ON THE CONJUNCTION OF NETWORK SECURITY REQUIREMENTS AND CLUSTERING: A NEW FRAMEWORK FOR RELIABLE AND ENERGY-EFFICIENT COMMUNICATION

By

ALIREZA BOLOORCHI TABRIZI

Bachelor of Science in Computer Engineering
Amirkabir University of Technology
Tehran, Iran
2008

Master of Science in Computer Science
Oklahoma State University
Stillwater, OK
2011

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July 2014

ON THE CONJUNCTION OF NETWORK SECURITY REQUIREMENTS AND CLUSTERING: A NEW FRAMEWORK FOR RELIABLE AND ENERGY-EFFICIENT COMMUNICATION

Dissertation Approved:

Prof. M. H. Samadzadeh

Dissertation Adviser

Prof. Mike Weiser

Dr. Johnson Thomas

Dr. Tingting Chen

ACKNOWLEDGEMENTS

Though only my name appears on the cover of this dissertation, a great many people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

Foremost, I would like to express my sincere gratitude to my advisor Prof. M. H. Samadzadeh for the continuous support of my Ph.D studies and research, his patience, encouragement, enthusiasm, and immense knowledge. His guidance helped me in all the time of conducting this research, writing of this dissertation, and many hard decisions.

Besides my advisor, I would like to thank the rest of my committee:

Profs. Tingting Chen, Johnson Thomas, and Mark Weiser for their encouragement, insightful comments, and great questions.

My sincere thanks also go to Profs. Nazanin Rahnavard, K. M. George, Michel Toulouse, and Chris Crick for helping me and their guidance in many different ways during my journey.

I thank The Crosby Group for financially supporting my research and internship, in particular, I thank my friend and supervisor Mr. Matthew Atherton for his support and help.

I would like to thank Mr. Matt Meier for his wonderful advice for assisting me in deciding my career path.

I am grateful to Mr. and Mrs. Masoud and Saeedeh Vafaeean and Dr. Mahmood Shandiz for their continuous support ever since I entered the US.

Many friends helped me stay sane through these difficult years. Specially, I am grateful to my dearest friend Solmaz who eased the path for me with her unlimited moral support and made the graduate school unforgettable for me. I also like to thank Hossein whose support helped me overcome setbacks and stay focused on my graduate studies.

I finish with Iran, where the most basic source of my life energy resides. I would like to thank my parents, Nasrin and Ebrahim, and my dear brother Habib for their everlasting unconditional support in every possible way.

Name: ALIREZA BOLOORCHI TABRIZI

Date of Degree: July 2014

Title of Study: ON THE CONJUNCTION OF NETWORK SECURITY REQUIREMENTS AND CLUSTERING: A NEW FRAMEWORK FOR RELIABLE AND ENERGY-EFFICIENT COMMUNICATION

Major Field: COMPUTER SCIENCE

Abstract: Several perspectives of network security and energy efficiency were investigated and a scheme is proposed for each. A new approach is introduced to enhance communication security among nodes based on the threshold secret sharing technique and traditional symmetric key management. In the proposed scheme, key distribution is online, which means key management is conducted whenever a message needs to be communicated.

The cost and security analyses of the proposed scheme showed that its use enhances communication security among the nodes in networks that operate in hostile environments compared to related work. Another aspect of security is the storage and retrieval of data in energy-sensitive networks. The proposed scheme aims to provide an energy-efficient and secure in-network storage and retrieval protocol that could be applied to Wireless Sensor Networks. A predictive method is also proposed to adaptively instantiate the appropriate parameters for the threshold secret sharing technique. Simulations were utilized to illustrate the effect of several network parameters on energy consumption and to come up with optimal value recommendations for the parameters of the proposed secret sharing scheme. Analysis and experimentation showed that, by using the proposed technique, the confidentiality, dependability, and integrity of the sensed data are enhanced with fairly low communicational and computational overhead.

Collaborating for in-network processing is another issue (along with security) that is a concern for energy-sensitive networks. This part of the proposed framework concerns introducing a new clustering algorithm to enhance the efficiency of resource assignment for the purpose of assigning just enough components to each service-requesting application while minimizing the overall distances among the cooperating components. The proposed algorithm groups the components of a network into different-size clusters and results in a clustered network in which most of the components in a cluster, which provides service to an application, are busy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Introduction

In the cryptography literature, there are three basic sets of techniques that are used by virtually all protocols and key management schemes. The three are: symmetric shared keys, asymmetric public keys, and threshold secret sharing. Each has its advantages and disadvantages. For example, the symmetric approaches typically need a secure key distribution technique while the asymmetric ones provide this security implicitly [Diffie and Hellman 76]. Also, the symmetric algorithms are typically less computationally complex than the asymmetric algorithms [Stallings 10].

Threshold secret sharing is a widely-used technique in key management literature that was introduced in 1979 [Shamir 79]. This technique is used either by itself [Ogata and Kurosawa 96] [Ren et al. 08] or in combination with other techniques [Deng et al. 04] [Wu et al. 07]. For example, the threshold secret sharing technique has been used to add security to the shared keys (secrets) in symmetric key management techniques. This technique has also been used in enhancing the security of private keys (secrets) in asymmetric key management techniques (for more detail see Section 2.1). More specifically, the threshold secret sharing technique is used to protect vulnerable data (secret) by splitting the data into shares and distributing the shares among a number of nodes [Shamir 79] [Deng et al. 04] [Ogata and Kurosawa 96] [Wu et al. 07].

Although in the threshold secret sharing technique the data is not literally divided into shares (rather represented as a number of shares), the phrase *splitting the data into shares* is used in this work in the tradition of its usage in the relevant literature. The most important property of the threshold secret sharing technique is that a specific number of the shares are needed for reconstructing the original secret. In other words, access to fewer than that specific number of shares will not disclose any information about the original secret.

Wireless Sensor Networks (WSNs) have been extensively studied from several perspectives in recent years. The mission of WSNs is to collect data from the environment, process the data in some cases, and provide access to the data. To carry out this mission, the storage and retrieval of the sensed data has become one of the most challenging concerns in recent studies. Security of the data in storage and during communication is a broad research area.

One way to categorize network protocols is from the centralized and decentralized perspectives [Subramanian et al. 07] [Ren et al. 08]. In most of the centralized WSN schemes, the collected data is sent to a *sink* node for being stored and processed. A sink node usually has more computational and storing capabilities than the regular nodes of the network. In some cases, it might not be desirable to send the data right after it is sensed. The sensors may need to store the data in the network or do some in-network computation. The other major approach is decentralized (distributed) where no relatively more powerful sink nodes are available. In decentralized WSNs, data should be stored in the network. There are several hindrances in doing in-network storage. First of all, the nodes in WSNs do not have large memories attached to them. Moreover, the nodes are usually vulnerable to attacks and can be compromised easily. On the other hand, nodes are energy sensitive and schemes with high communication costs cannot be used in WSNs [Boukerche 08]. Many investigations have proposed novel schemes for data storage and retrieval in WSNs with security issues as the main concern of most of these studies [Subramanian et al. 07] [Eschenauer and Gligor 02] [Garay et al. 02] [Ren et al. 08] .

In Wireless Sensor Networks (WSNs), resource management and energy efficiency are important issues in the context of node interactions [Heinzelman et al. 00] [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07] [Zhang et al. 06].

It is necessary to assign sensors to the services that WSNs provide for incoming queries in a way that each service acquires its required sensors in a finite amount of time. A “service” could be monitoring environmental parameters such as temperature and moisture in different sections of a forest, or intrusion detection in different sections of a bank. A sensor might need to interact with a number of other sensors of the network to obtain the results of their operations to serve a specific query. Communication might be also between the sensors and a center that manages the sensors or functions as a sink to which the processed, collected, or generated data are sent for further processing.

1.2 Purpose of Study

This dissertation work introduces a framework of three new schemes that address several aspects of security and energy efficiency in networks. The first scheme introduces an online key management mechanism that generates keys right before using them, thus the main keys are not predistributed. This scheme uses a tailored and hybrid sharing approach which is shown to enhance security compared with related studies from several security points of view. The strength of this proposed scheme is in distributing the symmetric keys that are used to encrypt messages with enhanced security in a light-weight manner (i.e., less computationally complex and with less communication overhead).

In the second proposed scheme in this framework, nodes are grouped into a number of clusters and, at certain time intervals, the nodes send the data they have gathered to their Cluster Heads (CH) which are the nodes at the geographical centers of clusters [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07]. The CHs then concatenate the data which they could in turn split using the threshold secret sharing technique. Subsequently, the CHs disseminate the splits throughout the respective clusters. The user would send a query as a request to access the data of a geographical location and piggyback its public key to the corresponding CH. The CH gathers the data from the cluster's nodes and send back the data encrypted with the public key of the user. The user then would be able to decrypt the data with its private key.

In the third proposed scheme, a clustering algorithm is provided to group the sensors of a network into clusters of different sizes. The proposed clustering algorithm decreases the sum of the distances between the nodes and the CH in each cluster. To this end, an iterative approach is used to find the best CHs on the basis of their locations. To test the proposed clustering algorithm, a list that represents clusters with different sizes is assumed as input to the clustering algorithm. The input list shows how many clusters with each size exist in the clustered network. In this work, the input list is generated pseudo-randomly. In future studies, the list could be generated using historical data or based on the jobs' requirements.

In summary, this dissertation work provides a new framework considering the security of communication, the security of the storage and retrieval of the sensed data in WSN, and the energy efficiency of communications and collaborative processing in energy-sensitive networks.

CHAPTER II

BACKGROUND AND LITERATURE REVIEW

2.1 Threshold Secret Sharing Technique

Shamir introduced the threshold secret sharing technique that has two parameters m and n [Shamir 79]. The threshold secret sharing technique splits a secret into n shares in a way that with fewer than m shares, $m \leq n$, no information would be disclosed about the secret. The secret could be reconstructed with m or more shares. In this technique, a random polynomial of degree *m minus one* is used to split the secret into n shares. This polynomial can be represented as $S(x) = a_0 + a_1x + \cdots + a_{m-1}x^{m-1}$. In this formula, a modular p arithmetic is used with p a large prime number greater than n and a_0. In this polynomial, a_0 is the secret and a_1 through a_{m-1} are selected randomly from a uniform distribution in the range $[0,p)$. Then, $S(1)$, $S(2)$, *…, and* $S(n)$ are evaluated as the n shares. Using interpolation techniques, the coefficients can be calculated from m or more shares of the secret [Shamir 79]. Since its introduction, the threshold secret sharing technique has been improved upon and tailored in different studies. Wu et al. provided several references for these improvements, e.g., verifying the validity of a received share and periodically updating shares [Wu et al. 07].

2.1.1 An Example

Assume a threshold secret sharing scheme with parameters *n = 4* and *m = 3*. A prime number that is larger than a_0 and n should be chosen. In this example, *p* is chosen as 7. The respective polynomial would be of degree *m-1* or *2*, i.e., $q(x) = a_0 + a_1 x + a_2 x^2$, where a_0 is the data, which is *1,* and a_1 and a_2 are selected pseudo-randomly from a uniform distribution in the range *[0, 7)*. In this example, it is assumed that a_1 and a_2 are *4* and *5*, respectively. Thus the random polynomial would be $q(x) = 1 + 4x + 5x^2$. Then *S(1), S(2), ..., and S(4)* could be evaluated as the *4* shares of the secret.

S(1) = 1 + 4 * 1 + 5 * 1^2 = 10 mod 7 = 3

S(2) = 1 + 4 * 2 + 5 * 2^2 = 29 mod 7 = 1

S(3) = 1 + 4 * 3 + 5 * 3^2 = 58 mod 7 = 2

S(4) = 1 + 4 * 4 + 5 * 4^2 = 97 mod 7 = 6

Therefore, *3, 1, 2*, and *6* are the shares that are distributed in the network. By obtaining any 3 of these 4 shares, the polynomial can be reconstructed. For example, a_0, a_1, and a_2 can be calculated from the following equations:

S(1) = $a_0 + a_1 1 + a_2 1^2\ mod$ 7 = 3

S(2) = $a_0 + a_1 2 + a_2 2^2\ mod$ 7 = 1

S(3) = $a_0 + a_1 3 + a_2 3^2\ mod$ 7 = 2

Likewise, a_0, a_1, and a_2 can be calculated using the following equations:

S(1) = $a_0 + a_1 * 1 + a_2 * 1^2\ mod$ 7 = 3

S(3) = $a_0 + a_1 * 3 + a_2 * 3^2 \ mod\ 7$ = 2

S(4) = $a_0 + a_1 * 4 + a_2 * 4^2 \ mod\ 7$ = 6

Using any three of the four possible equations one could regenerate the coefficients of the random polynomial, i.e., $q(x) = 1 + 4x + 5x^2$, and consequently the secret a_0.

2.2 Eschenauer and Gligor (EG) Scheme

The EG scheme is a well-known and highly referenced symmetric based key management technique, and several subsequent studies have improved its security and efficiency [Chan et al. 03] [Chan et al. 04] [Du et al. 04] [Du et al. 07] [Ito et al. 05]. The EG scheme is probabilistic in that there is a key pool and nodes draw keys from that pool which are subsequently put in their respective key rings (sets of keys that nodes draw from the pool) [Eschenauer and Gligor 02].

If each node draws *k* keys from a key pool without replacement (i.e., if the keys are distinct) and the number of keys in the pool is *x*, then the probability that two nodes have shared keys would be as follows [Eschenauer and Gligor 02].

Pr = 1- C(x-k, k) /C(x, k) = $1 - \frac{(x-k)!}{k!(x-2k)!} / \frac{x!}{k!(x-k)!} = 1 - \frac{((x-k)!)^2}{x!(x-2k)!}$

Since x is typically large, Stirling's approximation can be used for n!:

$$n! \approx \sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n}$$

Using the approximation, the result would be:

$$Pr = 1 - \frac{(1 - \frac{k}{x})^{2(x-k+\frac{1}{2})}}{(1 - \frac{2k}{x})^{(x-2k+\frac{1}{2})}}$$

The q-composite random key pre-distribution scheme, which is an extension on the EG technique, was introduced by Chan et al. [Chan et al. 03]. In this scheme, two nodes can communicate only if they share q keys, q>1. Their work requires more keys in the key ring than the EG scheme but provides better resilience against nodes being compromised as long as the number of compromised nodes is relatively small. For example, for a key ring of size 200 and q = 2, q-composite outperforms EG for less than 90 compromised nodes. Qian [Qian 12] proposed a new scheme for key pre-distribution and computed the effect of capturing and accessing the stored data in a number of nodes on the number of compromised messages. Du et al. [Du et al. 07] provided a new key management scheme that supports two-tier heterogeneous sensor networks using a clustering approach. The authors proposed using the EG scheme [Eschenauer and Gligor 02] in heterogeneous networks where the nodes in one tier draw M keys from the pool and the nodes in the other tier draw L keys from the pool. They compared their results with the EG and q-composite schemes [Chan et al. 03]. It is claimed that their new scheme outperforms both the EG scheme and the q-composite scheme based on the number of messages that could be decrypted by adversaries where the same number of nodes are compromised in both schemes [Du et al. 07].

Du et al. [Du et al. 04] introduced an EG-based scheme based on the availability of knowledge about the deployment point prior to the deployment of the network. This study was extended by Ito et al. [Ito et al. 05] by removing the deployment knowledge requirement. Like Ito et al.’s scheme [Ito et al. 05], the proposed scheme in this work will not assume any deployment constraints.

2.3 Secure Communication

Symmetric shared keys, asymmetric public keys, and threshold secret sharing are salient examples, from among a number of different techniques, that have been introduced in the cryptography literature to tackle the problem of security in network communication [Diffie and Hellman 76] [Shamir 79] [Lu et al. 12]. Alongside the symmetric and asymmetric key management techniques, many proposed schemes have used the notion of threshold secret sharing technique for the purpose of adding security to symmetric schemes or in order to reduce the computational overhead of the asymmetric schemes [Deng et al. 04] [Wu et al. 07] [Kumar et al. 12] [Xiong and Gong 11]. The threshold secret sharing technique has also been used by itself (as opposed to being used in combination with other techniques) to enhance data security in a number of studies (e.g., [Ogata and Kurosawa 96] and [Ren et al. 08]).

