sink or No Sink: Even though most wireless sensor networks contain one or more sink nodes, the application allows the user to define a wireless sensor network without sinks.
- One or multiple sinks in a single VNI: the application allows to define a single network with a single sink or multiple sinks.

- One VNI or multiple VNIs: Having multiple VNIs is the way in which the static global topology maintenance techniques were implemented, where there are several subsets of topologies and one sink.

Structure of the nodes

In terms of the nodes, the application can manage homogeneous networks, in which all nodes have the same characteristics, and heterogeneous networks, where there is at least one node that is different from the others.

When creating a topology, the simulator works based on subsets of homogeneous nodes. Each set defines a family of nodes that share the same characteristics. For example, the user can define a set of weak nodes with low transmission range and low energy, and a set of powerful nodes with high transmission range and higher energy. the application also allows for different random distributions for the location and energy of each group of nodes, that will allow the user to create denser zones in the topology, or zones with nodes that have less energy.

The data structure that models these families of nodes is called Creation Words. These creation words are created based on the values defined on the Deployment Options panel in the simulator. The user can create as many creation words as desired: these families can model from a single node to the complete set of regular nodes. In the panel there are two list boxes where the creation words are stored until the topology is created: The regular nodes list (top) and the sink creation words (bottom). There are three buttons for adding a new creation word, removing a selected creation word, and clearing both list boxes.

The following table summarizes the parameters that can be defined for a homogeneous family of nodes, as well as some other parameters that are related to the execution of the topology maintenance and the sensor-data protocols. All these options can be found in the

Deployment Options tab, under the tabs named Main Parameters and Other Parameters.

Parameter	Description
Number of Nodes	Integer value
Communication Radius	r_{comm} is an integer value
Sensing Radius	r_{sens} is an integer value
Position Distribution	- Uniform in a square area of size h and w, centered in (x, y) - Normal with $x' \in N(\mu_x, \sigma)$ and $y' \in N(\mu_y, \sigma)$ - Grid H-V: Distribute nodes in the deployment area with a distance of r_{comm} between nodes, so nodes are adjacent with their vertical and horizontal neighbors - Grid H-V-D: Distribute nodes in the deployment area with a distance of $r_{comm} \cdot \sqrt{2}$ between nodes, so nodes are adjacent with their vertical, horizontal and diagonal neighbors - Constant position: (x, y) for all nodes in the creation word - Center: (x, y) is the center of the deployment area - Manual: Use the mouse to select the position in the deployment area
Energy Distribution	- Constant position: $e \in \Re$ for all nodes in the creation word - Uniform with maximum energy e_{max} - Normal with $e' \in N(\mu_e, \sigma)$ - Poisson with λ_e
Sink?	This parameter determines if the nodes in the creation word are sink nodes
VNI Selection	This parameter associates sink nodes to the selected VNI
Inter-query time	Frequency of querying the sensor for readings. Only used by sensor-data protocols
Inter-reset time	Time period between the execution of time-triggered topology maintenance protocols
Energy threshold	Energy percentage differential used to invoke energy-triggered topology maintenance protocols. Every time the energy of a node changes this energy threshold value, since the last invocation of the topology maintenance protocol, the node will invoke the protocol again. For example, if the node has all its energy and the value is 0.10, the node will invoke the protocol at 90%, 80%, 70%, etc.

Figure 5: Parameters for deployment creation.

Topology control performance metrics

Topology control protocols are compared based on a specific set of metrics that most of the available tools for simulating wireless networks do not include. The following list enumerates the most important metrics to measure the performance of the topology control algorithms.

For the evaluation of topology construction algorithms, the application includes as performance metrics the number of active nodes per VNI, the overhead in terms of number of packets, and the area of coverage.

For the evaluation of topology maintenance algorithms, the metrics include the number of active nodes reachable from the sink during execution, the lifetime of the network, the number of data messages received by the sink, and the network coverage during execution.

