Simulation Modeling of Secure Wireless Sensor Networks [*]

Tuan Manh Vu   Carey Williamson   Reihaneh Safavi-Naini

Department of Computer Science
University of Calgary

ABSTRACT

This paper describes an extensible simulation environment for the modeling of wireless sensor networks (WSNs). In particular, our simulator facilitates the study of secure connectivity between sensor nodes. The simulator has five main components: network topology model, key establishment protocol, adversary model for node capture, network analysis tools, and a graphical user interface (GUI) to facilitate rapid simulation, visualization, and analysis of WSNs. We present the design and implementation of our simulation tool, including the methodologies and algorithms underlying each component, and the data flow dependencies between them. Additionally, we describe the data collection and analysis functions integrated within the simulator, and how these can be used for in-depth simulation studies. We have used the simulator to verify asymptotic analytical results for secure WSNs, as well as to investigate their structural characteristics. We present selected results to demonstrate the use and value of our simulation tool.

Keywords

Wireless Sensor Networks, Key Establishment, Simulation

1. INTRODUCTION

Wireless sensor networks (WSNs) are emerging as a promising technology with a wide range of potential uses, including environmental monitoring, building surveillance, and military applications. A typical WSN consists of many (perhaps thousands) of sensor nodes deployed over a region to observe phenomena of interest. Each sensor node is a small battery-powered device with sensing hardware to measure one or more physical conditions, such as temperature, humidity, pressure, sound, light, or radioactivity. Sensors have a short-range wireless radio for communication, as well as limited storage (memory) and computational (CPU) resources. Sensors can be programmed to work collaboratively by collecting, exchanging, and forwarding the sensed data to WSN base stations, where data is processed and analyzed further.

Two important considerations when deploying WSNs are *connectivity* and *security*. The WSN is *connected* if there exists a physical (wireless) communication path from any sensor node to any other node, including the base station. These paths are typically multi-hop, relying on intermediate sensor nodes to forward data. The WSN is *secure* if all communication paths, including the intermediate hops, are protected by pair-wise secret keys, so that all communication can be encrypted.

In this paper, we are interested in WSNs that are *both* connected and secure. In a secure WSN, a malicious adversary cannot intercept communication between sensor nodes, and cannot disrupt network operations by broadcasting bogus information. Depending on different applications, sensor nodes may be deployed with unpredictable topologies. Furthermore, because of the resource constraints of sensors, conventional cryptographic protocols, such as Diffie-Hellman [6] and Kerberos [8], may not be practically applicable. There have been several *probabilistic* key establishment schemes specifically developed for WSNs that allow sensor nodes to compute pair-wise secret keys. These schemes can provide secure communication, while ensuring good network connectivity. Due to the unpredictable deployment of sensor nodes, simulation plays an important role in the evaluation of these key establishment schemes.

Our work:

In spite of the growing interest in WSN research, there are few public-domain tools for the simulation and analysis of large-scale WSNs. The *ns-2* network simulator [9] has many built-in features for the simulation of TCP, as well as routing and multicast protocols, but is not well-suited to the study of secure connectivity in large WSNs with thousands of sensor nodes. For these reasons, we developed our own tool for WSN simulation, primarily to evaluate our own new approach for establishing secure communication between sensor nodes. We have since redesigned, extended, and general-

ized this tool by adding support for other key establishment methods, and improved the functionality and usability for WSN network analysis. We have found this simulation tool very useful in our work, and believe that it will also be beneficial for others.

In this paper, we describe an extensible simulation environment for modeling secure WSNs. We have used it to verify asymptotic analytical results for different key establishment methods. Also, when it is not possible to study WSNs mathematically, one can use this tool to model and predict the performance of the simulated WSNs.

The simulator has distinct components for tasks such as network topology generation, secure connectivity establishment, and node capture by an adversary, as well as postprocessing tools for network analysis and graphical visualization. Currently, we have implemented two physical deployment models for sensor nodes, three key establishment methods, and one simple strategy for the adversary to capture nodes. Each component can be extended and integrated into the simulator seamlessly, as long as the new implementation conforms to the overall design API. Extensibility allows one to easily enhance the system in order to study WSNs under various assumptions.

To simplify WSN analysis, we implement a set of widely-known *graph algorithms* (i.e., shortest path, connected components) as well as some custom algorithms to study node degree, wireless connectivity, and key graph properties.

Our prototype runs in a Java Virtual Machine (JVM) environment on a typical desktop or laptop. In our testing environment, with a 4-CPU Pentium (3.4 GHz) and 1 GB of RAM, we can simulate WSNs of 1,000 to 10,000 nodes in about 1 second. Using our tool, it is simple to simulate a WSN multiple times (e.g., 1000 runs) to produce average results regarding connectivity. Simulation output data can also be redirected to a file, and then fed to external utilities such as a statistical package for analysis, or gnuplot [2] to produce graphical visualization.

The remainder of this paper is organized as follows. Section 2 provides background information on probabilistic key pre-distribution schemes, as well as terminology and metrics analyzing key establishment approaches. Section 3 provides an overview of our simulator, including its core components, their dependencies, and the data flow between them. Details about each component appear in Section 4 to Section 7. Section 8 presents some simulation results obtained from our tool. Finally, Section 9 concludes the paper.

2. BACKGROUND AND RELATED WORK
This section provides some brief background information on the theory of random graphs [10], which are often used in the analysis of secure WSNs. It then describes two prominent probabilistic key pre-distribution schemes from the literature, and how these methods rely on random graph theory to achieve the desired network connectivity. The notation and terminology used to discuss and evaluate key establishment schemes are briefly explained.