Researchers have used the threshold secret sharing technique, along with other security methods, to enhance the efficiency and security of routing protocols, storage and retrieval methods, etc. [Gentry 03] [Gupta and Shrivastava 13] [Kumar et al. 12] [Lu et al. 09] [Ruan et al. 12] [Ruan et al. 11] [Shamir 84] [Wu et al. 07] [Zhou and Haas 99]. Zhou and Haas [Zhou and Haas 99] introduced a scheme to enhance the security of ad hoc networks. They used public/private key pairs for communication among nodes. Subsets of nodes in the network can be considered as trusted entities and referred to as *certificate authorities*. Certificate authorities guarantee the binding of the public key and the identity of the owner, hence they are responsible for network-wide security. Zhou and Hass believed that a single node could not be trusted as a certificate authority. They proposed using the threshold secret sharing technique to break the private key of the certificate authorities into shares and distributing them across multiple paths. These nodes collaborate to verify the binding of the public keys and the nodes. This verification is accomplished by using a digital signature. For a successful attack to the certificate authorities, an attacker needs to gain access to multiple nodes.

In an analogous study, Wu et al. [Wu et al. 07] proposed a framework for Secure and Efficient Key Management (SEKM) in mobile ad hoc networks. In their certificate based scheme, the threshold secret sharing technique is used to divide the private key of the certificate authority. Next the shares of the private key (secret) are distributed to a subset of the nodes in the network called server nodes. In SEKM, to make the secret’s shares update and the certificate update services efficient, the servers form a multicast server group. The authors also proposed a secure method to form and maintain the server group.

In a more recent study [Lu et al. 09], the threshold secret sharing technique was used to improve Certificate-Based Encryption (CBE) [Gentry 03]. CBE is an asymmetric approach that uses Identity-Based Encryption (IBE) [Shamir 84] in public key encryption. Lu et al. [Lu et al. 09] utilized the threshold secret sharing technique to split the system master key (secret) into shares and distribute the shares among the nodes, which in CBE are stored in a single location. In a somewhat similar work, Deng et al. [Deng et al. 04] used the threshold secret sharing technique and identity-based key management for authentication in Wireless Sensor Networks where entities have less computational and communicational capabilities compared to other types of networks.

Ren et al. [Ren et al. 08] proposed a new scheme called HybridS to securely store and retrieve data in a Wireless Sensor Network in a distributed manner. In their scheme, they used the threshold secret sharing technique and Reed-Solomon coding [Reed and Solomon 60] to split keys and data, respectively. Each node encrypts its sensed data with a key and then splits the key (secret) into shares by using the threshold secret sharing technique. It also uses the Reed Solomon coding technique to split the data shares. The scheme then attaches the secret key’s shares to the encrypted data’s shares and distributes the combined shares to its neighbors. The authors claimed that the HybridS scheme provides both confidentiality (only an authenticated entity can access the data) and dependability (resilience and fault tolerance).

Teo and Tan [Teo and Tan 05] proposed a hierarchical group key agreement scheme for wireless ad hoc networks based on the symmetric approach. They showed that their scheme is more cost effective than three other studies. Kumar et al. [Kumar et al. 12] extended Teo and Tan’s scheme using several modifications including applying the threshold secret sharing technique. Their scheme is more cost effective than Teo and Tan’s as they claim. Meng and Li [Meng and Li 12] proposed a key management scheme and used the threshold secret sharing technique to dynamically add, modify, or remove a node. In their scheme, based on the scale of the network, the threshold value could change. Changing the threshold value based on network parameters is a technique that is used in the scheme proposed in this work as well.

In all of the above-mentioned schemes, the security of transmitting the shares after splitting the secret is generally overlooked. While it is true that access to fewer than a specific number of shares does not disclose any information about the secret, making the procedure of distributing the shares more secure is desirable when transmitting critical data. One method for providing this added security is making the obtaining of the *m* or more shares harder for adversaries so that they cannot reconstruct the secret easily. In the first proposed scheme in this dissertation (Chapter III), a fully distributed secure scheme is provided with the goal of improving the shares' confidentiality.

2.4 Collaborative and Parallel Processing in WSNs

It is known that energy efficiency is one of the primary concerns in WSNs. While research works have been reported in the design of energy-concerned network protocols and data processing algorithms, there is still a need for energy-aware designs for collaborative processing among sensors [Yu and Prasanna 05]. Yu and Prasanna stated that, perior to their work, resource management in WSN designs had not been systematic and could result in inefficient performance of systems. They worked on energy-aware collaborative algorithms for WSNs and proposed an energy-balanced task allocation scheme for single-hop clusters in WSNs. In their proposed method, they assumed that each sensor is equipped with different voltage levels on a discrete scale which can control the speed of the processors in the sensors. While their results show that their proposed method increases the lifetime of the network, adjusting the sensors' voltages individually for each application will likely introduce too much overhead resulting in delays. As an alternative approach, the voltages of all sensors in each cluster can be set at the same time using the proposed clustering algorithm in this dissertation work. The reason is that only a single cluster is assigned to each application during the application's lifetime.

Tian and Ekici [Tian and Ekici 07] were first in proposing a task mapping algorithm for collaborative in-network processing in multi-hop WSNs which is preferred over single-hop clustered networks in large scale WSNs. They considered clusters where the longest path (in terms of hop counts) connecting any two nodes is a constant number. While they did not mention anything about the sizes of the clusters, their assumption essentially translates to having approximately equal size clusters. In their work, one application was assigned to each cluster, which with roughly same size clusters implies that it has not been considered that different applications may have different degrees of parallelism. Having same size clusters might result in a number of idle processors in one cluster while another cluster might need more processors to complete its assigned application in a smaller amount of time.

Several studies have been reported in the literature that focus on task allocation algorithms for parallel execution in WSNs. Chen et al. [Chen et al. 11] used the Particle Swarm optimization algorithm to optimize the allocation of tasks to sensors. Their fitness function considers task deadlines, total energy consumption for processing the tasks, and the total remaining energy level of all the nodes in the network. Edalat et al. [Edalat et al. 09] proposed a price-based scheme for optimizing task allocation. They defined price as a function of the energy balance and the delay constraint. Xiao et al. [Xiao et al. 09] proposed an algorithm based on the P-MinMin search algorithm for their task allocation. The proposed clustering algorithm in the third scheme of this dissertation work follows the same goal as Xiao et al. but from a different perspective (Chapter V). Instead of executing the optimization algorithm on all applications, the scheme proposed here provides an infrastructure which allows assigning applications to sensors immediately. The only required information about an application is its parallelism characteristics (in the form of a Directed Acyclic Graph representing the application's tasks dependencies). With knowledge about the parallelism characteristics of an application and using it to request a commensurate number of nodes, the proposed algorithm provides for more efficient task scheduling since each application is simply assigned to a cluster which provides, on average, a sufficient number of nodes so that the constituent tasks of the application can run in parallel. While space complexity implications should indeed be taken into account, the time complexity can approach O(1) provided that a hash table could be used.

2.4.1 Definition: Average Degree of Concurrency

It is assumed that each application is a query to the network and consists of a number of tasks some of which could be executed concurrently and others sequentially. Degree of Concurrency (DoC) and Average Degree of Concurrency (ADoC) are two commonly used concepts in parallel programming and task-dependency graphs [van Ham 05].

Task-dependency graphs are Directed Acyclic Graphs (DAGs) in which tasks comprising an application are represented by nodes and dependencies among nodes are shown by arrows between tasks [Grama 03].

In this dissertation work, Degree of Concurrency (DoC) is defined as the number of tasks that can be executed concurrently at a layer of an application's task-dependency graph. Average Degree of Concurrency (ADoC) is defined as the average of the DoCs of all layers of the task graph, i.e., the sum of the DoCs at all layers of an application's task-dependency graph divided by the height of the graph

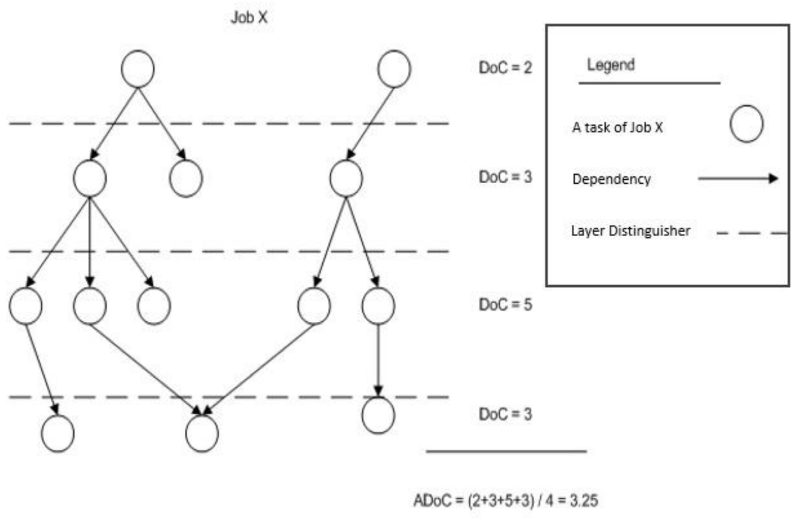


Figure 1. Example of calculating the ADoC of an application

(see Figure 1). Task-dependency graphs of applications are assumed to have been obtained using Sugiyama et al.'s method [Sugiyama et al. 81] [van Ham 05]. Their method includes the longest path layering in which each node is assigned to a layer at the depth equal to the longest path from a source to that node [Sugiyama et al. 81]. Several layering algorithms exist in the graph theory literature [Eades and Sugiyama 91].

Several methods are proposed in the literature for random DAG generation that can be used in simulation studies in the absence of DAGs from actual applications. Cordeiro et al. [Cordeiro et al. 10] compared a number of these methods, namely the Erdos-Renyi methods [Erdos and Renyi 59] (based on random graph theory), the Layer-by-Layer method [Tobita and Kasahara 02], and the Fan-in/Fan-out methods [Dick 98]. Several archives of graphs of existing applications as well as some tools for generating random graphs are publically available (e.g., [Feitelson 09], [Cordeiro 10], and [Dick 98]). Feitelson provides static logs for several graphs and parallel workloads. GGEN [Cordeiro 10] and TGFF [Dick 98] are two well-referenced tools for generating random graphs.

2.5 Clustering Algorithms

Backer and Jain [Backer and Jain 81] defined clustering analysis as follows: "a group of objects is split into a number of more or less homogeneous subgroups on the basis of an often subjectively chosen measure of similarity such that the similarity among the objects within a subgroup is larger than the similarity among the objects belonging to different subgroups".

The idea of clustering has been used in a number of studies on WSNs to enhance energy efficiency [Heinzelman et al. 00] [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07]. One of the frequently-considered constraints in designing new clustering algorithms is that a clustering algorithm for WSNs should provide energy efficiency while using the maximum capability of the nodes in the respective clusters to give the best possible performance.

In addition to the conventional sensing roles in WSNs, some nodes serve another role as well. These nodes, one per cluster, are referred to as Cluster Heads (CH). CHs sometimes have management responsibilities such as assigning less energy consuming tasks to nodes with less energy, dealing with node failures, and controlling intra-cluster communication and traffic [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07]. The responsibilities for the CHs that are assumed in LEACH [Heinzelman et al. 00] constitutes in the third part of this dissertation work.

Distances between nodes and their CHs is an important factor in the context of Wireless Sensor Networks where energy constraints limit communication capabilities [Abbasi and Younis 07] [Heinzelman et al. 00]. Several existing clustering algorithms such as Kmeans [McQueen 66] create clusters based on the distance between pairs of nodes in a network. In the Kmeans algorithm, which aims to divide a WSN into *k* clusters, *k* sensors are chosen randomly as CHs. Sensors join the clusters with the geographically nearest CH to them. Once all nodes have joined the clusters, the first step is completed. Then the nodes nearest the geographical center of each cluster are chosen as the new CHs. The procedure is repeated until the CHs do not change anymore. This procedure is convergent and finite by assuming a threshold value for the magnitude of the changes in the CHs' location less than which the change could be considered negligible [McQueen 66]. In many energy-sensitive networks [Younis and Fahmy 04] [Heinzelman et al. 00] [Bandyopadhyay and Coyle 03], Kmeans-like approaches are used as the basic clustering method and several features are typically added on top of that to provide a more efficient infrastructure for the network. Such features include changing CHs over time [Heinzelman et al. 00] and using non-random methods for choosing CHs [Younis and Fahmy 04].

One of the popular clustering algorithms for Wireless Sensor Networks is Low Energy Adaptive Clustering Hierarchy (LEACH) [Heinzelman et al. 00]. In LEACH, all sensors in the network are homogeneous and energy constrained. They assume a radio model for the transmitters and receivers. In this radio model, the receivers' energy consumption depends on the message size, and the transmitters' energy consumption depends on the message size and the square of the distance the data are being transmitting. In LEACH, it is also assumed that the radio channels are symmetric in the sense that, for a given Signal to Noise Ratio (SNR), the energy required to transmit a message from node A to node B is same as the energy required to transmit a message from node B to node A. It is also assumed that all sensors sense at the same rate, so they always have data to send to the end users.

In LEACH, sensors become CHs based on two parameters. The first parameter is a suggested percentage for the number of CHs in the network that is given as an input to the algorithm. The second parameter is the number of times a node can be a CH. CHs advertise or broadcast their status as CHs in the network. The strengths of the signals that are used for communication in the network decrease as the signals move away from the source. Based on the strengths of the signals that the sensors receive from the CHs, the non-CH nodes join a cluster with the CH that can be reached with the least energy consumption for communication [Heinzelman et al. 00].

In LEACH, CHs are changed in a timely manner to prevent them from running out of energy, which could come about much sooner than non-CH nodes. This could be considered the distinguishing feature of LEACH as a conventional clustering algorithm that has fixed CHs during its lifetime (i.e., the time period until the first sensor dies as a result of energy depletion). The LEACH protocol has been compared to three other protocols, namely direct transmission, minimum transmission control, and static clustering. In the direct transmission protocol, each sensor node transmits directly to the sink, which is efficient when there is a small coverage area and/or a high receive cost. Traffic is routed through independent nodes in minimum transmission energy protocol which is a good solution when the average transmission distance is large. In static clustering, the nodes in each cluster transmit the collected data to the CH and the CH transmits it to the sink.

It has been shown that, compared to conventional clustering algorithms, Heinzelman et al.'s algorithm increases network lifetime [Heinzelman et al. 00].

Another clustering algorithm for Wireless Sensor Networks is Hybrid Energy-Efficient Distributed clustering (HEED) [Younis and Fahmy 04]. Younis and Fahmy assumed that the sensors are stationary and all have the same amount of energy and identical processing capabilities. Like LEACH, the aim of the HEED protocol is prolonging network lifetime by adaptively changing the CHs based on their energy. In their protocol, Younis and Fahmy used a second parameter for decision making in the situations when a node receives advertisements from more than one CH. HEED works based on the probability of two CHs being in each other's transmission range, i.e., the probability of existence of nodes that might receive CH advertisement from both CHs. The smaller this probability is, the more uniformly distributed the CHs are going to be in the network, as Younis and Fahmy demonstrated. They modified LEACH slightly in order to be able to compare its result in terms of network lifetime with their own algorithm's results. They showed that their algorithm outperformed this extension of LEACH. Younis and Fahmy showed that the improvement was a result of a better choice of CHs by HEED as compared to LEACH where CHs are initially chosen randomly.

Bandyopadhyay and Coyle [Bandyopadhyay and Coyle 03] introduced an energy-efficient clustering algorithm for WSNs in which sensors in the network join clusters based on their distances from CHs. In thier algorithm, a probabilistic approach is used to select CHs. The event of a node becoming a CH follows a binomial distribution where each node becomes a CH with probability p that is determined based on the required number of clusters in the network, which was assumed to have been provided as an input to their algorithm [Bandyopadhyay and Coyle 03].

In Bandyopadhyay and Coyle's work, advertisements are broadcast in the network with the range of no more than a specific number of k hops (with k indicating the size of the cluster). The number of hops are the number of intermediate nodes in the path from a CH to a node that receives the advertisement.

Bandyopadhyay and Coyle tried to find optimal values for parameters p and k to minimize the energy used in the network [Bandyopadhyay and Coyle 03]. They simulated their algorithm and provided a comparison with another clustering algorithm, namely Max-Min D-Cluster [Amis et al. 00]. In the Max-Min D-Cluster algorithm, networks were clustered in such a way that each node is either a CH or at most d hops away from a CH, with $d \geq 1$.