Simulation results

the application offers three types of result logs that can be obtained from a set of experiments: General statistics, network lifetime, and the simulation event logs. The general statistics log

can register the state of several variables in a periodical manner, or provide just one snapshot of the network at the end of the simulation. The most usually consulted variables are the simulation clock, number of nodes, number of sink nodes, number of active nodes, number of dead nodes, average node degree, average level of nodes (if level is used by the protocol), number of messages sent and received, number of data messages received by the sink, energy on the tree, energy spent, and area of communication coverage. These values are stored in a text file where each individual row contains a snapshot of the variables of the network in the moment at which data was collected. If a experiment includes more than one topology, the user can decide if the data is going be stored individually per execution, or if all the results are going to be summarized in a single file. The usual format in which data is presented is in comma separated values (.csv), which is readable by most data analysis programs, like Excel, Matlab, etc. However, if the user does not want them to be saved in files, the Report panel has a text area that holds the statistics generated by the experiments in csv format. If just one topology is simulated, the user can use the Stats text area in the the application panel, which presents the results in plain text format.

The network lifetime log registers the status of the active topology. This log stores information every time the topology maintenance protocol is invoked, or when a node dies. This specific log cannot be stored in individual files per execution; it is stored in a summarized format in csv format.

The information stored in the network lifetime log by a single topology is represented in four rows. The first row registers the moment in which the information of the network was calculated. The second row represents the number of active nodes that can still reach the sink node those that still can provide information to the sink. This value is important because if the sink gets isolated, no matter how many active nodes remain, all of them are useless because the information they produce gets lost. In the case of having more that one VNIs, the program works with the active VNI in the nodes. The third row shows the ratio

between the number of active nodes that can reach the sink (value found on the second row) and the maximum number of active nodes. This value is the percentage of active nodes that are still alive and connected to the sink. Finally, the fourth row contains the percentage of covered area by the active nodes in the second row. This information can be very useful in order to compare efficiency between protocols, in terms of number of active nodes and real covered area.

The simulation event log registers all the events generated by the simulator during the execution of a single topology. The complete set of events allows the user to debug the protocol by seeing the sequence of operations executed. This information is only available in individual files per topology. the application not only offers statistics from the complete simulation point of view, but from the individual node perspective. It is always very useful to know the status of the nodes at any given point in time to check if the algorithms are working as desired. This information can be found in the Node Stats panel. In order to get the information of a node, the user can do it in two ways: dragging the mouse pointer to the desired node and clicking over it, or typing the id number of the node in the text field and press the button Generate Stats. Once the node is selected, the point in the topology turns orange and increases in size.

The application also allows for the visual representation of the reduced topology. The main window of the application is divided in two environments: The deployment visualization area and the control area. One of the panels in the control area contains the visualization options: changing view from MaxPower graph to the reduced topology (the application mode); showing the active nodes (Parent mode), communication and sensing coverage areas, and node ids; drawing a grid over the deployment the area, etc.

3.4 Uninstalling the application

The un-installation process is simple; Simply delete all application files. Terminate the application before performing any file deletion.

4.0 Supervisor's Manual

4.1 Extending the application

Features and limitations

It is not possible to extend this application in a non-programmatical way. Please refer to the Programmer's Manual for more details on altering/expanding the application.

5.0 Programmer's Manual

5.1 Portability

The application is written in JAVA. The source code (NetBeans project format) and the corresponding .JAR file are provided. No portability issues exist. A 1.6.5 JRE compatible JAVA platform is the only prerequisite in any targeted OS.

5.2 Architecture

This section describes the internal structure of the application. First, its main functional components. Then, the structure of the protocols is described in more detail, including how they communicate with the main class and with other protocols, how to initialize the nodes, and how to handle protocol events.

Functional Components

This section describes the applications main functional components and how they interact with each other. The functional components offer the big picture necessary to understand the critical components of the application. The following figure presents a global view of the internal structure of the simulator, which consists of the main simulator thread, the node handler, the batch executor, and the display manager. These elements are described next.