2.1 Random Graph Theory

A random graph $G(n, p)$ is a graph of n nodes in which the presence of an edge between any given pair of nodes is equally likely; in particular, the edges are selected independently with probability p. An asymptotic property of random graphs is that as n approaches infinity, $G(n, p)$ is connected with probability c if $p = \frac{1}{n} \cdot (\ln(n) - \ln(-\ln(c)))$. In other words, if the average node degree n_t in a random graph $G(n, p)$ is equal to $\frac{1}{n} \cdot (\ln(n) - \ln(-\ln(c))) \cdot (n - 1)$, then the graph is connected with probability $c \in (0, 1)$.

2.2 Probabilistic Key Pre-distribution
2.2.1 Eschenauer and Gligor's scheme
Eschenauer and Gligor (EG) [5] pioneered a probabilistic key pre-distribution method that preloads sensors with a set of keys prior to deployment. First, a general *key pool* with many (e.g., 100,000) random keys is created. Next, each sensor is loaded with a fixed-size set of keys (e.g., 300) drawn uniformly at random from the key pool (without replacement). This set of keys, called a *key ring*, contains the keys as well as their corresponding *key identifiers* (key IDs). Once deployed, each sensor node advertises (broadcasts) to its *wireless neighbors* the IDs of keys that it possesses. Two sensor nodes that share at least one common key can establish a secure communication channel, called a *secure link*. Two wireless neighbors that can communicate over a secure link are called *wireless trust neighbors*.

The secure connectivity in a WSN can be seen as a graph $G(V, E)$ in which each vertex represents a sensor node and an edge exists if and only if the two corresponding nodes have a secure link. The graph $G(V, E)$ is referred to as a *wireless trust graph*.

Given the assumptions that the sensor nodes are scattered randomly and keys are distributed arbitrarily, Eschenauer and Gligor modeled the wireless trust graph as a random graph. As described earlier, a random graph of n nodes is connected with probability c if the average node degree is $n_t = \frac{1}{n} \cdot (\ln(n) - \ln(-\ln(c))) \cdot (n - 1)$.

The average node degree can be determined based on network topology information and key distribution parameters. In particular, if N is the key pool size, and m is the key ring size, then the probability that two key rings share at least one common key is $\bar{p} = 1 - \frac{((N-m)!)^2}{N! \cdot (N - 2 \cdot m)!}$ [5]. From the wireless connectivity constraint, a sensor node can only set up secure links with its wireless neighbors. Hence, the expected node degree in a wireless trust graph is $n_t = n_w \cdot \bar{p}$, where n_w is the *network density* (i.e., the average number of sensor nodes in a neighborhood).

In short, given the network size n and the desired connectivity c, the expected node degree can be estimated by random graph theory. To achieve the desired node degree, either the the key pool size N or key ring size m can be set to a fixed value, and the other can be determined accordingly.

2.2.2 q-composite scheme
Chan *et al.* [3] proposed the q-composite scheme as an enhanced and generalized version of Eschenauer and Gligor's work. This scheme allows two wireless neighbors to establish a secure link if they have at least q common keys, where

$q \geq 1$. The pair-wise secret key is constructed as the hash value of the concatenation of all keys in common between two wireless neighbors.

The probability that two random key rings share at least q common keys is $\sum_{i=q}^{m} s(i)$, where $s(i) = \frac{((N-m)!)^2 \cdot (m!)^2}{N! \cdot (N-2 \cdot m+i)! \cdot i! \cdot ((m-i)!)^2}$ is the probability that two key rings have exactly i common keys [3]. Similar to the analysis of the EG scheme, if the key pool size or key ring size is fixed, the other can be determined so that the node degree in the wireless trust graph meets or exceeds the desired value, making the network connected with high probability (based on random graph theory).

However, several authors have observed that the construction of the wireless trust graph differs from that for a random graph [4, 11]. In a WSN, the probability that a secure link can be established is not really independent between every pair of sensor nodes. There are two factors that affect this edge probability between pairs of sensor nodes: the physical deployment of the WSN, and the way the keys are distributed to sensor nodes. The result obtained from random graph theory may not be accurate for certain network topology and key parameter settings. In Section 8, we compare analytical results with simulation results for secure WSNs.

Other than the network connectivity, two important criteria for evaluating a key establishment scheme are the *communication overhead* and the *resilience against node capture*. *Communication overhead* measures the volume of data (bits) exchanged between sensor nodes to complete the key establishment process. Our system reports this metric for the simulated WSNs, in addition to the connectivity information. *Node capture* by an adversary is also a concern. Through physical capture and control of a given sensor node, an adversary can gain full access to all the information stored in a node, including its keys. Using this information, the adversary may be able to monitor, intercept, or alter data being transmitted on many links in the WSN. Such links are said to be *compromised*. The resilience against node capture is measured by the proportion of uncompromised links remaining in the network after a specified number of nodes are captured.

3. SIMULATOR OVERVIEW
This section provides an overview of our simulator, the system requirements, and briefly highlights the main challenges in the implementation of the simulator. We also discuss the main components of the simulator, their dependencies, and the data flows between them.

3.1 System Requirements
Our simulator is developed in Java 1.6, and thus can run on any Linux, MacOS, Unix, or Windows system with support for a Java Virtual Machine (JVM). While most of the user interface is written in Java Swing GUI, gnuplot is used to generate graphical output. In order for the simulator to function fully, an appropriate platform-specific version of gnuplot is needed. A complete stand-alone version of our simulator for a Windows XP environment is available at [1].

Our tool can simulate a WSN of any size, as long as the required memory is available. For hardware requirements, at least 64 MB RAM is recommended to simulate WSNs of size 1,000. To study larger networks with up to 10,000 nodes, 512 MB RAM or more is sufficient. As mentioned earlier, our testing was done on a Pentium(R) 4 CPU 3.4 GHz and 1 GB of RAM, which is enough to simulate a WSN with up to 50,000 nodes, assuming a moderate network density (e.g., from 30 to 60).