CHAPTER III

SYMMETRIC THRESHOLD MULTIPATH SCHEME

3.1 Introduction

As mentioned earlier, this dissertation concerns the design of a network security framework. In the first part of this framework, a new scheme for highly secure communication in a network is introduced. The proposed scheme is a symmetric key management technique with secure online key distribution. The strength of the proposed scheme is in the enhanced security of the distribution of the symmetric keys that are used to encrypt messages. The symmetric keys are generated for each message and, in order to provide further security, the keys/secrets are split using the threshold secret sharing technique [Shamir 79]. A multipath approach along with a pre-distributed symmetric key management scheme are utilized to enhance the security of transferring the shares of the secrets to their respective destinations. Based on the analysis that is provided in this dissertation, confidentiality (i.e., access to the confidential data being restricted only to the authenticated entities) is assured to a higher level compared to related studies.

A reasonable level of dependability (i.e., being resilient to compromise and fault tolerant) is provided by the proposed scheme. In the proposed scheme, the shares of the secrets are distributed through different paths so that reconstructing the secrets depends on the reception of the shares. The level of dependability can be determined based on the threshold secret sharing technique's parameters and the redundancy they provide for the shares. A part of this research has already been published [Boloorchi et al. 14a].

Compared to similar techniques, the proposed scheme is expected to reduce the space cost which is one of the benefits of the online nature of the proposed key management system utilized in this work.

The proposed scheme is scalable in the sense that it can be adapted for different network sizes and can also be modified to provide different levels of security. Furthermore, the scheme is lightweight from different prospective such as space usage, computational overhead, and communicational overhead, hence it should be applicable in Wireless Sensor Networks.

3.2 Proposed Scheme

In this section, a new approach is introduced to provide enhanced security for communication among nodes in a network. The proposed technique is divided into two schemes with one complementing the other. They are referred to as the basic and enhanced schemes. Figure 2 presents the layout of the proposed technique. For the basic scheme, it is assumed that each node, intending to send a message through the network to a given destination, encrypts the message with a key. Notation Th(n, m) represents applying the threshold secret sharing technique [Shamir 79] of dealing with n shares where access to m or more of these n shares, with $m \leq n$, is needed to decrypt the secret. The sender generates n shares from the key, i.e., the secret. Then the sender sends the shares to the destination on different paths. The destination, using polynomial interpolation [Shamir 79], can reconstruct the key from the m shares. Obtaining fewer than m shares does not reveal any information about the key [Shamir 79].

In the enhanced scheme proposed in this section, more security is added to the basic scheme. The EG pre-distributed shared key scheme [Eschenauer and Gligor 02] is utilized to encrypt the shares that are being transferred via a number of nodes in the paths of length two from the sender to the receiver of the shares. These intermediate nodes will be referred to as shareholders hereafter. The shareholders decrypt the shares that are passing through them, discard the sender information, and send the clean decrypted shares to the destination. As a result of discarding sender information, the links between the shareholders and the destination do not contain any information about the sender, and consequently they contain no information about the encrypted main message either.

As a clarifying note, in this work, ciphertext and encrypted main message are used interchangeably. They both refer to a message that is going directly from a sender to a destination, as opposed to partial messages that go through shareholders.

To generate different sender-shareholder shared keys for each new communication, a bi-variable function f(x, y) is used, where parameter x is a shared key between the sender and a shareholder and parameter y is a counter. A secure hash function (e.g., a suitable hash function from Secure Hash Algorithm family [NIST 08]) is utilized to enhance the integrity of data in communication, with integrity defined as detectability of unauthorized data modification.

The rest of this section is divided into two subsections. The first subsection describes the basic scheme. The basic scheme is subsequently improved upon in the second subsection where it evolves into the enhanced scheme.

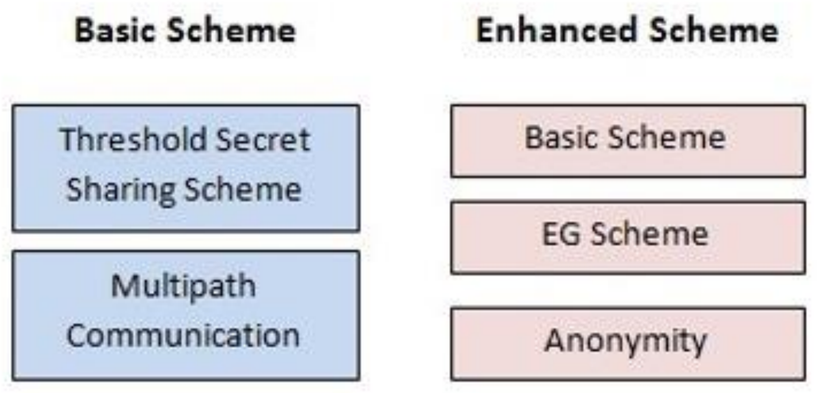


Figure 2. Layout of the two parts of the proposed technique

3.2.1 Basic Scheme

Symmetric key management techniques generally have conceptually simpler and less computationally complex algorithms compared to asymmetric techniques. So, in spite of the increased security that the asymmetric methods usually provide, symmetric methods are still used widely [Teo and Tan 05]. In this context, security refers to resistance to any type of attack and not just to cryptanalysis. A number of different protocols have utilized the symmetric approach, their common denominator being that they used the same key for both encryption and decryption. One of the most important challenges in symmetric techniques is the mechanism for distribution of the keys.

In the proposed scheme, a key is used as a symmetric key and the threshold secret sharing technique is utilized to split the key as a secret. The shares of the secret are sent through multiple paths to the destination.

As stated earlier, the advantage of using the threshold secret sharing technique is that access to fewer than m of the shares, out of a total of n shares with $m \leq n$, does not disclose any information about the secret. In this technique, at least m paths must be compromised for a breach of security. Based on the level of noise in the network, the possibility of active attacks to the links, and the probability of losing shares, the redundancy in the threshold secret sharing technique should be changed, i.e., the value of n in the threshold secret sharing technique should be increased or decreased. More redundancy enhances reliability but also increases the possibility of detection. A measure of the redundancy of this technique is the difference between n and m because access to only m of the shares is enough to reconstruct the secret.

The proposed basic scheme exposes the individual shares to being compromised since the shares are not securely transmitted. A second component ought to be added to the infrastructure of the scheme in the form of a distribution method with the goal of providing a more secure scheme. This component is described in the following subsection as a part of the enhanced scheme.

3.2.2 Enhanced Scheme

In an attempt to improve the security of the basic notion of symmetric key management introduced in the previous subsection, the enhanced scheme is introduced. The enhanced scheme is divided into two parts: Pre-Distributed Shared Keys and Symmetric Threshold Multipath Scheme.

3.2.2.1 Pre-Distributed Shared Keys

This subsection tailors the EG scheme, which was described in Section 2.1, for use in enhancing the security of links between a sender and the shareholders. In the proposed scheme, the existence of a key pool at the network controller is assumed, where a unique identifier (ID) is associated with each key. The analysis of finding an appropriate pool size for different networks is provided in the first subsection of Section 4. Before a network is deployed, each node picks (copies) k distinct keys from the pool and puts them in its key ring (k is a network parameter). After all nodes are finished with drawing their k keys, they broadcast the list of their keys' IDs. Each node then compares the received IDs with its own keys' IDs to find shared keys with the other nodes. There are a number of other more secure ways of finding shared keys among nodes, e.g., encrypting a certain challenge such as an integer, which is known by all nodes in the network, and broadcasting the encrypted challenge [Eschenauer and Gligor 02]. Any node that can decrypt the encrypted challenge has a shared key with the sender.

In the proposed scheme, the keys are pre-distributed (to avoid the security issues and the communication overhead of on-the-fly distribution of the keys) in a way that on average each node has shared keys with at least n other nodes, where n is the number of distributed shares in the threshold secret sharing technique.

The nodes in the network can be considered as the vertices of a *random graph* (the same model as the underlying model used by Eschenauer and Gligor [Eschenauer and Gligor 02]). In a random graph, denoted by G(n, p), every possible edge between two nodes occurs randomly with probability p. In the proposed network model, having at least one shared key between two nodes can be considered as a link between the two respective vertices. Each node has a degree, which is the number of nodes with which a specific node has at least one shared key [Eschenauer and Gligor 02]. For example, as a simple case of a non-random graph, assume that for nodes (a), (b), (c), and (d), the key rings are given below.

(a): k_0, k_2, k_5 *(b):* k_0, k_3, k_4 *(c):* k_2, k_3, k_6 *(d):* k_1, k_3, k_4

The degree for node *(a)* is *2* because it shares k_0 with *(b)* and k_2 with *(c)*. Likewise, the degree for nodes *(b)*, *(c)*, and *(d)* are *3*, *3*, and *2*, respectively.

Assume the event of two nodes having at least one shared key as a *success* and having no shared keys as a *failure*. The total number of successes for a node is the number of all other nodes in the network that have at least one shared key with that node. Considering that the underlying network model is a random graph, the successes and failures of any pair of nodes are independent of any other pair of nodes. For example, a node *x* having at least one shared key with a node *y* does not depend on whether or not node *x* has shared keys with other nodes. Furthermore, the probability of two nodes having at least one shared key is the same for any pair of nodes. If *Pr* is the probability that two nodes have at least one shared key in their respective key rings, the expected degree for each node will be:

$$d = Pr * (b - 1) \qquad (1)$$

where *b* is the number of nodes in the network [Eschenauer and Gligor 02].

The degree for each node can be calculated as:

$$d_i = \sum_{j=1}^{b} Pr_{ij},\ 1 \leq i \leq b$$

where Pr_{ij} is the probability of node *i* having at least one shared key with node *j* with Pr_{ii} considered to be 0.

In Equation (2) below, originally derived by Eschenauer and Gligor [Eschenauer and Gligor 02], a key pool is assumed. If each network node draws *k* distinct keys from the pool and the number of keys in the pool is x, $0 < k \leq x$, then the probability of two nodes having at least one shared key is (for more details see Section 2.2):

$$Pr = 1 - \frac{(1-\frac{k}{x})^{2(x-k+\frac{1}{2})}}{(1-\frac{2k}{x})^{(x-2k+\frac{1}{2})}} \tag{2}$$

From Equations (1) and (2), the expected degree for each node could be calculated as follows:

$$d = \left[1 - \frac{(1-\frac{k}{x})^{2(x-k+\frac{1}{2})}}{(1-\frac{2k}{x})^{(x-2k+\frac{1}{2})}}\right] * (b-1) \tag{3}$$

After distributing the keys in the network, some of the nodes might not obtain sufficient number of the shared keys to be able to connect to the required number of nodes in the network since *d* is the expected value. In the proposed scheme, the degree for each node is *n*, which is the first parameter of the threshold secret sharing technique. Suppose the degree of the nodes which have not obtained enough shared keys are $n - t_z$, where t_z is the number of missing shared keys for node z, $0 \leq z < b$ and $1 \leq t_z < n$. The nodes whose degree is less than *n* request the network controller to send them t_z missing keys. A shared key requester constructs a key, V_z, consisting of its current shared keys, $V_z = r_1 \oplus r_2 \oplus \ldots \oplus r_k$, where $\oplus$ indicates the exclusive-or operator, r_j is the *j*th key in the destination node's key ring, and *k* is the key ring size. The shared key requester encrypts the list of the IDs of the t_z missing shared keys using V_z and sends it to the network controller. The network controller is aware of all nodes' keys and is able to construct V_z to decrypt the list of t_z missing shared keys. After finding the t_z missing keys from its key pool, the network controller encrypts the t_z new keys with V_z and sends them back to the shared key requester.

3.2.2.2 Symmetric Threshold Multipath Scheme

As mentioned in the previous subsection, the classic pre-distributed symmetric key management scheme used in the proposed scheme is the EG scheme [Eschenauer and Gligor 02]. The EG scheme is deployed in a way that each node has shared keys with at least *n* other nodes in the network, where *n* is a parameter in the *Th(n, m)* threshold secret sharing technique which denotes the number of distributed shares. These *n* nodes are referred to as the neighbors hereafter.

A communication session starts every time a sender sends a message to a destination. As mentioned in Section 2.1, the sender generates a random polynomial S(x) of degree *m-1* at the beginning of each communication session where *m* is the second parameter in the threshold secret sharing scheme which indicates the minimum number of shares needed to reconstruct a secret. In the following equation for S(x), a_0 is the secret that is used as the key to encrypt the main message which is going to be sent from the sender to the destination:

$$S(x) = a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1}$$

The sender generates *n* shares as follows:

$$S_i = a_0 + a_1i + a_2i^2 + \dots + a_{m-1}i^{m-1} \bmod p \qquad (5)$$

where *n* is the first parameter in the threshold secret sharing scheme indicating the number of distributed shares for each communication session, S_i is the share between the sender and the *i*th shareholder, $1 \leq i \leq n$, and *p* is a large prime number.

The sender then sends the shares to its *n* neighbors. The shared keys are used to enhance the security of the links between the sender and the shareholders. Because of the properties of the threshold secret sharing technique, it is guaranteed that adversaries will not find any information about the secret by successfully attacking fewer than *m* shareholders. Also, with shared keys from at least *m* shareholders, the destination node can rebuild the secret and decrypt the encrypted main message [Shamir 79].

Using just the shared keys to encrypt the shares is risky because whenever adversaries access the data in a new node, they gain more shared keys. After a while, the adversaries might obtain enough shared keys to reconstruct the secret and consequently decrypt the communicated ciphertext. To deal with this problem, a bi-variable function $f(x, y)$ and two *counter table*s are pre-distributed in each node of the network. The encrypting and decrypting counter tables (tables which initially contain zeros as the starting counters for all neighbors of a node) and the counters are indexed in the counter tables in each node based on the neighbors' addresses. Whenever a sender intends to send out its shares, it uses $f(SK_i, c_i)$ to encrypt the shares. In this function, SK_i is the ith shared key and c_i is a pointer to the counter of the ith shareholder in the encrypting counter table of the sender where $1 \leq i \leq n$ with n as the first parameter of threshold secret sharing technique. The sender then sends the following message to the shareholders and increments c_i (the respective counters in its encrypting counter table) by one:

$H ((share_i || address\ of\ the\ destination)_{f(SKi,ci)})$

where H is a secure hash function used to enhance the integrity of the message, $share_i$ is the ith share, $1 \leq i \leq n$, the notation || indicates concatenation, and subscript $f(SK_i, c_i)$ refers to the key with which the share is encrypted.

Then the sender sends the following message to the destination:

$H ((Data)_K)$

where *Data* is the clean main message and $(Data)_K$ is the main message that is encrypted with the secret K.

Upon receiving a message from the sender, the shareholders use the pre-distributed $f(x, y)$ to decrypt the shares using their shared keys and their decrypting counter tables. The shareholders then send the following message to the destination.

$H (share_i)$

where $share_i$ is the ith share, $1 \leq i \leq n$.

Before sending the clean shares to the destination, each shareholder increments c_i in its decrypting counter table. Each shareholder sends a message to the sender including the amount of time it takes for the clean share message to be transferred from that shareholder to the destination. This time interval is referred to as the time distance between a sender and a destination.