The main simulation thread

This is the core of the system. The simulator thread, defined in the class the sim, is in charge of fetching the next event from the simulation event queue, and sending the event to the node handler for execution. An instance of this class is created by the method StartSimulation() whenever a simulation is executed. This class contains the event queue, the simulation clock,

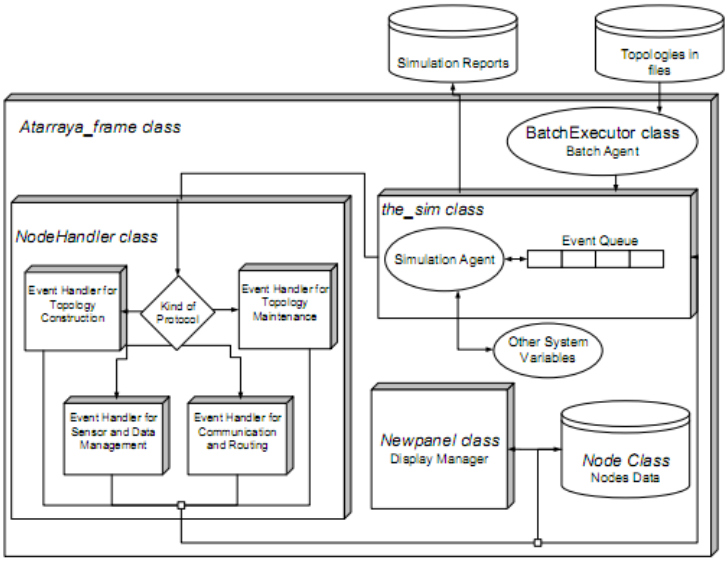

Figure 6: Functional components of the application.

the display manager, the database with the data about the nodes, and the simulation agent, which is in charge of storing the simulation results for the reports in the respective logs.

When an instance of the the sim class is created, it is necessary to add the initial events to the queue before the thread is started. The first thing the thread will do once started is to check if there are any events in the Event Queue. If the thread is started without any events, it will consider that an error has occurred, and the simulation will be suspended. Once the first events have been loaded into the queue, the simulator thread can get started. The thread starts a loop that will execute until one of the three termination conditions is true: there are no events in the queue, all the nodes have reached the final state in the topology control protocol, or the protocols have called for the end of the simulation (for example, the TM protocol has found that the sink has no more neighbors, so the network is dead). If the first condition occurs and the simulator has not been notified that the protocols finished execution, it means that there was an error during the simulation, and it will be notified on the simulation report.

In the loop, the first thing the thread does is to verify if the event is valid. If so, the

event will be registered (if this option was selected by the user), the simulation clock will be updated, and the event will be sent to the NodeHandler. There, the event will be delivered to the appropriate EventHandler according to the protocol. Once the event is executed, the simulator will go back to the loop and start again the process. The simulator updates the clock with the execution time of the events based on the fact that all the events in the queue are sorted by their projected execution time, so there is no such thing like a trip to the past.

Once the simulator breaks the loop by any of the finalization conditions mentioned above, the thread goes to the report construction section, saving all the events and statistics, as selected by the user. This section also takes into account whether or not the simulation is part of a batch execution, in which case all the data from all previous execu tions is kept until the last one finishes. All this information is stored in data structures that are stored in the report files after the simulation is finished. Reading this section of the code will provide the user with information about all the options for each configuration of report in both single and batch simulation cases. Once the simulation and the report building section finish, the thread ends too.

In the current version of the simulator, just one simulator thread can run at a time because there is only one data structure to store the topology, which is localized in the atarraya_frame class. Individual instances of the data structure running several simulations in parallel will consume all the resources of the Java virtual machine, especially if the network topologies are big.

The node handler

This class is in charge of de?ning the protocols to be used in the simulation and routing the event to the appropriate protocol once received from the simulation thread. The NodeHandler class de?nes four possible protocols that a node can have running during a simulation: Topology control, topology maintenance, sensor-data management, and communication-

routing protocols. Given that there are different algorithms for each type of protocol, the main purpose of this class is to make that selection transparent to the rest of the simulator, so that no details about the selection are required in order to execute the simulation. When a simulation is started, this class creates the instances of the selected protocols in each of the four di?erent categories. The simulation thread sends the next event from the event queue to the NodeHandler class. Once the event is received, it is routed to the appropriate protocol based on the protocol identifier included in the event.