Execution time for simulations depends on network size and the details of the analysis required. Some analysis routines take significantly longer running time and more memory than others, depending on the complexity of the algorithm used. For instance, calculating the path lengths between every pair of nodes has a running time of $O(n^3)$ and memory space requirement of $O(n^2)$, where n is the network size.

3.2 Implementation Challenges
The simulator would be used for simulation of WSN with thousands of nodes, where each node has a large key ring (e.g., 300 keys). Important properties of interest are connectivity of the network, both for wireless and wireless trust graphs in addition to the key-sharing connectivity between nodes. To determine connectivity one needs to consider any pair of nodes. Therefore, it is crucial to have highly efficient implementation in order to be able to run the simulator for large networks. Efficiency of the simulator depends on the efficiency of the algorithms, data structures and calculations on large data sets. We use *adjacency list* with some modifications instead of *adjacency matrix*, together with *linked list* and *queue* to efficiently store all necessary data.

Also, the simulator can be used to calculate quantities analytically for validating the simulation results. These calculations in many cases are computationally intensive. For example, finding the expected probability that two key rings, each of size 300, have at least four keys in common given that the key pool size is 100,000, performs arithmetic operations on very large integers. This and similar calculations need to be optimized for fast performance.

3.3 Main Components
To simulate a WSN and analyze the performance of a key establishment scheme, there are essentially three steps. First, the locations of sensor nodes are generated, which determines the wireless connectivity based on the specified communication range. Second, a particular key establishment scheme is used to establish secure links where possible between wireless neighbors. Third, the captured nodes are chosen and the compromised links are determined accordingly. The third step is optional, and is only required if one wishes to study the effect of node capture. After the three steps are complete, one can use the available network analysis functions to study the simulated WSN, such as its connected components, path length between nodes, distribution of wireless trust neighbors, secure connectivity, fraction of compromised links, and so on.

Since the process of WSN simulation can be separated into distinct steps, we build several components to accomplish different tasks. Figure 1 illustrates the five main components in our framework: *physical deployment*, *key establishment*, *node capture*, *network analysis*, and *graphical user interface*.

In the following, we explain the main purpose of each component, as well as its input and output dependencies.
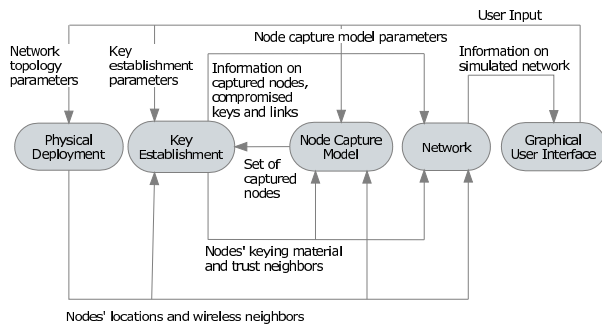

Figure 1: Main components in WSN simulator

Physical Deployment: This component defines the classes used for creating the physical layout of a WSN. Depending on the user-specified network topology and deployment region, a sensor node's location can be generated either randomly or deterministically. Repeating this process for all nodes produces the complete physical deployment of a WSN, after which the wireless connectivity between nodes is automatically determined. The information on the locations of nodes and their wireless neighbors is accessible by other components.

Key Establishment: This component defines the classes used for establishing secure links in the WSN, given the known locations of nodes, their wireless neighbors, and the keying material available at each node.

Each sensor node is initially assigned different keying material according to a user-specified key establishment method, as well as the key pool size and key ring size (in the case of EG and q-composite schemes). Trust neighbors and their pair-wise keys are determined following the chosen protocol. The communication overhead for the key negotiation phase is also recorded.

The 'key establishment' and 'node capture model' components have mutual dependencies. For example, suppose that the 'node capture model' component selects nodes that are captured by an adversary. Given the set of captured nodes, the 'key establishment' component identifies what information is accessible to the adversary, and subsequently which secure links in the WSN are compromised.

The information on keying material of the nodes, trust neighbors, pair-wise and captured keys, as well as the secure and compromised links is made available to other components.

Node Capture Model: When a WSN is deployed in a hostile area, an active adversary might capture one or more sensor nodes. If this happens, all of the information stored in the memory of the captured node is available to the adversary, which can compromise secure links elsewhere in the WSN. To study the resilience of a WSN against node capture, one needs to estimate the proportion of links that remain secure when one or more sensor nodes are captured.

An adversary may have different strategies for capturing nodes. The simplest is a random capture model, where captured nodes are selected uniformly at random. Another is a spatial approach, where all sensor nodes in a certain geographic area of the deployment region are captured. A third possibility is an adaptive adversary, who always attacks the most sensitive nodes in the network. Such an adversary tries to acquire as much information as possible, maximizing the chance to compromise links in the remaining network.

The 'node capture model' component enables one to study the effect of different node capture strategies on link compromise. In particular, a node capture model specifies how nodes are selected one-by-one by the adversary. Given the adversary's strategy, the user-specified number of captured nodes, as well as the locations of nodes and their keying material, this component returns a set of captured nodes.

Network Analysis: This component provides a suite of graph algorithms and statistical tools to assess the strength of a simulated WSN. This component uses 'physical deployment', 'key establishment', and 'node capture model' components to simulate a complete WSN. The analysis functions provide information about the simulated WSN.

Graphical User Interface: This component allows users to easily set up parameters for network topology, key establishment, node capture model, and number of replications for the simulation. Once the WSN has been simulated, one can study the result visually such as WSN layout, wireless connectivity, secure connectivity, and so on. Visualization options are easily settable for customized simulations.

4. PHYSICAL DEPLOYMENT COMPONENT
This section explains the 'physical deployment' component, specifically the classes, methodologies, and algorithms used to generate different network topologies.