The sender sends the shares' ID list to the destination with a delay equal to the maximum of the time intervals that it has received from the shareholders. The sender also sends share discard messages to the shareholders that have not yet sent their shares, asking them to discard their shares. This procedure ensures that there is no residual information about the shares of ciphertext nor its secret in the network. Furthermore, the destination has already received all the needed shares as long as the clean shares have not been modified by adversaries during their transmission from the shareholders to the destination. To deal with the case of possible modification of clean shares by adversaries, the amount of time that the sender waits before sending the shares' ID list and the discard messages can be increased in environment's with more possibility of being attacked. In such cases, since the discard messages will be received later, it is obvious that the probability of receiving redundant shares at the destination increases as well. Note that the modified clean shares could be recognized using hashing techniques. Even if adversaries obtain all the main messages and the shares, as is shown in Subsection 3.4.2, it is

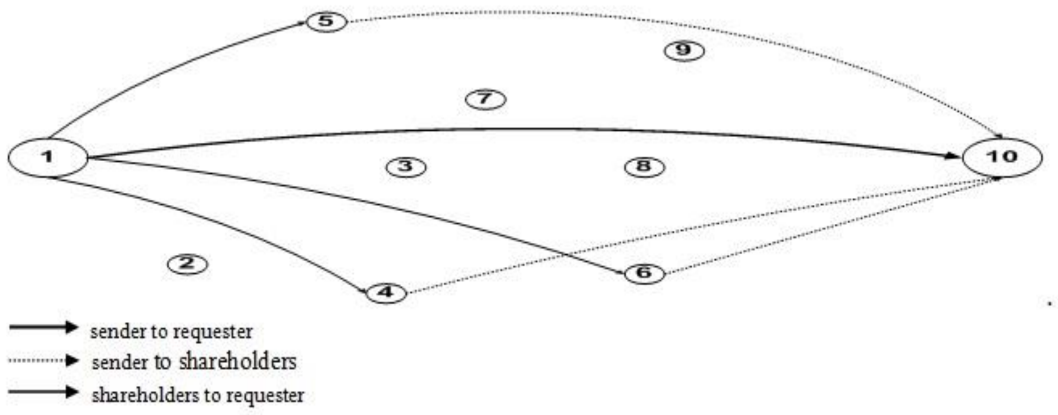


Figure 3. An illustrative example for the proposed scheme

not easy to find the exact relation of the shares and the main message since there are going to be many other messages and shares in the network.

Finally, upon receiving the ciphertext and enough shares, i.e., *m* shares, the destination can rebuild the secret, which is the key that is used to encrypt the main message and to use it to decrypt the ciphertext.

3.2.2.3 Illustrative Example

Before starting the analysis, further explain the proposed scheme, an illustrative example is provided in this subsection. Assume that there are 10 nodes in a network and a *Th(3, 2)* threshold secret sharing function is used. Figure 3 illustrates a possible non-random graph example of this network. Nodes 1 and 10 are a sender and a destination, respectively, and the other nodes are potential shareholders. The sender has shared keys with 3 nodes: 4, 5, 6 (let's call them shareholders 1, 2, 3). The pool contains 10,000 keys.

In this example, the desired degree for each node is considered to be 3, that is, each node has shared keys with at least 3 nodes in the network.

So $d = 3$, $b = 10$, and based on Equation (3):

$$3 = 9 * \left(1 - \frac{\left(1 - \frac{k}{x}\right)^{2\left(x-k+\frac{1}{2}\right)}}{\left(1 - \frac{2k}{x}\right)^{\left(x-2k+\frac{1}{2}\right)}} \right)$$

with $x = 10{,}000$, k from this equation would be around 150, i.e., each node should have approximately 150 keys in its ring to have on average a degree equal to 3.

From Equation (5), the shares are generated as follows:

$S_1 = a_0 + 1a_1 \bmod p,$

$S_2 = a_0 + 2a_1 \bmod p,$

$S_3 = a_0 + 3a_1 \bmod p$,

where a_0 is the secret K, a_1 is the coefficient of a pseudo-randomly generated polynomial of degree one, and p is a large prime number $p >> 3$.

Using function $f(SK_i, c_i)$ for the first message, the shares could be encrypted by the following shared keys:

$k_1 = f(SK_1, 0)$, $k_2 = f(SK_2, 0)$, $k_3 = f(SK_3, 0)$ where c_i is 0 because this is the first time there is a communication between the sender and the shareholders.

The shareholders can then decrypt the shares using their shared keys and the senders' counter in shareholders' decrypting counter tables.

Using two of these shares and interpolation methods, the destination node can reconstruct the secret K, and use it to decrypt the ciphertext.

3.3 Theoretical Analysis

This analysis section contains four subsections. First, the security of the proposed scheme is discussed then the space, communication, and computation costs of the scheme are analyzed.

3.3.1 Security

In the proposed scheme, adversaries might attack six component types to obtain information about the main messages: senders, destiniations, shareholders, sender-destination links, sender-shareholder links, and shareholder-destination links. To analyze the security of the proposed scheme, the six component types can be divided into two groups (nodes, links) where, based on the data they contain or transfer, different types of attacks should be considered for each group.

Component type 1 - Senders: The sender type nodes contain clean main messages and they are vulnerable to any type of attack.

Component type 2 - Destinations: The nodes of this type only contain ciphertexts before receiving all the required *m* shares, where *m* indicates the minimum number of a ciphertext's shares required to reconstruct its secret.

Component type 3 - Shareholders: Nodes of this type contain shared keys during the lifetime the network. They also contain shares from the time they finish decrypting the shares until they send out the clean shares (unencrypted shares) to the destination.

Component type 4 - Sender-Destination Links: Links of this type only transfer ciphertexts.

Component type 5 - Sender-Shareholder Links: Any link of this type transfers the encrypted shares and information about a share's corresponding sender which might disclose the relation between the share and the related ciphertext.

Component type 6 - Shareholder-Destination Links: By eavesdropping a link of this type, an attacker might acquire a clean share but no information about the share's related ciphertext.

Considering the six component types listed above and the information they contain, adversaries could follow one of the following four scenarios to obtain a clean main message.

Scenario 1: Attack a component 1 type node (a sender), and obtain a clean main message. This vulnerability is a common problem in security schemes unless a mechanism is provided to discard the sensitive data as soon as they have been sent or encrypted. In the proposed scheme, clean main messages are discarded as soon as they are encrypted. Therefore, adversaries do not have a chance to obtain a clean main message by compromising a sender, and instead they can only deal with a ciphertext after the message has been encrypted.

Scenario 2: Attack a destination node or a component 2 type node. Before the arrival of all shares related to a ciphertext, if adversaries attack a destination, they are not going to obtain any data except for the ciphertext which is useless without its corresponding secret (the main key). After the arrival of enough shares to rebuild the secret, adversaries might be able to decrypt the corresponding ciphertext. However, that does not affect any other ciphertext since each main message is encrypted with a different secret.

Scenario 3: Brute force attack after adversaries eavesdrop a link between a sender and a destination and obtain a ciphertext. In this scenario the adversaries are not likely to succeed since a secure symmetric key algorithm with a sufficiently large key is used to enhance the security of the main message [Barker and Roginsky 10]. On average, the attackers should check $2^{key\text{-}size\ -1}$ different keys, and as Barker and Roginsky recommended [Barker and Roginsky 10], *112*-bit long keys are safe considering until the computing power of processors undergoes dramatic increases. This suggests that from this aspt the proposed scheme is reasonably computationally secure.

Scenario 4: Acquiring a ciphertext and its secret's related shares, rebuilding the related secret, and decrypting the ciphertext. To obtain shares, adversaries could attack a component of type 3, 5, or 6. Different possible scenarios are discussed below.

An attack to a type 3 component (a shareholder): Using this attack, adversaries can obtain the shared keys of a shareholder and the shares arriving at that shareholder. Adversaries need to obtain at least m related shares to be able to decrypt the corresponding ciphertext. The shared keys in the network are shared among several nodes and, by attacking a node, adversaries actually also obtain several other nodes' shared keys. However, obtaining the shared keys is not enough to decrypt the other encrypted shares, since the adversaries also need the related counter for each share, which is different for each node. Therefore, even if through attacking a component of type 1, 2, or 4, adversaries do gain the encrypted shares and the shares' related ciphertext, the adversaries would need to either attack all the related shareholders or check all the available keys in the key pool to decrypt the shares. On average, adversaries can find the key for the decryption of a captured share after trying half of the keys in the pool size. This scenario can come to pass only if an adversary knows all the keys in the pool, otherwise the adversary would need to check on average half of all the keys in the key space. For example, if the key size is *112* bits, adversaries need to try 2^{111} different cases. Existence of the counters forces the adversaries to try all combinations of every possible counter and the keys in the key space in order to be able to decrypt an encrypted share. In the case where adversaries know all the keys, they would only need to try the combinations of the counters and the keys in the key pool. This is computationally infeasible since the counter range could be arbitrarily large. Note that all of the above-mentioned computations are only for obtaining a single share, and adversaries would need to obtain more than *m-1* clean shares as well as the ciphertext in order to decrypt the message.

For the following analysis, let a node's being compromised be considered a success and its not being compromised a failure, where attack to a node is independent of attacks to other nodes. And also let's consider that attacks to nodes occur with the same probability. Then the number of compromised nodes in a network by definition follows a binomial distribution.

Lemma 1: In the case of attacking shareholders to acquire shares, the probability of decrypting a main message can be computed as follows:

$$p_d = \frac{1}{p_s * b} * \frac{c(p_{cn} * b * n * p_s, m)}{c(p_s * b * n, m)}$$

where b is the total number of nodes, the probability of a node being compromised is p_{cn}, n and m are the threshold secret sharing technique's first and second parameters, and p_s, which is a network parameter, is the probability that a node sends out a message.

Proof: The expected value of the number of compromised nodes is:

$$N_{cn} = p_{cn} * b \qquad (6)$$

N_{link}, the average number of links emanating from each node, which is the average number of links to which adversaries can gain access by attacking each node, can be stated as follows:

$$N_{link} = \frac{n * p_s * b}{b} = n * p_s \tag{7}$$

where *n* is the threshold secret sharing technique's first parameter and p_s is the probability that a node sends out a message.

The probability of specific ciphertext *i* being acquired by adversaries is:

$$p_{c_i} = \frac{1}{p_s * b} \tag{8}$$

The average number of the third type of links in the network (a component of type 6) which are the links between shareholders and destinations, is:

$$N_{3rd} = p_s * b * n \tag{9}$$

The probability of all the shares related to a specific ciphertext *i* being acquired by adversaries can be calculated as follows:

$$p_{sh} = \frac{c(N_{cn} * N_{link},\ m)}{c(N_{3rd},\ m)} = \frac{c(p_{cn} * b * n * p_s,\ m)}{c(p_s * b * n,\ m)} \tag{10}$$

where *m* is the second parameter in the threshold secret sharing technique.

The probability of an encrypted main message being decrypted by adversaries can be computed as follows:

$$p_d = p_{c_i} * \ p_{sh} = \frac{1}{p_s * b} * \ \frac{c(p_{cn} * b * n * p_s,\ m)}{c(p_s * b * n,\ m)} \tag{11}$$

In Subsection 3.4.2.1, the probability of decrypting a ciphertext is analyzed based on Lemma 1.

An attack to a component of type 5 (sender-shareholder links): Using this attack, adversaries can obtain an encrypted share from each link they attack. As it was mentioned for Scenario 3 earlier, significant computational overhead is incurred by adversaries who intend to decrypt an encrypted message without having its key. This will likely lead attacks to failure. Based on the same reasoning as in Scenario 3, a link in this type of attack is computationally secure since the computational overhead makes it infeasible for adversaries to decrypt the encrypted messages they obtain [Stallings 10].

An attack to a component of type 6 (shareholder-destination links): This attack is admittedly to the weak point of the proposed scheme. If an adversary knows a ciphertext and the ciphertext's related clean shares, only the corresponding communication session will be in harms way. The reason is that a new main key is generated for each session and the information in each session is useful only for that specific session. In addition, in the proposed scheme, shareholders discard all information about the senders as soon as they send the shares to the respective destinations. Therefore, adversaries cannot find any correspondences among the shares and the ciphertexts that they might obtain. The more shares that the adversaries obtain, the harder it would be to find the related ciphertexts. This is because adversaries would need to check all combinations of any *m* of all the shares they obtain, where *m* is the minimum number of shares needed to reconstruct the secret in order to decrypt a ciphertext. To check a combination, adversaries would need to utilize the interpolation methods with complexity $O(n^2 \log n)$ with *n* being the first parameter in the threshold secret sharing scheme [Shamir 84]. Note that the list of the related shares' IDs for each ciphertext is received at the destination once the destination node has obtained all the shares. Therefore, potential adversaries are denied the opportunity to acquire any useful information about the correspondence of the shares and the ciphertexts by eavesdropping the links between the shareholders and a sender.

For this scenario, similar to the case of the number of compromised nodes in an attack to a component of type 3, the number of links that are eavesdropped by definition follows a binomial distribution where a link's being eavesdropped is considered as success and its not being eavesdropped as failure.

Lemma 2: If adversaries attack the shareholder-destination links, the probability of decrypting a ciphertext would be:

$$p_d = \frac{1}{p_s * b} * \frac{c(P_{cl} * p_s * b * n\ ,\ m)}{c(p_s * b * n,\ m)}$$

where the total number of nodes in the network is b, P_{cl} is the probability of a link being eavesdropped (P_{cl} is the same for all links and does not depend on whether or not any other link is being eavesdropped), n and m are the threshold secret sharing technique's first and second parameters, and p_s is a network parameter which is the probability that a node sends out a message.

Proof: Let N_{3rd} be the number of third type links in the network. The average number of eavesdropped links can be represented as:

$$N_e = P_{cl} * N_{3rd} \tag{12}$$

Based on Equation (9), $N_{3rd}= p_s * b * n$ where p_s is the probability that a node sends out a message, *b* is the total number of nodes in the network, and *n* is the first threshold parameter.

The probability that adversaries obtain all the shares related to a specific ciphertext *i* can be expressed as:

$$p_{sh} = \frac{c(N_e,\ m)}{c(N_{3rd},\ m)} = \frac{c(P_{cl}*p_s*b*n,\ m)}{c(p_s*b*n,\ m)} \tag{13}$$

where *m* is the second parameter in the threshold secret sharing technique.

Thus, the probability of decrypting a ciphertext could be calculated as follows:

$$p_d = p_{c_i} * \ p_{sh} = \frac{1}{p_s*b} * \frac{c(P_{cl}*p_s*b*n,\ m)}{c(p_s*b*n,\ m)} \tag{14}$$

where p_{c_i} is the probability that a specific ciphertext i is compromised by advarsaries (Equation (8)).

Based on this lemma, the probability of decrypting a ciphertext is analyzed in Subsection 3.4.2.2.

3.3.2 Space Cost

In the proposed scheme, *n* keys need to be stored in each node with *n* being the first parameter of the threshold secret sharing technique. If *112 bit* keys are used, as strongly recommended for the federal government as of in 2011 by US National Institute of Standards and Technology [Barker and Roginsky 10], the space needed for storing keys would be *n * 112 bits*. For example, based on Equation (3), to have the expected degree of *500* for the nodes in a network with *10,000* nodes and a pool size of *100,000*, approximately *200* keys are needed to be stored in each node, thus requiring about *23KB* of memory space.

In the proposed scheme, while the shareholders are decrypting the shares, they temporarily hold the shares. The shares are discarded after being sent to the destination node.

There are some other sources of relatively negligible space overhead in the proposed system such as the bi-variable function *f(x, y)* and the secure hash function. Overall, the protocol does not introduce a significant amount of space overhead, and what it requires is not even considerable for space sensitive wireless sensors such as imote2 which has 32MB of external memory space [Crossbow 07]. For many of the previous EG-like schemes (e.g., [Chan et al. 03], [Du et al. 04], or [Ito et al. 05]), connectivity of the network introduces a constraint on the number of keys stored in each node. For example, Chan et al. [Chan et al. 03] suggested preloading $2(\sqrt{n} - 1)$ keys in each node with *n* being the network size (about 64 keys in a network with 1000 nodes). In all these schemes, the number of keys in each node is proportional to the network size. In the proposed enhanced scheme in this chapter, the number of keys stored in each node is related to the required security and is equal to the first parameter in the threshold secret sharing technique which is not directly related to the network size. Based on published calculation the [Ibriq and Mahgoub 12], in a network with 10,000 nodes, the needed number of keys for each node is about 250 [Eschenauer and Gligor 02], 70 Ito [Ito et al. 05], 121 [Ibriq and Mahgoub 12], and 99 [Chan et al. 03]. For the proposed scheme, it is shown that fewer than 20 keys in each node provides for a fairly secure scheme. The number of keys needed to be stored in each node will affect the space requirement which is an important issue in WSNs.

3.3.3 Communication Cost

Three of the six component types (as defined in Subsection 3.3.1) are involved in a communication session where a new message, i.e., a main message, is transmitted from a sender to a destination. The sender sends the encrypted main message and the list of the shares' IDs to the destination. The amount of time it takes for the shares to be sent from each shareholder to the destination, i.e., the time distance between these nodes, is sent to the sender. The shareholders also send the clean shares to the destination. The summary of the communicated data follows:

Senders to destinations: encrypted message and list of the shares' IDs

Senders to shareholders: encrypted shares

Shareholders to senders: The time distances between shareholders and their respective destinations

Shareholders to destinations: shares

The communicational overhead is approximately the same as the overhead of a simple multipath communication method since a number of messages that are not particularly large (i.e., as large as the secret key size which is *112* bits), are sent through multiple paths.