The batch executor

The main purpose of the BatchExecutor thread is to perform operations that require multiple executions, such as creating a set of topologies, performing a large number of simulations, and the Giant Component test. Since these operations are run on a thread independent from the main one, the graphical user interface does not freeze while these operations are being executed, which allows for the interaction between the user and the simulator even while some of these operations are running in the background. This class is instantiated whenever one of the mentioned operations is started.

The display manager

The display manager, or newpanel class, is the one in charge of the graphical representation of the topologies. The heart of this class is the override of the Paint method of this class that extends a Panel class. All the painting options for the topology are de?ned in this method. The other methods perform minor but necessary actions like obtaining information about the options, providing coordinates from the deployment area, etc. This class was defined as a private class of the atarraya frame class so it can have direct access to the topology data structure.

The application provides several options for topology visualization, which can be seen in more detail in the visualization options in the main window of the simulator. The most

relevant visualization options are the following:

MaxPower Topology: This is the original view of the topology with all nodes transmitting at full power, and all the links that their unit disks provide.

Single selected network configuration or tree: In this view the user defines which of the Virtual Network Infrastructures he or she wants to see. The default configuration is Black in most protocols.

All network configurations: If several configurations are defined in a certain topology, this view allows the user to see all of them and appreciate the differences between them.

Active network configuration: Each node is assumed to be able to maintain several VNIs, but use only one at a time. This view allows the user to see in realtime in which network configurations the nodes of the topology are.

Protocol Structure and Design

This section introduces the design and structure of protocols. The next subsections describe the types of events that a protocol in the application can model, how states are labeled, how each protocol communicates with the atarraya frame class, how protocols interact with each other, how nodes are initialized, and how the simulator handles events.

Simulation Events and the EventHandler

Given that the application is an event-driven simulator, everything that happens during a simulation is an event, so protocols must be defined in terms of cause-effect when certain event occurs. Each of these types of events triggers some internal actions in the node that might modify its status, data structures, etc., and could also cause the generation of new events in the future. The EventHandler class is the one that model the structure of a protocol in the application and handles all the events. Common examples of events in the application

are sending and receiving messages. Also, the application includes provisions to program an event in the future and invalidate a programmed event.

The HandleEvent method is the core of the protocol, as it defines the actions taken by the protocol when an event occurs. The unique parameter that this method receives is the event taken from the event queue. The events are classified based on an event label. Each protocol defines a set of labels for all the events that it uses. These labels are defined in the constants interface. The first action taken by the HandleEvent method is to recover all the fields from the event and store them in temporary variables. Depending on the nature of the protocol, the classification of the events can be done in different ways: Label-then-State or State-then-Label.

In the first case, the most important information is the label of the event, which becomes the main classification factor. Once the label is found, the code inside determines if the state of the node is important or not for the execution of the actions associated with the event. In the second case, the most important information is the state of the node. This methodology is useful when there are not many types of events but each type is interpreted difierently based on the state of the node.

Node state labels

In general, a good number of topology construction protocols use node states to represent the evolution of the protocol. In the application, nodes can be in any of the following four states: Initial, Active, Inactive, and Sleeping states. These states are assigned to the protocols on the Node Handler class. The values defined as the parameters are usually defined in the constants interface, and they are all positive integer values. For example: tc protocol handler.setLabels(S_INITIAL_STATE, S_ACTIVE, S_SLEEP, S_SLEEP);

Given that most topology control protocols implemented in the application are completely distributed, the sink cannot call the end of the protocol because it has no information about

the state of all the nodes. That is the reason why the application knows that a protocol has finished when all the nodes have reached the final state. Each topology control protocol can define which states are selected as the final states. This is done in the method CheckIfDesiredFinalState(int s) that is defined in every EventHadler, which is invoked in the atarraya frame class when the simulation agent is trying to verify if the topology control algorithm is finished. In the following example, the protocol is selecting the active and the inactive states as the final states of the nodes.

```
public boolean CheckIfDesiredFinalState(int s)
{
if(s==active ||\| s==inactive)
return true;
return false;
}
```

the application stops whenever the nodes of the topology are in any of the selected states, no matter if there are still events in the queue.