4.1 Abstract Classes
Figure 2 shows a simplified UML diagram of the 'physical deployment' component including two abstract classes (i.e., *Location* and *Deployment*) and two implementations of each.
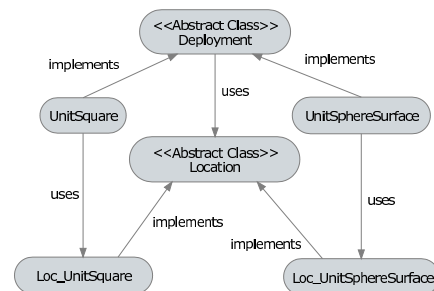

Figure 2: Physical deployment (topology) models

The *Location* class describes abstract properties of a physical location of a sensor node when deployed in a particular region. Any sub-class of *Location* must implement a method to determine if two sensors are within a given communication range or not. How the communication range is defined depends entirely on the implementation.

The *Deployment* class contains attributes and methods of an object that represents a WSN deployment. The default constructor takes two parameters: the network size, and the communication range. The most important method in this abstract class generates a single node's location. It is an abstract method, which must be implemented by sub-classes. Repeatedly calling this method generates the locations of all nodes. We assume that node locations remain static throughout the simulation, and that all nodes have homogeneous communication range. Another important method is used to determine wireless connectivity in the network. This method is non-abstract, and provides a default implementation to determine wireless connectivity. Basically, it loops through every possible node pair, and checks if they are within communication range of each other or not. A sub-class can override this default method, if desired. For example, execution speed may be improved by exploiting specific characteristics of a deployment (e.g., regular grid layout).

We implement two examples of deployment types. The first type assumes that sensor nodes are scattered uniformly at random within a unit square; this is a common model when studying WSNs [4]. One issue with this type of deployment is *boundary effects*: nodes near edges and corners have fewer wireless neighbors than other nodes. Thus, after the key establishment phase, these nodes are more likely to be isolated from the network. This leads to discrepancies between analytical results and simulation results, even when the network has 1,000 nodes or more. (Section 8 discusses this issue in more detail). In order to avoid boundary effects, our second deployment type considers sensor nodes that are distributed uniformly at random on the surface of a unit sphere (e.g., to model a planetary deployment). Although this is not a realistic model for most practical WSNs, it can be useful when studying WSNs without the boundary effect.

In the following, we describe the two implementations of WSN deployment in our system.

4.2 Sub-classes

4.2.1 Unit Square Deployment
Location and Communication Range
Each location in a unit square is given by x and y coordinates. To generate a random location with uniform distribution, we use the default random number generator provided by Java Development Kit (JDK) 1.6 to set each coordinate to a random value between 0 and 1. The distance between two nodes is evaluated as the Euclidean distance between their locations.

To reduce the computation time for wireless neighbor discovery, our implementation sorts the locations of all nodes according to their x coordinate. This technique substantially reduces the number of node pairs that must be considered when determining wireless connectivity. This simple optimization significantly improves the execution time (e.g., 10-100 times faster than a brute force approach), especially when the WSN is very large (e.g., up to 10,000 nodes).

Determining Communication Range
Using the default constructor, one can generate a WSN physical deployment given a network size and a communica-tion range. However, in some cases, it is more desirable to generate a network deployment given a network size and a specified average node density.

Analytically, we have derived a formula for the relationship between network size, average density, and communication range. Specifically, if n nodes with communication range r $(0 < r < 1)$ are distributed uniformly at random within a unit square, then the average network density n_w is:

$$n_w = (n-1) \cdot (\pi \cdot r^2 - \frac{8}{3} \cdot r^3 + \frac{1}{2} \cdot r^4) \qquad (1)$$

With this model, the simulation user can specify *either* the communication range or the network density for a given WSN size; the other value is computed automatically by the simulator.

4.2.2 Unit Sphere Surface Deployment
Location and Communication Range
In the unit sphere model, each point is represented by three Cartesian coordinates (x, y, z). We employ Marsaglia's method [7] to generate random points on the unit sphere's surface with a uniform distribution. The distance between two points is computed as the great-circle distance between them (i.e., path along the surface, not through the interior).

Determining Communication Range
We have also analytically determined the relationship between network size, communication range, and network density in the unit sphere model. Given n sensor nodes distributed uniformly at random on a unit sphere's surface, if the desired average density is n_w, then the communication range r is given by:

$$r = acos(1 - \frac{2 \cdot n_w}{n-1}) \qquad (2)$$

5. KEY ESTABLISHMENT AND NODE CAPTURE MODEL COMPONENTS
This section discusses the 'key establishment' and 'node capture model' components, as illustrated in Figure 3.

5.1 Key Establishment Scheme
Class *Scheme* defines the abstract design of an object representing a generic key establishment scheme. Examples of the abstract methods are a function to determine if two nodes share a key, a function to return the status (compromised or not) of a secure link, and a function to compute the proportion of compromised links in the network.

There are different categories of key establishment schemes, based on how they work. However, we focus only on probabilistic key pre-distribution approaches. *ProbabilisticPredistributionScheme* is an abstract class that extends *Scheme*, specifying additional methods of a generic probabilistic key pre-distribution scheme, such as key ring generation. A sub-component that facilitates key ring manipulation is described in Section 5.2.

We have fully implemented three schemes for probabilistic key pre-distribution: EG, q-composite, and our own multi-round scheme. The multi-round scheme is our ongoing research and due to the nature of this paper, we do not go into the details of this scheme.
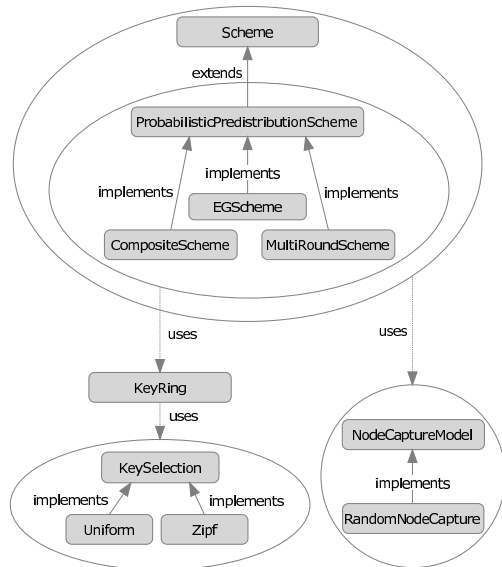
Figure 3: Key establishment and node capture

5.2 Key Ring and Key Selection

The key ring sub-component provides a method to construct a random key ring given the key pool size. Normally, a fixed number of keys are drawn uniformly at random from the key pool to create a key ring. Nevertheless, one may be interested in how different key selection methods influence the performance of a particular probabilistic key pre-distribution scheme (e.g., heterogeneous key ring sizes, non-uniform key selection).