3.3.4 Computation Cost

The overall computational overhead in the proposed scheme can be broken down as follows:

Network controller: Managing the pre-distributed shared key scheme (offline hence O(1))

Senders: Decrypting shares *(O(n))*

Shareholders: Creating shares *(O(n))*

Destinations: Reconstructing secrets using interpolation methods *$(O(n\log^2 n))$* [Shamir 79]

where *n* is the first parameter in the threshold secret sharing technique.

The computational overhead of the protocol introduced in this chapter is mostly for the destination node which needs to reconstruct the polynomial using interpolation methods. Polynomial interpolation methods' overhead is not high since "even straightforward quadratic algorithms for polynomial interpolations are fast enough for practical key management algorithms" [Shamir 79].

3.4 Numerical Analysis

3.4.1 Pre-distributed Shared Keys

This subsection provides a numerical analysis for the pre-distributed shared key part of the proposed scheme. Figure 4 depicts the number of keys that each node should draw from a key pool to satisfy a certain expected degree for the nodes in the network, with degree being the expected number of nodes

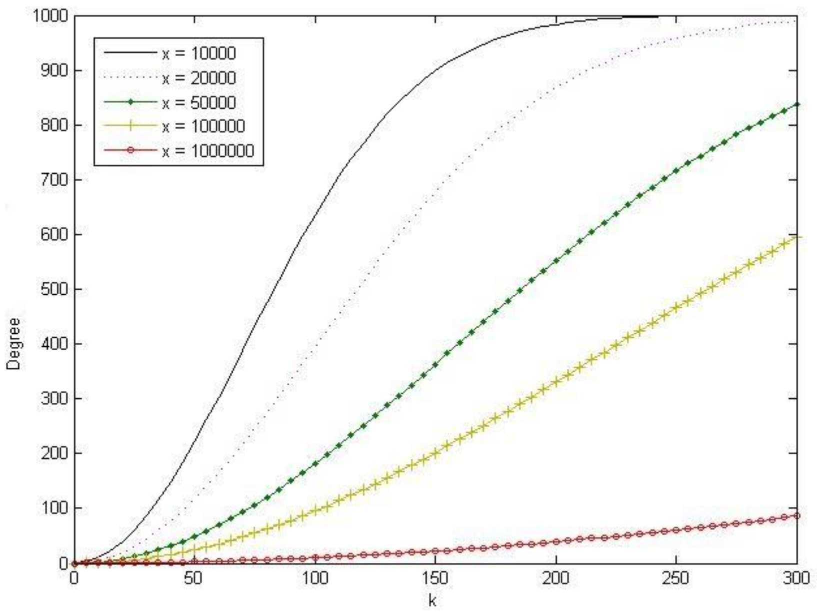


Figure 4. Expected degrees for 10,000 nodes in the network for different pool sizes based on key ring size (k) for different pool sizes (x)

with which a node shares at least one key. In this figure, several pool sizes are assumed, in a *10,000*-node network. As Figure 4 shows, as the pool size decreases, the expected degree for a given key ring size increases. Figure 5 shows the effect of the growth of the network size on the expected degree for different key ring sizes. These two diagrams indicate that the proposed system is scalable, and a targeted expected degree can be obtained by choosing an appropriate pool size for different network sizes. However, the required expected degree would depend on parameter *n*, where *n* in threshold secret sharing technique, *Th(n, m)*, defines the level of redundancy which, for the sake of security, should not be much larger than *m*. On the other hand, *m* should not be very large because of the computational overhead incurred in the secrets' reconstruction phase. Accordingly, it is not necessary that the actual required expected degree be very large. The key ring's size varies between 0 and 300 in Figures 4 and 5 which is meant to be comparatively demonstrative based on similar studies [Eschenauer and Gligor 02].

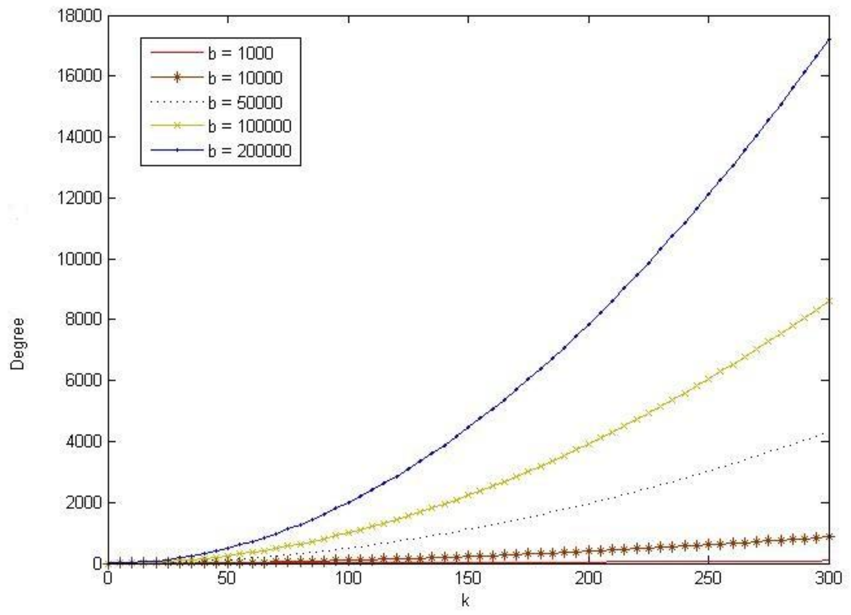


Figure 5. Expected degrees for different number of nodes and a pool with 1,000,000 keys based on key ring size (k) for different network sizes (b)

3.4.2 Symmetric-Threshold Technique

In the following two subsections, attacks to two of the component types that are described in Subsection 3.3.1 are probabilistically analysed.

3.4.2.1 Attack to components 3 (shareholders)

Based on Lemma 1, the probability of a main message being decrypted by adversaries can be computed as follows:

$$p_d = \frac{1}{p_s * b} * \frac{c(p_{cn} * b * n * p_s,\ m)}{c(p_s * b * n,\ m)}$$

For varying threshold secret sharing parameters values, Figure 6 shows the probability of a single main message being compromised in a network with 500 nodes and the probability of a single node being compromised being 0.5 (Figure 7 is comparing other probabilities). As Figure 6 indicates, the probability of a node being compromised ranges between 10^{-16} and 10^{-9}, in the case where adversaries do not know the correspondences between ciphertexts and their shares. Note that the curves are strictly descending and, with larger values for threshold parameter *n*, security will increase, therefore threshold parameter values less than 20 sufficiently demonstrate the security of the proposed scheme in Figure 6 (the same situation holds for Figure 9 in Subsection 3.4.2.2). Even if adversaries do obtain information about the relation between a ciphertext and its related shares, i.e., $p_{c_i} = 1$, then, according to Figure 6, the probability of a node being compromised still ranges between 10^{-10} and 10^{-5}. A comparison of the probability of adversaries acquiring a clean main message for different number of nodes in the network and different probabilities of attack to the nodes is provided in Figure 7. In this figure, the first parameter of the threshold secret sharing technique, which is equal to the minimum degree of each node in the network, is assumed to be equal to 3. The number of nodes is changing between 0 and 1000. As a comparative basis, identical assumptions for the degree of the nodes and the

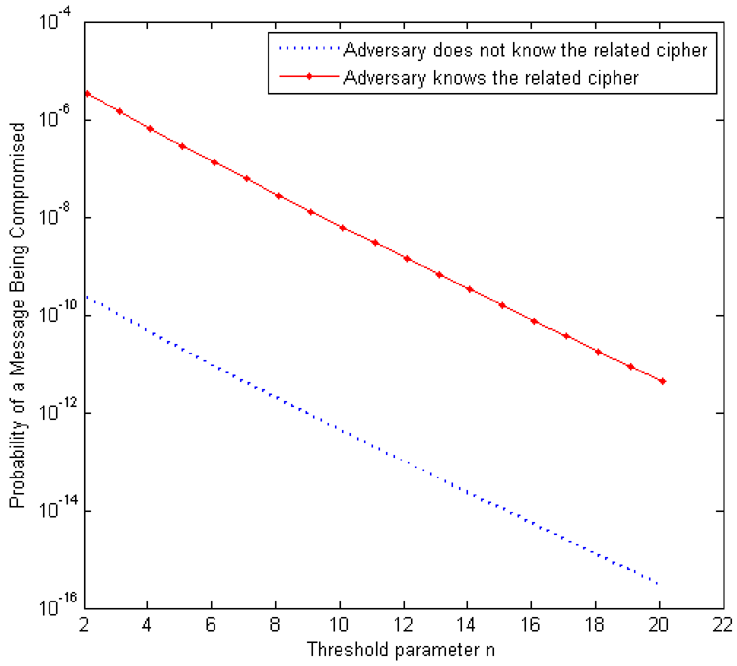


Figure 6. Comparing the probability of a message being compromised based on threshold parameter values in the two cases where the adversary either knows or does not know the related cipher

number of nodes in the network have been made by Du et al. [Du et al. 07]. Figure 7 shows that even if 90% of the nodes in the network are successfully attacked in a network with 200 nodes, the probability of a message being compromised is about 10^{-4}. This amount is significantly better than the EG scheme and Du et al.'s scheme based on their published results [Du et al. 07] where with 180 nodes being compromised in the network, adversaries can obtain messages with probability 0.4 in the EG scheme and 0.05 in Du et al.'s scheme [Du et al. 07] [Eschenauer and Gligor 02]. Du et al. [Du et al. 07] showed that their scheme outperforms EG [Eschenauer and Gligor 02] and q-composite [Chan et al. 03]. Figure 8 provides a comparison between the proposed scheme and Du et al.'s scheme. In Figure 8, the parameters values used for Du et al.'s scheme are the same as the parameters values in their own simulation (i.e., M=200, l=10, pool size=10,000) [Du et al. 07]. It is observable in Figure 8 that the proposed scheme outperforms Du et al.'s scheme with a fair difference when the number of compromised nodes is between 0 and 1000.

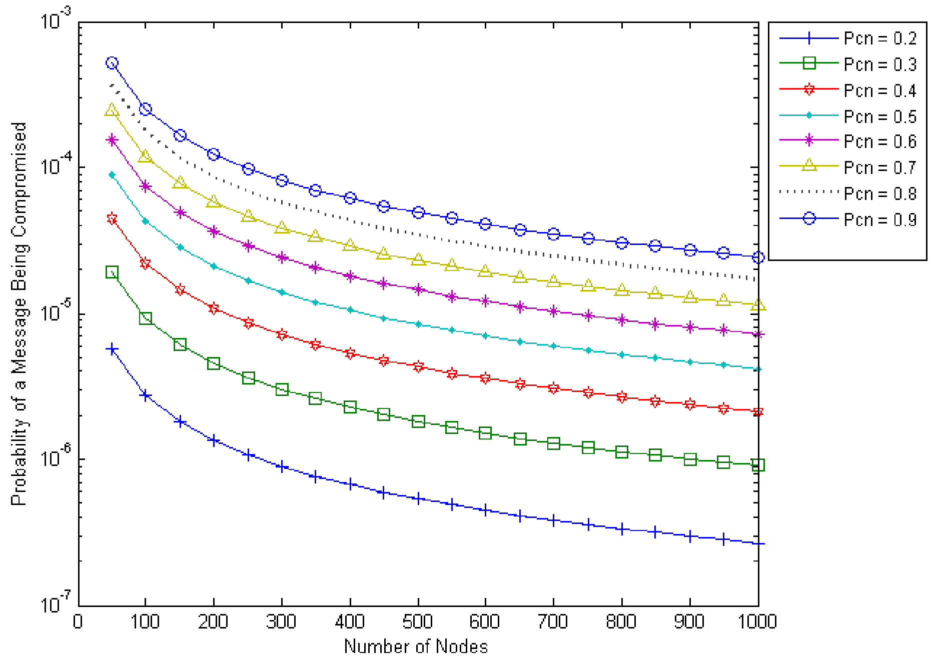


Figure 7. Comparing the probability of a message being compromised based on the number of nodes in the network for different probability values of a node being compromised (P_{cn})

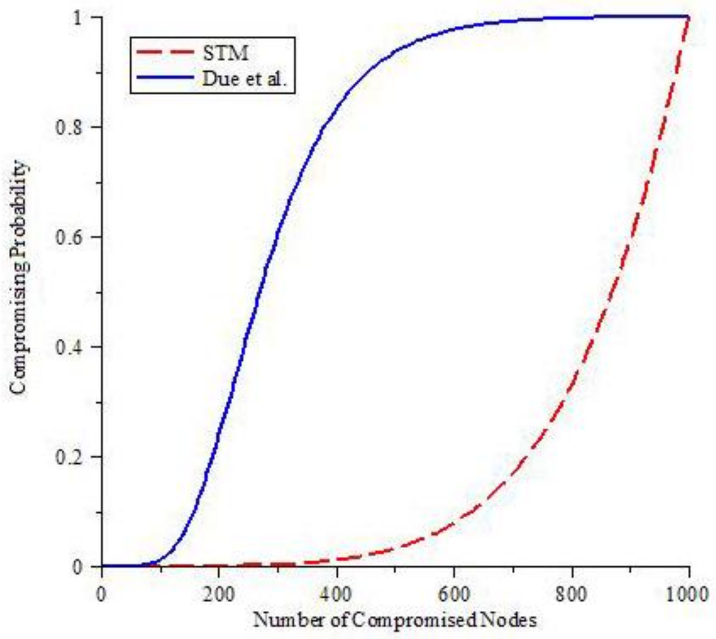


Figure 8. Comparing the proposed scheme STM (n=10, m=5) and Du et al.'s scheme (M=200, l=10). Pool size is 10,000 and network size is 1,000

3.4.2.2 Attack to component 6 (shareholder-destination links)

Figures 9 and 10 show the effects of varying the parameters values on the probability that a message is compromised when adversaries eavesdrop third type links. These figures are drawn based on Lemma 2. In Figure 9, it is observable that by increasing the first threshold secret sharing parameter, the probability of a massage being compromised decreases. For example, when the probability of eavesdropping links is 0.9, with a threshold secret sharing scheme's first parameter being 10, the probability of a message being compromised is about 10^{-5} which could be considered very small compared to the published results of the EG scheme and Du et al.'s scheme with identical assumptions about the number of nodes and degree [Du et al. 07] [Eschenauer and Gligor 02].