Communication with the atarraya_frame class

Each protocol receives a reference to the instance of the applications main frame, as defined on the NodeHandler. This reference allows the protocol to have access to variables from the main class. In order to access the variables from the simulator, the protocol needs to use the method father.getVariable(int code), where the parameter code is a defined label for the sets of variables that can be accessed. This list can be found in the constants interface, and in the atarraya frame class where the getVariable() method is defined.

For example, if we need to know how many nodes are in the topology (including the sink nodes), the following line returns this value:

```
tam = father.getVariable(NUMPOINTS)
```

When the protocol needs to get information about a node or modify it, the method to use is getNode(int id), where id is the unique id of the node. In order to set node i in the initial state of the protocol in the VNI vniID, the following line can be used:
getNode(i).setState(initial,vniID).

Interaction with other protocols

There will always be some level of communication between protocols. For example, inter-protocol communication is needed to avoid situations like one node wanting to send a data message without having a route to the sink. Given that in the application every event in the simulation goes to the same queue, it is necessary to determine to which protocol it must send the event to. Each type of protocol has its own identifier label, which is included in the event definition. This allows the Node Handler to send the event to the appropriate protocol.

One of the premises of the application is to create modular protocols that can be used in as many combinations as possible with the other protocols. Accordingly, protocols in the application can only generate events in other protocols. For example, once a node reaches the final state of its topology construction algorithm, it can notify the topology maintenance protocol to start the maintenance procedure. Most of the times these inter-protocol events are meant to initiate or stop certain activity, so the protocol and how it works internally are completely independent, but the other protocols can decide the starting points. The following ex- ample illustrates a topology construction protocol when it invokes the topology maintenance protocol:
pushEvent(new event sim(temp clock+DELTA TIME, temp clock, receiver, receiver, INIT_EVENT, , temp tree,TM PROTOCOL)).

Initialization of nodes and the initial events

The init nodes(int vni) method is used to set the nodes ready to start the execution of the simulation. Nodes are set to their initial states, and any previously defined events regarding other protocols and all necessary variables are set to their default values. This method is invoked in the Start- Simulation method in the atarraya frame class, for all nodes, including the sink. The following code is an example of a init nodes routine, in which every node is set to its initial state, every state label is defined, any existent programmed event in the queue is canceled, and the execution of the topology maintenance and sensor and data management protocols are reset. The initial event(int _id, int _vniID) method is used to define the first events to be included in the queue, before the simulation agent is started. Remember that if the simulation agent finds an empty queue it considers that the simulation has finished in an incorrect way. This method needs two parameters: the ID of the node that will perform the first event, and the VNI_ID. This method is also invoked in the StartSimulation() method in the atarraya frame class, but only for the sink nodes. If no sink nodes are defined in your topology, make sure that events are included in the queue using the init_nodes(_vniID) method.

References

[1] Pedro Wightman and Miguel A. Labrador, Atarraya: A Simulation Tool to Teach and Research Topology Control Algorithms for Wireless Sensor Networks , ICST 2nd International Conference on Simulation Tools and Techniques, SIMUTools 2009, February 2009.

[2] Miguel A. Labrador , Pedro M. Wightman, "Topology Control in Wireless Sensor Networks: with a companion simulation tool for teaching and research", Springer; 1 edition (February 23, 2009).

[3] Paolo Santi, "Topology Control in Wireless Ad Hoc and Sensor Networks", Wiley (September 2, 2005).

[4] topo3 Holger Karl, "Protocols and Architectures for Wireless Sensor Networks", Wiley-Interscience (October 26, 2007).