Class *KeyRing* can use any implementation of the abstract class *KeySelection* to choose keys from the key pool when forming a key ring. We have implemented two key selection methods: keys can be chosen with Uniform or Zipf distributions. Additionally, the class *KeyRing* provides a set of functions for basic key ring manipulations, including union, intersection, and set membership operations.

5.3 Node Capture Model

The abstract class *NodeCaptureModel* defines a method that returns a set of captured nodes. Currently, we provide one simple model, class *RandomNodeCapture*, which selects x captured nodes (where x is the input parameter) uniformly at random from the set of all sensor nodes.

6. NETWORK ANALYSIS COMPONENT

In this section, we describe the most essential analysis functions provided by the 'network analysis' component.

6.1 Connectivity

Connected components

We provide a function to find all connected components and their sizes by doing *breadth-first search* on a wireless trust graph. From our observations, in many cases, when the network is disconnected only a few nodes around the boundaries and corners are isolated from the rest. In other words, despite the fact that the network is disconnected, the largest connected component consists of most of the nodes. Simply saying a network is connected or disconnected may not provide enough information when analyzing the connectivity. For this reason, we compute and report all connected components.

Network connectivity

To determine if the network is connected or not, we provide a simple function that compares the size of the largest connected component to the network size. If they are equal, then the network is connected. Otherwise, the network is disconnected.

Coverage area

In some applications, it is useful to know the geographic area covered by a set of sensor nodes that are connected to each other. We provide a function to estimate the coverage area of one or more connected nodes.

Path length distribution

Another useful way of analyzing WSN connectivity is to measure the average path length between nodes. This metric provides an indication of the average power consumption for WSN communication. We provide a function implementing Floyd's algorithm (all pairs shortest paths) on the wireless trust graph. It computes all paths, and then returns the distribution of the path length between every pair of nodes.

6.2 Key Ring Size and Key Usage

Initial key ring size

The key ring size is important in probabilistic key pre-distribution schemes. Large key rings improve the chances of connectivity, but they also require more memory in sensor nodes, and increase the risk of link compromise when a node is captured. Often one needs to estimate how many keys should be assigned to each sensor node so that there is a high probability that the deployed WSN is connected. We provide a small program in the data collector package to determine this value via simulation. It performs *binary search* to find the smallest initial key ring size so that the deployed WSN is connected with high probability (i.e., at least i times out of j simulation runs, where i and j are specified values for a given network topology and key pool size).

Key usage

In most probabilistic key pre-distribution schemes, the assigned keys from the key pool are rather sparsely used. We provide a function to examine the key usage distribution, particularly, how many keys are used once, twice, and so on. This usage distribution affects resilience to node capture.

Number of compromised keys

To study how many keys from the key pool are obtained by an adversary who captures x nodes (using a particular node capture model), we provide a function that reports all the compromised keys.

6.3 Other Analysis Functions

In addition to the functions mentioned above, other useful analyses compute the distribution of wireless neighbors, the distribution of trust neighbors, the proportion of compromised links, and the probability that a link is compromised if multi-path key reinforcement technique [3] is applied.

(a) A network with 10 nodes   (b) A network with 1,000 nodes
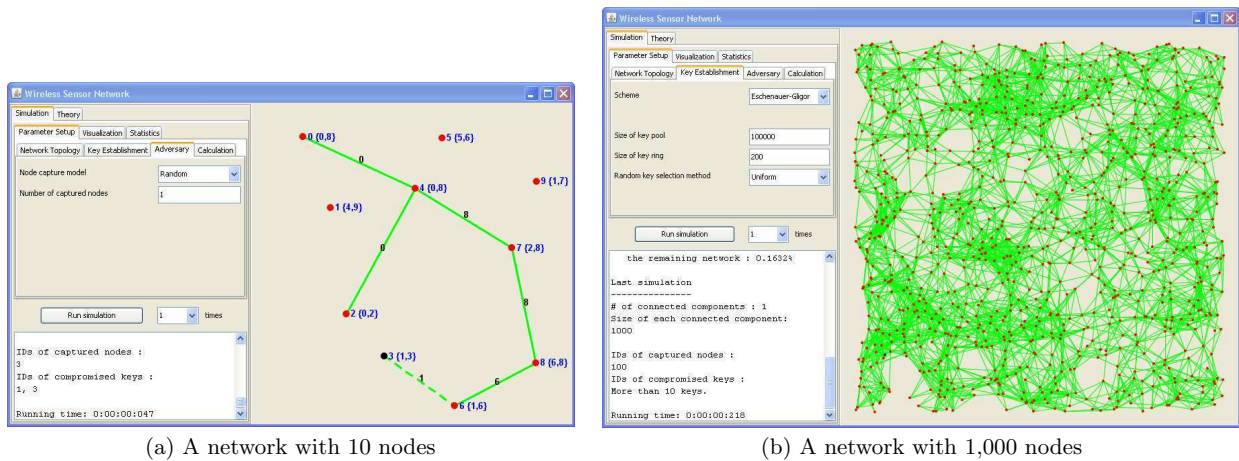
Figure 4: Screenshots of GUI for WSN simulation tool

For almost every analysis function on a single WSN, there is a corresponding program in the data collector package to calculate the average result from a user-specified number of simulation replications.