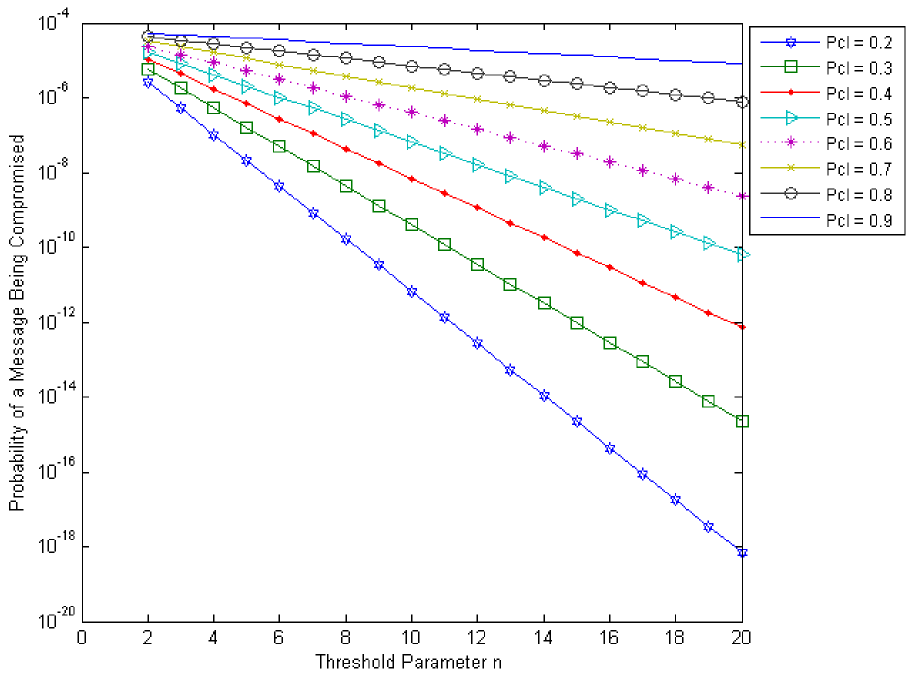


Figure 9. Comparing the probability of a message being compromised based on threshold parameter values for different probability values of a third type link being compromised (P_{cl}), by eavesdropping

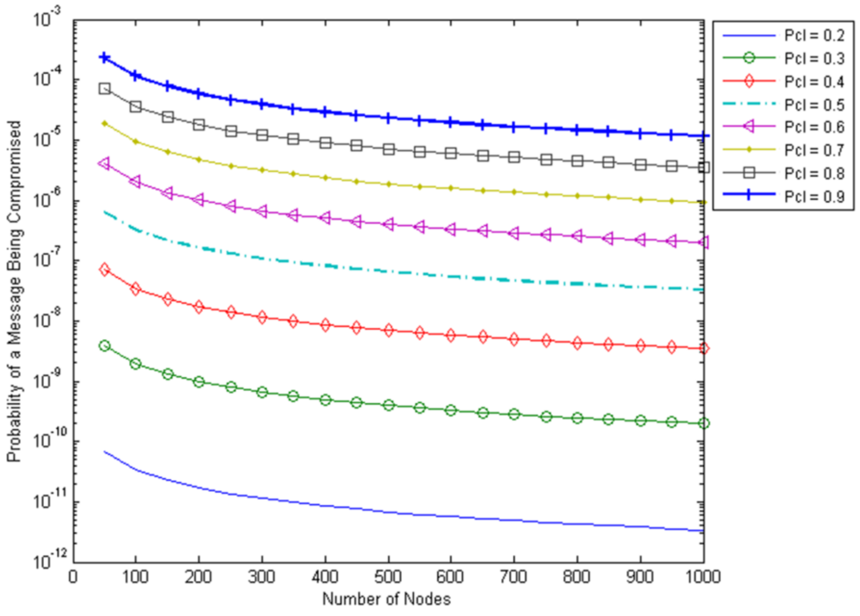


Figure 10. Comparing the probability of a message being compromised based on the number of nodes in the network for different probability values of a link being compromised (P_{cl})

CHAPTER IV

ENERGY-EFFICIENT AND SECURE IN-NETWORK STORAGE AND RETRIEVAL

4.1 Introduction

The second part of this dissertation work introduces a new scheme for storing and retrieving collected data in Wireless Sensor Networks. The proposed cluster-based scheme works with a threshold secret sharing technique and a symmetric key-management scheme. Several security and performance issues are addressed for this scheme. The sensed data is transmitted to the Cluster Heads (CH) in a secure manner and disseminated in the respective clusters using the threshold secret sharing technique. Moreover, an adaptive threshold technique is introduced that operates based on historical data and a multicasting protocol. The effect of several system parameters on energy consumption and overhead is investigated using simulations. The results of simulations indicate that the proposed scheme performs better than comparable related work. The proposed protocol is lightweight in terms of energy consumption and suitable for WSNs while providing reasonable security for stored data in the network. A method for data retrieval from the network is also investigated. A part of this work has already been published [Boloorchi and Samadzadeh 13].

4.1 Network Model

In this chapter, it is assumed that the network contains regular low power wireless sensors and they are responsible for sensing the data. The sensors convert the data they have collected from the environment to electrical signals. The electrical signals can be sent out up to a specific distance which is the range of the sensors in the network. It is assumed that all regular sensors have the same range. The radio model for wireless sensors defines the parameters that have influence on energy consumption. In this dissertation work, a radio model analogous to the one used by Heinzelman et al. is assumed where the number of bits in a message and the range of the nodes are effective factors on energy consumption for transmitting and receiving messages [Heinzelman et al. 00].

There might be a number of powerful nodes in the network that are equipped with tamper resistant devices to prevent them from being compromised. Compared to the regular sensors, these nodes have more computing capabilities and more memory space. Also, the existence of a network controller is assumed for offline reconfigurations and some online tasks.

In this work, networks are modelled with *random graphs* following the tradition of Eschenauer and Gligor [Eschenauer and Gligor 02]. A random graph, denoted by *G(n, p),* is a graph with n vertices for which an edge between two vertices is present with probability *p*.

Clustering is a classic solution for saving energy in different computer science fields [Xu and Wunsch 05]. In the proposed scheme, clustering is used where nodes in a cluster are closer to a specific node, called a Cluster Head (CH). Distance is considered as the direct geographical distance between two nodes; however, one may use the number of hops instead. CHs are responsible for cluster management, secret sharing, and some other tasks that are explained in detail in following sections. They are also the gates for clusters' communication with the outside of the clusters, i.e., with other CHs or with the users.

Two types of networks might use the proposed scheme: heterogeneous and homogeneous. For heterogeneous networks, it is assumed that the more powerful nodes and the regular sensors are deployed uniformly, and the clusters are formed during network deployment. CHs are the more-powerful nodes in these networks, and the cluster to which each sensor belongs is determined during deployment. For homogeneous networks, the clusters could be formed either at the time of sensor deployment or afterward. Forming the clusters during deployment is applicable only if the nodes are being deployed non-randomly since the cluster of each node is predetermined. In the case of

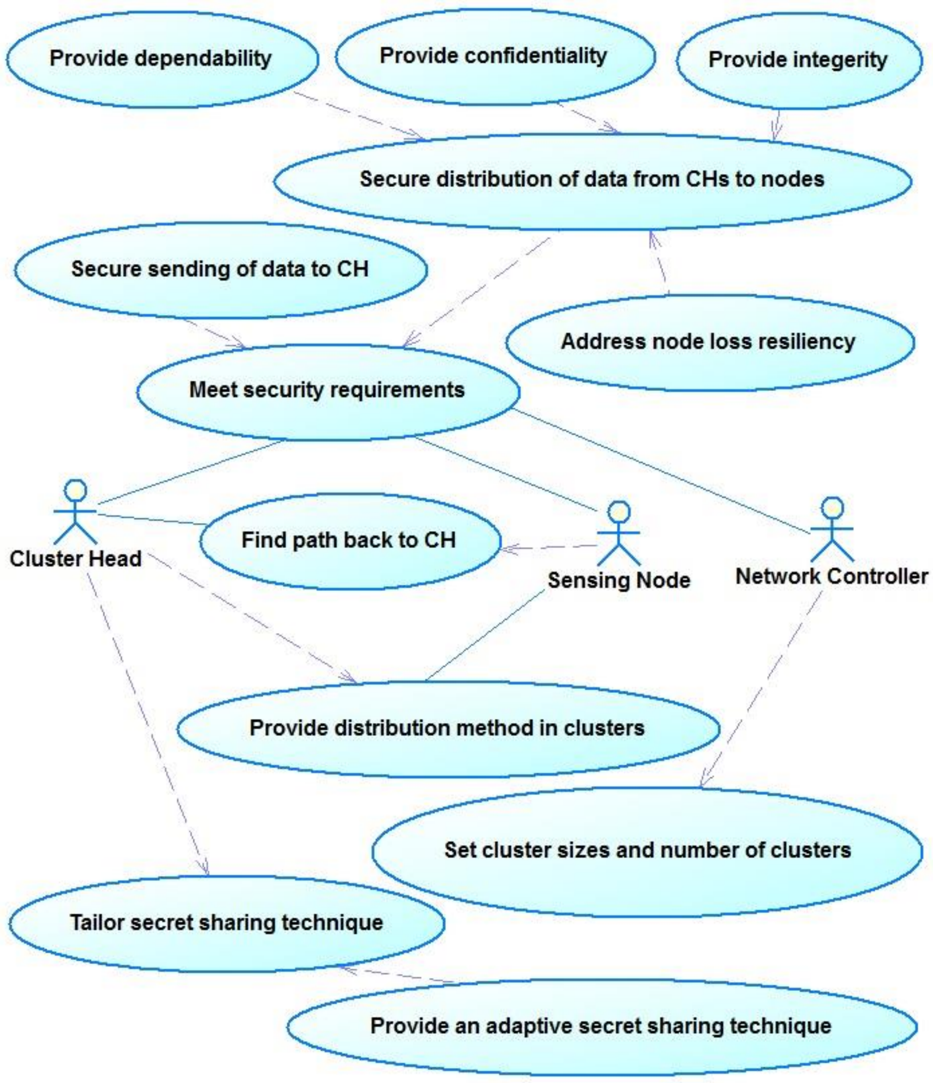

Provide dependability
Provide confidentiality
Provide integerity
Secure distribution of data from CHs to nodes
Secure sending of data to CH
Address node loss resiliency
Meet security requirements
Cluster Head
Find path back to CH
Sensing Node
Network Controller
Provide distribution method in clusters
Set cluster sizes and number of clusters
Tailor secret sharing technique
Provide an adaptive secret sharing technique

Figure 11. Usecase diagram for in-network storage of sensed data

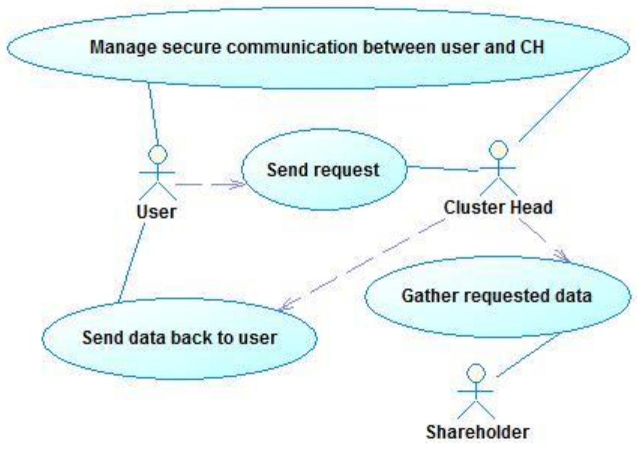


Figure 12. Usecase diagram for retrieval of data

homogeneous networks, in which the nodes are randomly distributed, e.g., from a plane, several energy-efficient clustering algorithms for wireless sensor networks have been proposed [Heinzelman et al. 00] [Younis and Fahmy 04].

The number of clusters and consequently the number of nodes in each cluster depend on several factors such as the accuracy needed for the system, the energy cost, and the needed security. Accuracy in this context refers to how accurate the access of the users is expected to be to each geographic area. For calculating the energy cost, several factors are considered. These factors include the overhead of computation for the threshold secret sharing technique and the key-management approach, and the overhead of communication among network nodes.

4.2 Method

In this section, several issues of in-network storage, retrieval, and communication are discussed. Figure 11 is a usecase diagram [Booch et al. 05] of the proposed method for in-network storage of sensed data. A usecase diagram of the proposed scheme for retrieval of data appears in Figure 12.In the following subsections, first, a protocol for dissemination of sensory data is introduced followed by a discussion of the major security requirements.

Next, a protocol is proposed for retrieving the stored data upon request from users, followed by a discussion of the protocol's security. A number of methods to increase the scalability and performance of the proposed scheme are proposed at the end of this section.

4.2.1 Distribution of Sensed Data from Cluster Nodes to a CH

To enhance the security of communications between nodes in clusters and their respective CHs, the idea of pre-distributed shared keys [Ren et al. 08] [Eschenauer and Gligor 02] [Du et al. 07] is adopted with some adaptation. In the proposed method, the existence of a key pool is assumed and each node, including the CHs, draws a number of distinct keys from the key pool. Each key in the pool has a unique ID. The next step is to find at least one shared key between each CH and its nodes. Each node sends a clean-text list (without encryption) of its key IDs to its CH. Note that this list does not give potential adversaries any information they did not already have since the keys themselves are not related to the IDs, and only the network controller has the list of each node's keys. CHs check their keys to find shared keys with each node in their cluster. If a CH finds any shared key with a node in its cluster, it sends the ID of that key to the corresponding node. Such nodes, called *connected* nodes, can now send the sensed data to the CHs using their shared keys with their respective CH in a more secure manner.

In cases where there is no shared key between a CH and a node in its cluster, the CH broadcasts the list of IDs of keys of that node through the cluster. Each connected node checks the broadcast list against its own keys, and if any shared key(s) is found, the connected node sends the shared key to the CH. After this procedure, if there is any node in a cluster that is not connected, the corresponding CH sends an encrypted message to the network controller and asks for a shared key with the unconnected node. The key that a CH uses for this communication with the network controller is $K_{ch} = k_1 \oplus k_2 \oplus \ldots \oplus k_i$, where $\oplus$ stands for *exclusive or* and k_1 through k_i are the CH's keys [Eschenauer and Gligor 02]. The message from a CH to the network controller is a list containing a pseudo-randomly chosen key ID from each unconnected node's key ID list that has already been sent from each node to its respective

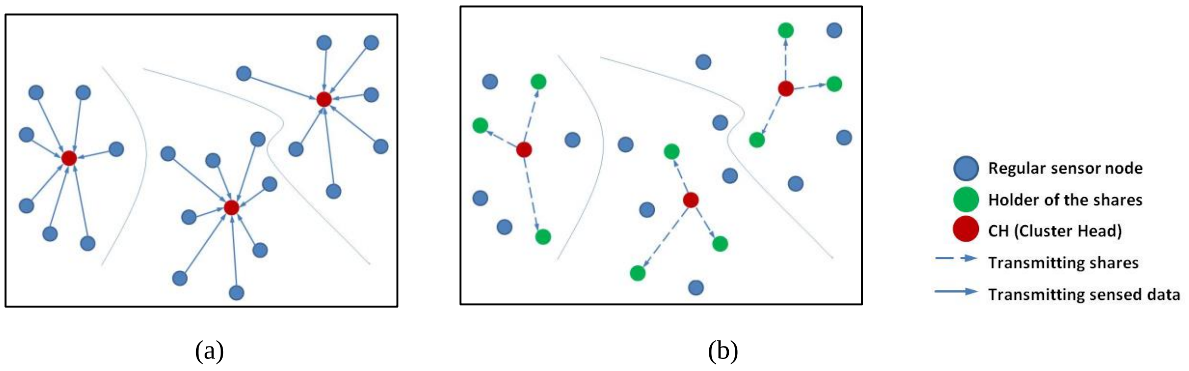


Figure 13. An example of storing sensed data in a network of 25 nodes and 3 clusters with parameters of threshold secret sharing being n=3 and m= 2 (a) Nodes in each cluster send their sensed data to the corresponding CH (b) Each CH chooses *n* nodes in its cluster and sends the pieces of the key that is used to encrypt the data to those *n* nodes, with *n* being the first parameter in the threshold secret sharing technique

CH encrypted with K_{ch}. Several other methods have been introduced in other studies to deal with the unconnected nodes [Du et al. 07] [Chan et al. 03].

The network controller finds the shared key for each unconnected node and creates a message consisting of the unconnected nodes' keys and their IDs. Then the network controller encrypts the message with K_{ch} to send it back to the respective CH. Note that the network controller has a list of each node's (including the CHs) keys so that it can indeed find shared keys with the unconnected nodes. Finally, CHs send the ID of the shared keys to the previously unconnected nodes and they become connected.

As mentioned earlier, after the key pre-distribution, each node can send its sensed data to the corresponding CH in a more secure manner. Each node encrypts the sensed data using its shared key and sends it to the CH, and the CH decrypts the data and uses it in the subsequent procedures. Figure 13 is an example of how the proposed in-network storage scheme works.

4.2.2 Storage of the Clusters' Data

In each CH, the data received from its sensors are concatenated. Then, the threshold secret sharing technique is utilized to split the concatenated data into shares. Finally, the CHs distribute the shares of the data to the nodes in their clusters. In the following subsections, several requirements towards the enhanced security of the concatenated data's storage and communication are addressed.

4.2.2.1 Dependability

CHs use the threshold secret sharing technique to provide dependability defined as the situation where the network is compromise resilient and fault tolerant. Since the volume of data might be large, it might be better to encrypt the data with a key, apply the threshold secret sharing technique to the key, and then distribute the key. For the data, a communication coding technique such as (n, k) Reed-Solomon, as suggested in Ren et al.'s work, could be used [Ren et al. 03].

4.2.2.2 Confidentiality

The threshold technique provides confidentiality for the data without using keys to encrypt and decrypt them [Shamir 79]. As mentioned in Subsection 4.3.2.1 above, it might be better to encrypt the data with a key and split the key instead of splitting the data into shares. To prevent the shares of the data or the key from being disclosed in case a node is compromised, it is not desirable that each node decrypt the shares. To this end, the shared keys described in Section 2.2 should not be used for these communication sessions; rather they should only be utilized to send the sensed data to the respective CH. The CHs could encrypt the shares with a new key and save the key.

A critical problem in any centralized or semi-centralized topology in hostile environments is that the central nodes (e.g., CHs in the proposed work in this Chapter) are good targets for adversaries since they contain important data. The proposed scheme is not an exception and, by compromising a CH, adversaries could obtain secret data about the entire cluster. To prevent this from happening, if the network is heterogeneous with powerful CHs, tamper-resistant hardware can be installed and used [Du et al. 07]. In homogeneous networks, the CHs use the threshold secret sharing technique to split the key that is used to encrypt the data and distribute the shares in their cluster among their nodes.

Asymmetric cryptographic methods typically have large overhead and thus they are generally not good solutions for sensor networks [Stallings 10] although sensor node technologies have undergone some improvements in the areas of memory size and processing capabilities [Boukerche 08]. Nevertheless, an alternative method is proposed here to prevent CHs from being detected by potential adversaries. CHs could be changed anonymously at certain times so that an adversary will not be able to use network traffic analysis to easily detect the CHs. Sending packets in such a way that each forwarding node only knows the next step could also improve the anonymity of the CHs. In this method, since only the direct neighbors of each CH know the CH, CHs would be hidden from adversaries until one of their neighbors is compromised.

4.2.2.3 Authentication

A pre-installed Hash or MAC technique [Stallings 10] is utilized to provide integrity and authentication in the proposed scheme.

4.2.2.4 Node Loss Resilience

In the case of homogeneous networks, because of the redundancy that is provided by the threshold secret sharing technique, the system can tolerate up to (n – m) node losses for full data retrieval in an (n, m) threshold scheme [Shamir 79]. Even in the case of more than (n – m) node losses, a reasonable amount of the overall sensed data can be retrieved since the data from different nodes are distributed among different nodes in a cluster. Ordinarily, the number of nodes in a large scale network is tentatively more than the n used in an (n, m) threshold scheme so, in each distribution session, a potentially different set of n nodes is chosen to be the holders of the shares.

In the proposed technique, a back-up scheme is proposed to tolerate losing CHs. A node in each cluster is assigned as a vice CH. Whenever something is updated in a CH, that CH sends a message containing all lists and the important changed data encrypted with a shared key to the vice CH. Note that the shared key could be obtained via the shared key scheme described in Subsection 4.3.1.

4.2.3 Retrieval of Data

In the proposed scheme, users may want to access the data of a specific geographical location in the network. CHs are responsible for providing enhanced security for users' access to the data. In other words, each CH is the interface between the sensors in its constituent nodes and the users who may need any data that is gathered by those nodes. A user sends a request for data to a CH. The CH gathers the encrypted data's shares and the related key's shares from the sensors.

There are two options for providing access to the data for a requesting user. In the first method, the CH that receives a request reconstructs the key from the shares it gathers using the threshold secret sharing technique, decrypts the data, encrypts the data with the user's public key, and sends the encrypted data to the user. Upon receiving the encrypted data, the user can decrypt the data with its private key. In the second method, the CH concatenates the encrypted data shares, encrypts the generated package with the user's public key, and sends the encrypted package to the user. In this method, the user decrypts the encrypted package with its private key and is responsible for using the threshold secret sharing technique to reconstruct the data from the shares. The first method places a larger portion of the overhead on the CHs whereas the second method places a larger portion of the overhead on the user. Since usually the users have less energy and space sensitive equipment than the CHs, the second method is recommended.

4.2.4 Adaptive Secret Sharing Based on Historical Data

In a hostile environment, different areas typically have different levels of hostility. This is because of several factors such as the presence of adversaries or environmental parameters such as inclement weather in different areas. In such situations, it is reasonable to adjust the parameters of the threshold secret sharing technique based on the level of environmental harshness. In this section, the following method is proposed for instantiating these parameters.

The system starts with pre-defined parameters *n* and *m* in an *(n, m)* threshold secret sharing technique for all clusters regardless of the harshness of their respective geographic areas. After a certain periods of times, the historical data, which can be obtained from intrusion detection techniques or authentication techniques such as hash functions, is used to furnish feedback to the system. One method for predicting future events using past data is by using posteriori distributions [Jøsang and Ismail 02]. It is assumed that one of two distinct events can be reported whenever a CH sends a packet to or receives a packet from a node in its cluster. The two events are: "*compromised*" or "*safe*". A compromised event could be a node being compromised, a link being eavesdropped, the data of a link being manipulated, active attacks against a node, a link or node loss, or in general any problem that can be detected in a communication between a CH and a node. If an event is not reported as a compromised one, it is a safe event.

Based on the above assumptions, a binary event is attached to each communication and its probability is represented by Bernoulli distributions. The posteriori distributions of Bernoulli distributions are beta distributions that can be used to show the probability of occurrence of one event based on the previous occurrences of that event [Jøsang and Ismail 02] [Pissinou and Crosby 07] (further details about the estimation of binary events can be found in standard textbooks on probability theory, e.g., *Statistical Inference* [Casella and Berger 90]). Jøsang and Ismail introduced the use of the beta density function in reputation systems [Jøsang and Ismail 02]. They used the the reputation system's feedback as parameters of the beta density function. Based on similar reasoning process, by using the feedback of the proposed scheme, the same method could be adopted here as well. In what follows, the results of Jøsang and Ismail's work are presented and used to predict the probability of occurrence of a compromised event in each cluster.

The beta density function is given below where α and β are real numbers greater than 0.

$$f(p:\alpha,\beta) = \frac{p^{\alpha-1}\,(1-p)^{\beta-1}}{\int_0^1 p^{\alpha-1}(1-p)^{\beta-1}dp} \quad , 0 < p < 1$$

The expected value of beta distributions is

$$E(p) = \frac{\alpha}{\alpha + \beta}$$

Two outcomes of a process, x and x', are assumed where x is a compromised event report and x' is a safe event report. If the number of observed instances of x is r and the number of observed instances of x' is s, then the probability density function of the observed outcome x in the future can be represented as a function of past observations by letting $\alpha = r + 1$ and $\beta = s + 1$, where $r, s \geq 0$ [Jøsang and Ismail 02].

The expected value represents the chance of future attacks. Based on this analysis, the threshold secret sharing technique's parameters can be modified either for more or for less resilience in order to avoid unnecessary redundancies. For example, if the environment is reasonably safe, choosing a large n or having a large difference between n and m would result in unnecessary energy consumption for communication as well as in waste of space. More energy would be saved if the scheme has just enough node loss resilience so that even in the case of losing a number of nodes the network could keep working.

4.3 Performance Analysis

In this section, the energy spent in the basic scheme [Ren et al. 08] and the proposed enhanced scheme are compared theoretically. In the next section, with slightly different assumptions, a comparison between the two schemes is provided using simulations. The impact of several parameters on energy consumption in the proposed enhanced scheme is also illustrated in the next section.

The computational complexity of the threshold secret sharing technique is essentially the same in both schemes since they are both applied to keys of the same size that are used for the encryption of the sensed data. Using an energy efficient communication coding technique, which is independent of the size of data, to split the encrypted data, makes the computational overhead independent of the amount of data. Therefore, the energy spent on computation at each node, E_{cp}, in the basic scheme is essentially equal to the energy spent in each CH in the enhanced scheme.

As mentioned in Section 4.2, it is assumed that an *(n, m)* threshold scheme and an *(n, k)* Reed-Solomon coding technique are utilized. It is also assumed that the amount of energy consumption for communication of the sensed data from each node to a neighbor node is E_{DataCm}. The communicational overhead of transferring equal amounts of data between any two neighbors is assumed to be the same. Also, it is assumed that the sizes of the clusters are in a way that the CH in each cluster is a neighbor of every node in that cluster.

The communicational overhead for storing the sensed data is assumed to be the same in both basic and enhanced schemes, the reason being that the overall amount of data is the same and the number of communicated data packets are the same in both schemes. Since each packet is destined to a neighbor and it is assumed that the communicational overhead of two neighbors for the same amount of data is always equal, the communicational overhead of transferring data is the same in both schemes.

The following formula calculates the total communicational overhead for transferring data in the basic scheme.

$$E_{TotalDataCm} = E_{DataSplitsCm} * n * N$$

where *N* is the total number of nodes in the network and *n* is a parameter of the *(n, m)* threshold secret sharing technique.

Using an *(n, k)* Reed-Solomon coding technique, the data is divided into *n* shares whose sizes are the size of the data divided by *n* (in bytes, blocks, etc.).

The communicational overhead is directly proportional to the size of the data so the following equation is true where E_{DataCm} is the overall overhead for transferring the sensed data.

Communicational overhead for transferring data's splits: $E_{DataSplitsCm} = {E_{DataCm}}/{n}$

The communication overhead for the data in the enhanced scheme could be calculated as follows.

$$E_{Data:\,Sensors\ to\ CHs} = N_{CH} * \left(\frac{N}{N_{CH}} - 1\right) * E_{DataCm} = (N - N_{CH}) * E_{DataCm}$$
$$= (N - N_{CH}) * E_{DataSplitsCm} * n$$

where *N* is the number of nodes in the network, N_{CH} is the number of CHs, $\frac{N}{N_{CH}}$ is the average number of nodes in each cluster, and *n* is the coding technique parameter that indicates the number of shares to be sent out.

Communicational overhead to distribute the *n* shares of data from the CHs: $E_{DataCH} = N_{CH} * n * E_{DataSplitsCm}$

Total overhead for the enhanced scheme:

$$E_{TotalDataCm} = E_{Data:\,Sensors\ to\ CHs} + E_{DataCH}$$
$$= (N - N_{CH}) * E_{DataSplitsCm} * n + N_{CH} * n * E_{DataSplitsCm}$$
$$= E_{DataSplitsCm} * n * N$$

As discussed above, the communicational overhead for data is the same for both basic and enhanced schemes. In the rest of this section, the communication-related discussions are only about the keys that are used to encrypt the data. The keys are split using the threshold secret sharing technique. Therefore, to compare the basic scheme's overhead with the enhanced scheme's energy overhead, E_{total} is considered only as the sum of the overall computational overhead and the communicational overhead for transferring shares of the keys.

basic scheme: $E_{total} = N * E_{cp} + N * n * E_{KeyCm}$

In the basic scheme's equation, the computational energy for each node, E_{cp}, is the computational overhead for splitting the encrypted data, as well as for splitting the key and later for reconstructing them, thus making the total computational overhead $N * E_{cp}$. After applying the threshold secret sharing technique on the key that is used to encrypt the sensed data, each node chooses n of its neighbors and distributes the shares among those nodes, which would be $N * n * E_{KeyCm}$ energy units for N nodes in the network.

In the enhanced scheme, the total computational overhead is the sum of the overheads for splitting the encrypted data and the key, and later reconstructing them in all CHs. In the enhanced scheme, a pre-distributed symmetric key-management technique is utilized. The computational overhead of this technique, *O(n)* where *n* is the key size [Stallings 10], could very small compared to the threshold secret sharing technique's overhead $O(n^2 \log n)$ [Shamir 79], and could be ignored. In the case where the overhead of reconstructing the keys are put on the users, the computational overhead of the pre-distributed key-management technique and the threshold secret sharing technique in the network, *O(n)* [Stallings 10] [Shamir 79], could be considered equal, i.e.,

$$T_{cp} = N_{CH} * E_{cp} + (N - N_{CH}) * E_{cp} = N * E_{cp}$$

where N_{CH} is the number of CHs in the network, T_{cp} is the total computational overhead, E_{cp} is the computational overhead for splitting the encrypted data, and N is the total number of nodes in the network.

As for the communicational overhead, each CH distributes n shares of its encrypted data and the key used to encrypt the data among n of the nodes in its cluster, which would amount to spending the following amount of energy:

Communicational overhead to distribute the n shares from the CHs: $E_{CH} = N_{CH} * n * E_{KeyCm}$

Note that when a key is split using the threshold secret sharing technique, the shares are by definition the same size as the original key [Shamir 79].

The total overhead for the enhanced scheme can be represented as follows.

Enhanced scheme: $E_{total} = T_{cp} + E_{CH} = N * E_{cp} + N_{CH} * n * E_{KeyCm}$

The total energy of the computation is equal to the sum of E_{cp} of all CHs and non-CH nodes, i.e., $N * E_{cp}$, which is equal to the total E_{cp} in the basic scheme. The communicational overhead is $N * n * E_{KeyCm}$ for the basic scheme and $N_{CH} * n * E_{KeyCm}$ for the enhanced scheme. As long as the number of clusters is less than the number of nodes in the network, which is usually the case, the communicational overhead is less for the enhanced scheme.

In this analysis, the communicational overhead or the energy consumption for sending a key share from a node to a neighbor is E_{KeyCm}. Based on this assumption, it is implicitly assumed that all neighbors are at the same distance to a source node. Although this assumption might not be universally true, it is the same for both the basic [Ren et al. 08] and the enhanced schemes, so the comparison is still reliable. However, the CHs are also considered neighbors of the nodes in the clusters. This assumption is true only for a specific subset of the CHs and the nodes in the network. The reason is that the distances in the clusters increase by reducing the number of CHs, which means that more communication energy would be needed to send the sensed data to the CHs. On the other hand, this subset of the CHs and nodes in the network should not be very large, the reason being that the communicational overhead increases by having more CHs in the network as a result of distributing the shares from the CHs. In the next section, simulation results are provided to show that there is an optimal value for the number of CHs and nodes for achieving optimal energy saving.

4.4 Simulation

In this section, a comparison between the basic and enhanced schemes is provided using simulations. It is assumed that the threshold secret sharing technique is applied to the data itself and not to a key that is used to encrypt the data. The radio model that is used in this work (see Section 4.2 for more detail about the model) and the communication overhead is considered to be proportional to the square of the distances [Heinzelman et al. 00]. For simplicity, it is assumed that "distance" here is the direct geographical distance between each node and its respective CH. In this simulation, the focus is on whether or not each parameter (nodes' wireless range, threshold parameter, and computational overhead) affects the overall amount of energy consumption and not on the actual magnitude of the effect, which is relegated to future work. Hence, instead of the square of the distances in the radio model [Heinzelman et al. 00], the distances themselves are considered in this work. Also, the value of the parameters is scaled in a way that the effect of the changing them would be observable. Of course, the scaling factors are kept constant during the simulation to make the comparisons possible.

Java is used to implement the simulation for this scheme. In this simulation, 1,000 nodes of the virtual environment are clustered and the energy overhead is computed using the following formula.

$$E = \sum_{i=1}^{N_{CH}} \sum_{j=1}^{N_i} (E_{cp} + E_u * d_{ij} + n * E_u)$$

where N_{CH} is the number of clusters in the network, N_i is the number of nodes in cluster i, E_{cp} is the energy spent on the threshold secret sharing procedure, E_u is the needed energy for transmitting one unit of data in one hop, d_{ij} is the distance between node *j* in cluster *i* and the CH in cluster *i,* and *n* is the first parameter of the *(n, m)* threshold secret sharing technique used. Note that this equation shows the proportionality of the total energy with respect to the parameters on the right-hand side. These are the only parameters that vary with the changes in the number of clusters.

To find the optimum number of clusters that would optimize energy consumption for different system parameters, a number of simulation runs were conducted. In this section, the effect of the following parameters on the overall energy consumption are compared: 1) range of sensors, 2) variable *n* in an *(n, m)* threshold secret sharing technique, and 3) computational overhead.

In the rest of this section, the variations of the consumed energy when the number of clusters increases are investigated. For varying number of CHs, the effect of the aforementioned parameters is shown in Figures 14 through 16. Note that the system is completely centralized when the number of clusters is 1 and, as the number of clusters increases, the system behaves in a more distributed fashion.

In Figure 14, the range of the sensors is changed and its effect on energy variation is shown. In this work, the simulation is run with the aim of finding how (but not by how much) several parameters affect energy consumption in the network.