7. GUI COMPONENT

Figure 4 shows example screen shots of the graphical user interface of our simulation tool. Figure 4(a) shows a small 10-node WSN, while Figure 4(b) shows a large 1000-node WSN. In both examples, the rightmost pane shows the WSN layout, while the left side of the interface shows input parameters and output results from the simulation.

There are two main control tabs on the top left of the GUI windows: *Simulation* and *Theory*. The *Theory* tab allows the user to study the analytical results of random graph theory, probability of key sharing, and so on with different parameters for comparison with the simulation results. We concentrate here on the other main tab, *Simulation*. Under *Simulation*, there are three sub-tabs that let the user enter simulation parameters, customize visualization, as well as select and view statistics and graphs based on the simulation results.

One parameter that the user needs to specify is the number of simulation replications. This parameter (default value is 1) can be changed in the data field next to the 'Run simulation' button in Figure 4. Once this button is hit, a WSN is simulated a number of times as stated by the users. When the simulation is done, the result appears on the text area beneath the 'Run simulation' button. Specifically, this text area summarizes the simulation parameters and provides the simulation results including the average number of trust neighbors, the network connectivity, connected components, the number of compromised keys, the fraction of compromised links, running time, and so on.

The rightmost portion of the GUI in Figure 4 shows the network visualization from the *most recent* simulation run. In the current system, we only provide the visualization for the unit square deployment model. We have no 3D visualization support for the spherical model.

Using the GUI to study WSNs, the user first needs to specify parameters for simulation. Once the network has been simulated, network visualization can be customized to show different information. Furthermore, the user can examine the detailed (text-based) summary report from the simulation, or view the graphs generated from the simulation data. The following paragraphs elaborate on these steps.

Setting simulation parameters
To simulate a WSN, the user needs to select parameters for network topology, key establishment scheme, and (optional) node capture model. Besides, the user has to specify which analysis routines should be included in the simulation (for efficiency reasons), and the number of simulation replications.

Figure 5 shows some of the menu selections and parameter controls for the user. Most simulation parameters can be set manually or chosen using drop-down menus and check boxes.

Inside the *Network topology* tab (Figure 5(a)), the user can select the deployment type, and then set the network size and either communication range or network density. The data fields for network size and density are drop-down lists that consist of pre-defined common values. The text boxes are also editable so that the user can define any value they choose. As mentioned in Section 4.2, the communication range and the desired network density are related. Assuming that the user has chosen a network size and deployment type, whenever the desired density or the communication range is changed, the other setting is automatically calculated and updated accordingly.

From the *Key Establishment* tab (Figure 5(b)), the user can choose key establishment scheme, key pool size, key ring size, and key selection method.

The *Adversary* tab (Figure 5(c)) specifies the node capture model (only one model currently) and the number of captured nodes. The default value is 0, which means there is no adversary in the simulated WSN.
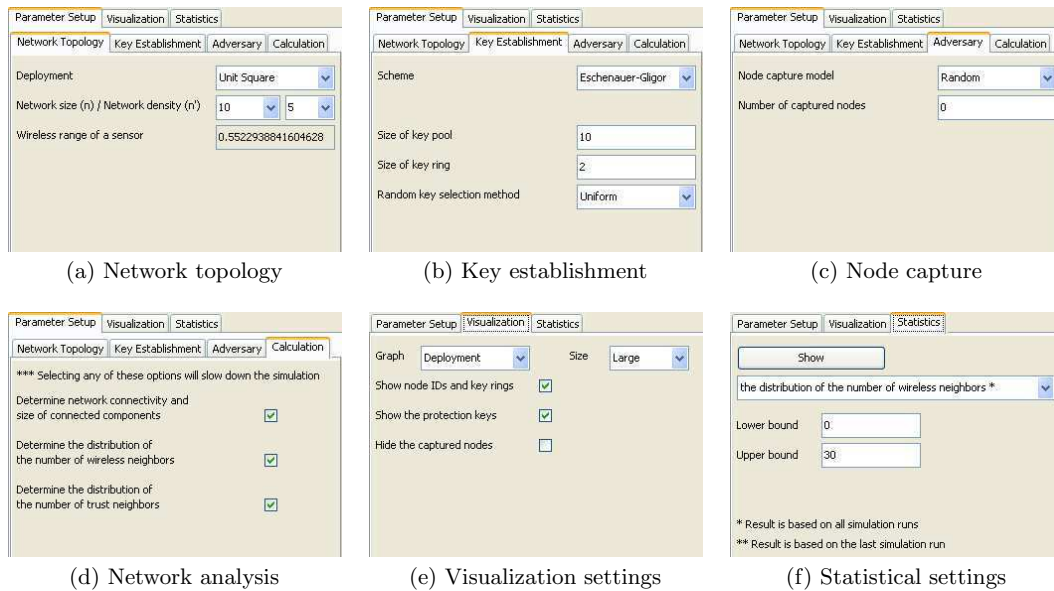
(a) Network topology   (b) Key establishment   (c) Node capture

(d) Network analysis   (e) Visualization settings   (f) Statistical settings

Figure 5: Setting simulation parameters

The *Calculation* tab (Figure 5(d)) allows the user to specify which network analysis functions are desired. There are some computationally expensive routines in the analysis of simulated WSNs which can be deactivated, avoiding extra overhead when simulating WSNs if the user is not interested in those analyses. By default, all of these options are selected.

Customizing visualization
Figure 5(e) shows the 'Visualization' tab with various settings for customization. The drop-down list of graph types lets the user see different information about sensor nodes. The available options are *wireless range* (the circular communication range of each sensor node), *wireless connectivity* (nodes that can communicate directly with each other), the *key graph* (nodes that share at least one key), and the *wireless trust graph* (secure connectivity).

To improve the clarity of the visualization, the user can select one of the three sizes: 'Small', 'Medium', or 'Large'. This setting scales the font size used, as well as the size of the dots and lines used to illustrate the WSN. For a large network, the visualization looks better with 'Small', and vice versa. The user also has an option to show or hide the node IDs, key rings, and protection keys of secure links, as well as the captured nodes and their attached compromised links.

The network visualization shown in Figure 4a is a wireless trust graph of 10 sensor nodes. In this example, the key establishment scheme is EG, and each node is assigned two keys chosen randomly from a key pool of 10 keys. It is assumed that one node is captured by the adversary. In the visualization, the black dot indicates the captured node, while the red dots represent the other (uncaptured) nodes. For each node, we can see its ID followed by the IDs of keys that it has. In this example, node 3 is captured, and thus the adversary has keys 1 and 3. There is one compromised link

in the network, which is shown by the dashed line between nodes 3 and 6. The links that remain secure are shown with solid lines.

Graphical results
Figure 5(f) shows the options available under the *Statistics* tab. In this tab, the user can choose to see the distribution of the number of wireless neighbors or trust neighbors, and the distribution of path lengths based on the simulation result. The simulation data is fed into gnuplot to generate graphs, which are then presented to the user. Figure 6 shows an example diagram of the distribution of path length in a simulated WSN.
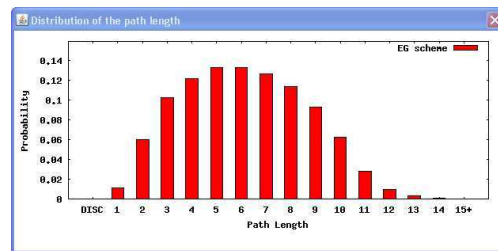

Figure 6: A diagram showing the distribution of path length in the simulated WSN

8. APPLICATIONS AND SIMULATION RESULTS

This section presents selected simulation results to demonstrate the functionality and value of our simulation tool for secure WSNs. First, we validate the theoretical results of network connectivity and link compromise probability in the EG and *q*-composite schemes. Next, we examine the boundary effects on network connectivity by i) comparing the predictions and experimental results of key ring size and

ii) investigating the difference of the distribution of node degree in random and wireless trust graphs. Finally, we use the simulator to study if the boundary effects become less noticeable in larger networks.

8.1 Simulation model validation

Network connectivity

Security analysis of EG protocol and its extension, q-composite scheme, relies on modeling the key pre-distribution by a random graph. As the first application of the tool, we examine the validity of this assumption. Assume that we want to deploy a network of 1,000 nodes in which the nodes are scattered uniformly at random on the unit sphere with an average density $n_w = 30$ nodes. The desired connectivity is 99.9% and the key pool size N is 100,000.

According to random graph theory, a network of 1,000 nodes should be connected 99.9% of the time if the expected node degree n_t exceeds 13.8. Let $\bar{p}$ be the probability that two random key rings can be used to establish a pair-wise key. To satisfy $n_t > 13.8$ for $n_w = 30$, where $n_t = n_w \cdot \bar{p}$, the initial key ring size m must be chosen such that $\bar{p} > \frac{13.8}{30}$. For a key pool size $N = 100,000$, using the equations in Section 2.2, we require $m = 249$ in the EG scheme, and $m = 394$ for the q-composite scheme when $q = 2$.

We ran the simulation experiments using these *theoretical* key ring sizes for EG and q-composite schemes on the unit sphere. The result shows that the network is connected 9,992 times out of 10,000 simulation runs for the EG scheme, and 9,994 times out of 10,000 simulation runs for the q-composite scheme. These results confirm the mathematical analysis. The distribution of node degree is also very similar to that in a random graph.

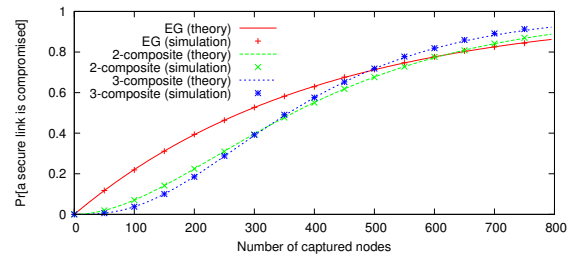**Resilience against node capture attack**

In the second set of simulation experiments, we verify the theoretical results of the effect of node capture attack. Recall that the adversary can obtain all keys stored in captured nodes' memory, allowing to compromise other secure links with a non-zero probability[1]. Assume that we want to compare the resilience against node capture attack in the EG and q-composite schemes when $q = 2$ and $q = 3$. By a similar derivation procedure of key ring size, we estimate $m = 502$ for q-composite scheme when $q = 3$ to achieve a connectivity of 99.9%. Using the same parameter settings for network topology, the simulation results are obtained and then compared with theoretical predictions in Figure 7. The results generated from our simulator are consistent with the expected link compromise probability. That is, q-composite scheme provides better resilience under small-scale node capture attacks; however, its security weakens as the number of captured nodes increases[2].

8.2 Boundary effects

To study boundary effects, we repeated the previous experiments with the same parameter settings on the unit square deployment model. To approximate the initial key ring size

---

[1]The derivation and exact formulas of link compromise probability in the EG and q-composite schemes are available in [5] and [3], respectively.

[2]Chan *et al.* explained this trade-off in [3].

Figure 7: Comparison of the resilience against node capture attack in the EG and q-composite schemes when $q = 2$ and $q = 3$

for the connectivity of 99.9% using our tool, we perform binary search for the smallest m such that the simulated WSN is connected at least 9,990 times out of 10,000 simulation runs (i.e., the connectivity is roughly 99.9%). The simulation results indicate that the EG scheme requires at least 392 keys, while the q-composite scheme requires 537 keys.

The simulation results differ significantly from the theoretical results. Much larger key ring sizes (about 50% larger) are required for a desired connectivity of 99.9%. These discrepancies can be explained by:

- the small network size (1,000) compared to the asymptotic random graph theory result, and

- the existence of nodes around the boundaries and corners (these nodes have much lower chance to establish secure links with others because they have fewer wireless neighbors).

8.3 Node degree

As mentioned earlier in the validation of network connectivity on the unit sphere, the distribution of node degree is very similar to that in a random graph. Figure 8a presents the comparison of this distribution, showing that they are very close. This is because in random graph and on sphere, virtually there is no boundary.

To investigate the influence of boundary effects on WSN connectivity, we run another set of simulation experiments to compare the node degree distribution in random graph to those for the wireless trust graphs with EG or q-composite scheme.

Recall that a random graph of 1,000 vertices is connected with 99.9% if the average node degree is 13.8; thus, the edge probability is $\hat{p} = \frac{13.8}{999}$. The probability that a vertex has a degree equal to d is $\binom{d}{n-1} \cdot \hat{p}^d \cdot (1 - \hat{p})^{n-1-d}$ where n is the number of vertices and hence equal to 1,000. This equation allows us to find the distribution of node degree in a random graph of 1,000 vertices and the edge probability equal to $\frac{13.8}{999}$.

Also, the mathematical analysis of EG and q-composite ($q = 2$) schemes says that the initial key ring size should be 249

and 394, respectively, in order to ensure the expected node degree exceeds 13.8, making the deployed networks connected 99.9% of the time. We run the simulation 10,000 times using the theoretical initial key ring size with the corresponding key establishment scheme to find the distribution of node degree.

Figure 8b compares the distribution of node degree in random graph and the simulation results with both EG and q-composite schemes on the wireless trust graph. Even though the average node degree in all three cases is approximately 13.8, the node degree distributions in the wireless trust graphs differ from that for the random graph. In particular, the wireless trust graphs have more nodes with low degree. The high number of low-degree nodes in the wireless trust graph indicates that nodes near the edges and corners are disadvantaged in terms of secure connectivity.

8.4 Network size
In the last set of simulation experiments, we want to compare the theoretical initial key ring size with the actual key ring size in a larger network when the deployment region is a unit square. Specifically, the network size is 2,000, the average density is 30, the key pool is 100,000 and the desired connectivity is 99.9%. The theoretical initial key ring size for EG scheme under this parameter set up is 287. The simulation result suggests that the initial key ring size should be at least 397 for a connectivity of 99.9%. In the 2000-node case, the difference between the theoretical and simulation results is 38.4%, which is much lower than the 57.4% error for the 1000-node case. When the network size increases, the accuracy of the (asymptotic) random graph theory result improves, but the estimation error is still noticeable even when $n = 10,000$ nodes.

In summary, the boundary effects have a significant impact on network connectivity. Although the estimation of initial key ring size is more accurate as the network size grows, it is still far from the actual value even if the network size is large. Therefore, studying the network connectivity for practical WSNs should consider boundary effects.

9. CONCLUSION
This paper has presented an extensible simulator for the study of secure WSNs. We have found the simulator useful in the study of network connectivity, and demonstrated some discrepancies from the results predicted by random graph theory. For future work, we are planning to implement adaptive node capture models and study their effect on the resilience of WSNs.
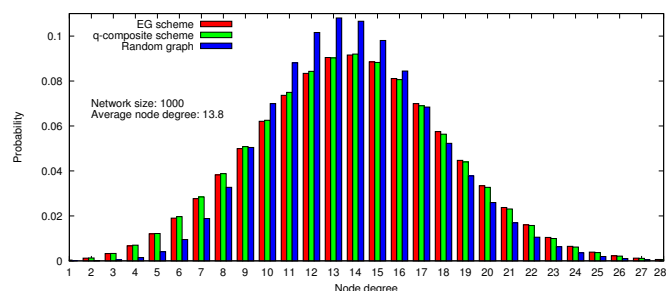
Acknowledgments

10. REFERENCES
[1] Carey Williamson's software page. http://www.cpsc.ucalgary.ca/~carey/software.html.

[2] gnuplot. http://www.gnuplot.info.
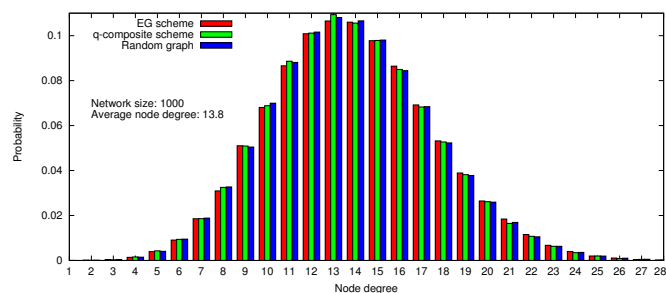
(a) Sensors are scattered uniformly at random *on a unit sphere*

(b) Sensors are scattered uniformly at random *in a unit square*

Figure 8: Comparison of the node degree distribution in random graph and in wireless trust graph with EG or q-composite key establishment

[3] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 197–213, May 2003.

[4] R. Di Pietro, L. V. Mancini, A. Mei, A. Panconesi, and J. Radhakrishnan. Redoubtable sensor networks. *ACM Trans. Inf. Syst. Secur.*, 11(3):1–22, 2008.

[5] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47, New York, NY, USA, 2002. ACM.

[6] Internet Engineering Task Force. Diffie-hellman key agreement method. http://tools.ietf.org/html/rfc2631.

[7] G. Marsaglia. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, 43:645–646, 1972.

[8] B. C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, Sep 1994.

[9] ns-2. http://www.isi.edu/nsnam/ns.

[10] J. Spencer. *The Strange Logic of Random Graphs*. Springer-Verlag, 2000.

[11] O. Yagan and A. Makowski. On the random graph induced by a random key predistribution scheme under full visibility. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 544–548, July 2008.