In Figure 14, the energy consumption for four values of the sensors' ranges is depicted. The values

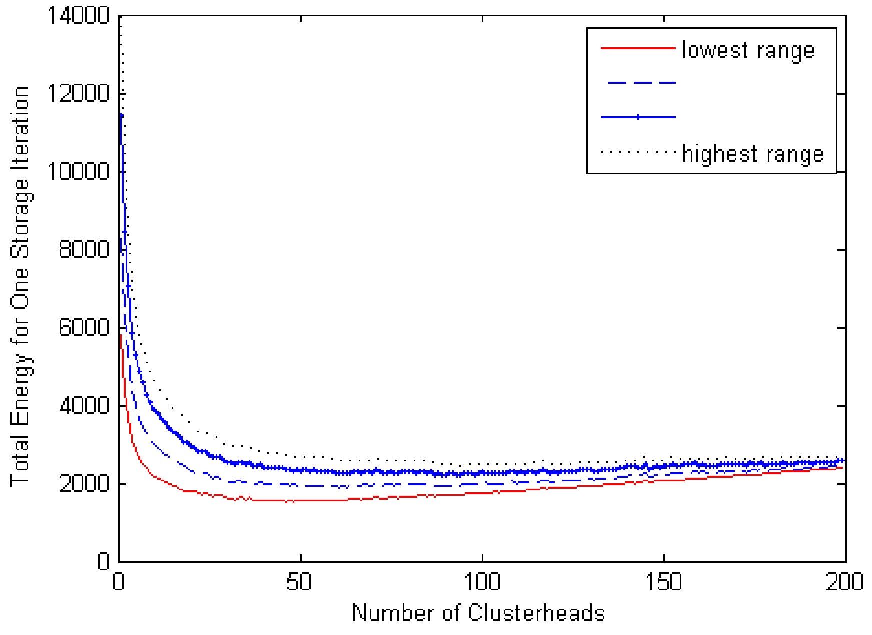


Figure 14. Impact of sensors' communication range on energy consumption for different number of clusters for one iteration of running the proposed storage scheme

indicate the proportionality among the parameters and the direction of changes in a parameter upon changing the value of the other parameters. As stated previously, the study of the magnitude of changes is relegated to future work.

Based on Figure 14, it is better to use a dynamic range protocol as opposed to a static range protocol, where all sensors have the same range which does not change during the network's lifetime. The dynamic range protocol works in a way that each node can find the shortest path to a CH and adjust its range to communicate to its nearest node in the path. Although it was assumed that there is a straight path from the nodes to the CHs in the simulation, the results adequately show the effect of range on energy overhead.

Figure 15 depicts the changes of the threshold secret sharing technique's variable on the energy consumption. In this figure, the result for five of the threshold parameter's value is depicted. Simulation results show that the changes for other values follow the same pattern. As the number of clusters

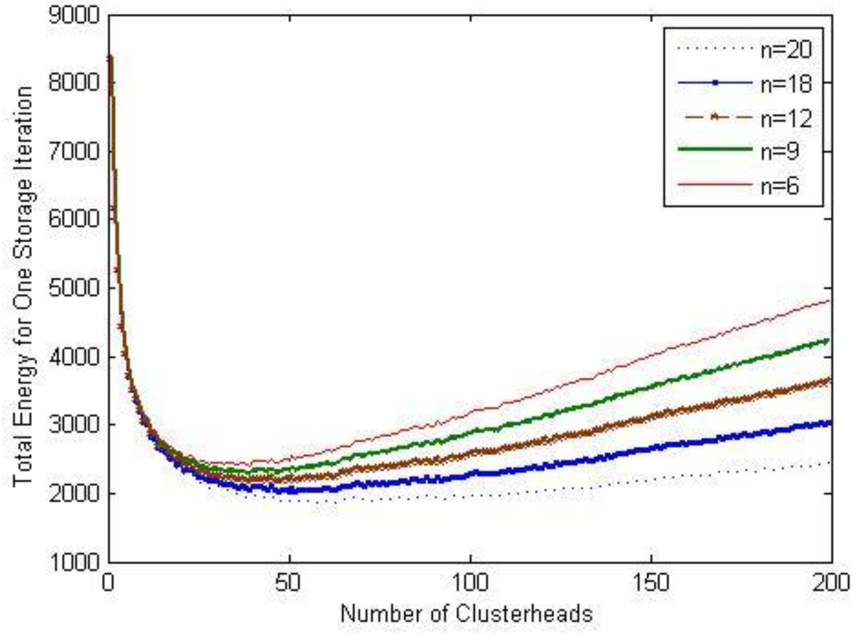


Figure 15. Impact of threshold parameter on energy consumption for different number of clusters for one iteration of running the proposed storage scheme considering different values for the secret sharing technique's variable n

increases, i.e., as more accurate access to the geographical locations are provided for the users, the effect of *n* is more observable, which indicates that for energy there is a trade-off between accuracy and *n*. Based on Figure 15, it could be inferred that, in a more accurate system, i.e., with smaller clusters, the role of the proposed adaptive threshold secret sharing protocol is more pronounced in saving energy. The reason is that the smaller the threshold parameter, the less energy is spent in a cluster.

In this work, the accuracy of a network is defined as the accuracy with which a user can access information in different geographical regions covered by the network. Since information from different regions of a network is sent to the nearest CH, it can be argued that, by the definition given above, the accuracy of the network directly depends on the number of clusters. It is observable from Figure 16 that, in less accurate networks, it is reasonable to do more in-network computation and data aggregation resulting in saving space within the network and saving energy in communicating with entities outside the network. Note that in Figure 16 the computational overhead might be calculated based on different factors in different networks. In this study, only the proportionality is investigated and the four provided

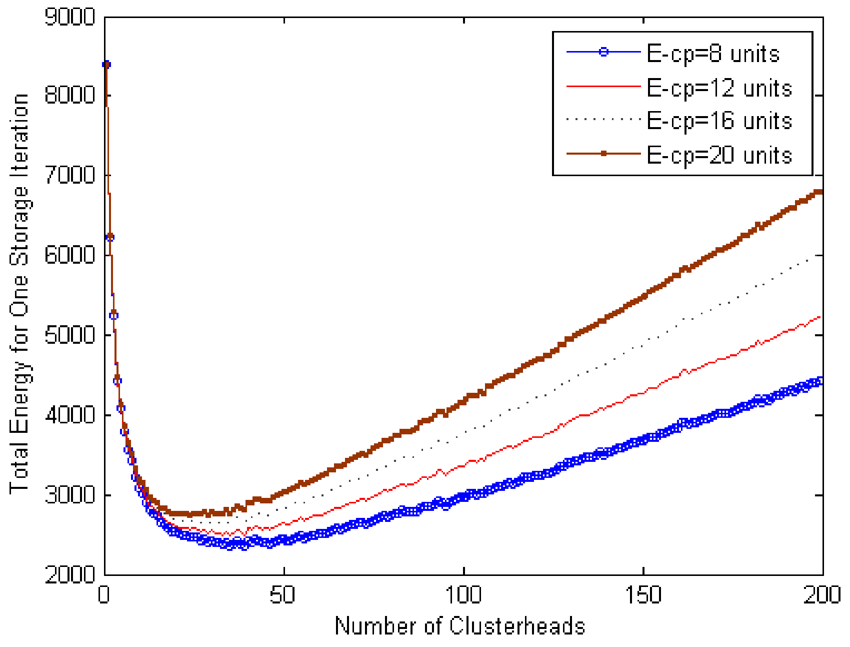


Figure 16. Impact of computational energy on energy consumption for different number of clusters for one iteration of running the proposed storage scheme

values for computational overhead (8, 12, 16, and 20) are found to be illustrative. Considering the actual magnitude of the changes based on the changes in the computational overhead is beyond the focus of this study.

In the analyses provided in this section, in the extreme case where the number of nodes is equal to the number of CHs, the formed clusters would be the same as the basic scheme [Ren et al. 08]. In Figures 14, 15, and 16 the largest number of CHs is 200 and the number of nodes is 1,000. However, based on the observation that with more than about 50 CHs the curves are strictly ascending in all three graphs, it could be stated that for 1,000 CHs (the maximum possible number of clusters in a 1,000-node network), the total consumed energy is more than the minimum of the curves which occurs at around 50 CHs. This analysis shows that the enhanced scheme could outperform the basic scheme for certain range of values of the network parameters.

4.5 Security Analysis

The security goal of the proposed scheme is to protect the sensed data from being compromised while stored in the network. The security of the three main types of communication links should be provided: the links between the network controller and each CH, the links between nodes and their corresponding CH, and the links between CHs and the nodes selected for storing the shares generated by the threshold secret sharing technique. For the first type of communication links, an on-demand key based on pre-distributed keys is used that establishes a symmetric scheme since both CHs and the network controller use the same key. In the next type of links, the EG technique [Eschenauer and Gligor 02] is adopted which has been used in several other systems and has been shown to provide a secure scheme in several studies [Chan et al. 03] [Du et al. 07]. Finally, the links between the CHs and the nodes in clusters are not needed to be protected since the shares from threshold secret sharing technique are transmitted through these links and acquiring fewer than m shares (with m being the second threshold secret sharing technique's parameter) does not provide the potential adversaries with any information about the secret.

CHAPTER V

A NEW PARALLELISM-AWARE CLUSTERING ALGORITHM

5.1 Introduction

A new clustering algorithm is proposed to support energy-sensitive networks in the third part of this work. The proposed algorithm groups the nodes of a given network into clusters of different sizes. The proposed clustering algorithm is built upon two quantitative aspects of the underlying network. The first aspect is the required number of nodes in each cluster that could be different for each cluster. The number of nodes in each cluster is aimed to match the size expected for that cluster. In this scheme, the expected size for each cluster is determined based on a pseudo-randomly generated input. The second quantitative aspect of the underlying network is the consideration of the sum of the distances between the nodes in each cluster and the cluster's corresponding CH. CHs are chosen based on their locations, which means that they are chosen in a way that the sum of the distances between nodes and their CH is minimized. For this scheme, first a basic algorithm is introduced which is subsequently extended in two stages. The proposed algorithm is simulated based on pseudo-randomly generated expected number of clusters with different sizes.

The results show that the number of clusters with each size in the clustered network closely correspond to the given expected numbers. The proposed algorithm is executed based on 100 different pseudo-randomly generated "expected number of clusters" with different sizes to prototypically evaluate the applicability of the algorithm for more general cases. The scalability of the proposed algorithm is discussed with the observation that the algorithm would work for different network sizes.

To the best of our knowledge, the proposed clustering algorithm is the first clustering algorithm that can provide an infrastructure to make task scheduling efficient and provide a solution to the problem of task scheduling without needing to invoke time consuming task scheduling optimization methods. This chapter is an extension to the published work by the author [Boloorchi 11] [Boloorchi et al. 14b].

5.2 Network Model

In this work, the focus is on a network with a number of homogeneous components where each component is a processing unit with local memory, and no limitation is imposed on the number of components in the network. By considering clusters with different number of homogeneous sensors as nodes of another network, it is possible to convert a homogeneous network that is clustered using the proposed clustering algorithm to a heterogeneous non-clustered network (composed of nodes with different processing capabilities and memory space capacities). Assuming a specific pattern for deployment of the WSNs might reduce the generality and generalizability of the proposed algorithm and complicate the justification of the proposed algorithm. Thus, a simple deployment pattern is assumed which could be adapted for different situations by changing the neighbor and distance definitions. In this chapter, it is assumed that the components are deployed in rows and columns, and the layout of the network is assumed to be rectangular with a hard boundary (i.e., the network is assumed to be a rectangular grid). The components to the right and left of a component as well as above and below it are called its neighbors. Each component is directly connected to its four neighbors. The distance between two neighbors is defined to be one unit. Since the layout has a hard boundary, there are components at the periphery of the network which might have fewer than four neighbors. Investigation of the case where the components are distributed in a pattern with differing neighbor distances, which misshapes the rectangular layout, is relegated to the future work in this area.

The proposed clustering algorithm needs to be initially centrally deployed, i.e., a centralized node is needed to manage the clustering.

5.3 Assumptions

In this work, following the conventions of similar studies [Yu and Prasanna 05] [Tian and Ekici 07], applications are assumed to consist of a set of tasks that are represented as Directed Acyclic Graphs (DAGs) with tasks being the vertices and the edges being direct communications between pairs of tasks.

The input to the proposed clustering algorithm is a list of a number of cluster types together with how many of each cluster type is needed. An example of the input list is given in Table I. In this example, it is assumed that the clusters have between 6 and 25 nodes.

Depending on the network that uses the proposed clustering algorithm, the input list can be generated using predictive methods based on historical data, e.g., based on the number and size of the clusters used in the past. However, the focus of this work is on the clustering algorithm. Therefore, as a simplifying assumption and to maintain narrow research focus, the input, based on which the algorithm informs the clustered network, is generated pseudo-randomly. To evaluate the level of

Table I. Number of clusters of each “cluster size”. Granularity factor in this example is 4.

Cluster size	6	7	8	9	10	11	12	13	14	15
No. of Clusters	1	2	0	1	2	0	1	2	0	4
Cluster size	16	17	18	19	20	21	22	23	24	25
No. of Clusters	1	1	0	0	2	3	0	3	2	0

Table II. The mapping of the first and last four clusters of Table I to cluster types with a granularity factor of 4.

Cluster size	6	7	8	9
No. of Clusters	1	2	0	1
Cluster size	22	23	24	25
No. of Clusters	0	3	2	0

Cluster type	Cluster Type's Range	Median of the Group range	No. of Clusters
1	{6, 7, 8 , 9}	7.5	4
5	{22, 23, 24, 25}	23.5	5

dependence/sensitivity of the algorithm on the pseudo-randomly generated input, the algorithm was tested for 100 different pseudo-randomly generated inputs as outlined in Subsection 5.5.4.

As for using historical data to predict the needed number of clusters of different size, it should be noted that generating an accurate table like Table I might not be easy. In other words, precisely determining the number of clusters needed for each cluster size would be a nontrivial task. To make this decision easier, in this work, without the loss of generality, the clusters are grouped based on a parameter named granularity factor and the groups are named cluster types. The granularity of this grouping can be determined based on the accuracy of the method used for generating the input list. In the example provided in Table II, the clusters in Table I are grouped in a way that the difference between any pair of cluster sizes in a cluster type is at most 4 nodes, to be referred to as the granularity factor in the rest of this work. For example, the number of clusters for the first cluster type is equal to the sum of the number of clusters with each size in that cluster type which are 1, 2, 0, and 1 in Table I, and 4 for the first cluster type in Table II. It is obvious that Table I is a special case of Table II with a granularity factor of 1. Only the first and last cluster types are shown Table II.

A parameter is assigned to each clusterhead that shows the precise number of nodes that the corresponding cluster should contain. This parameter is named the Range of Advertisement (RoA) which indicates the expected cluster size. The namesake for the RoA is that the number of nodes that join a cluster should be proportional to the distance (range) up to which a clusterhead advertises its membership message. All clusters in a cluster type were assigned the same RoA which is the median of that cluster type's range. For example, the RoA of the five clusters of the fifth cluster type (second row) in Table II is equal to 23.5. A method could be used to lessen the effect of the inaccuracy that may be introduced due to the grouping method mentioned above. The real number that is mentioned here indicates the expected number of clusters for each case. To make it clearer, it can be assumed that the clusters to which each RoA in a cluster type's range is assigned are chosen using a pseudo-random number generator that is based on a uniform distribution.

The computational complexity of the proposed clustering algorithm is an issue to be considered. Based on reported studies on a special case of the proposed clustering algorithm (Kmeans) [McQueen 66], the problem is computationally difficult (NP-hard). With different assumptions, the questions of best case and worst case complexities of the algorithm could be investigated.

5.4 Proposed Clustering Algorithm's Design

In this section, first a basic version of the proposed clustering algorithm, which is based on an advertisement protocol, is explained. The proposed clustering algorithm is then improved with two extensions. The first extension consists of an iterative advertisement protocol with the purpose of improving the clustering process. The second extension introduces a method for assigning the as-yet unassigned nodes to clusters.

5.4.1 Basic Algorithm: Clusterheads and the Advertisement Protocol

In the proposed algorithm, Nch nodes are pseudo-randomly chosen as initial candidate clusterheads from among the nodes in the network. Each clusterhead advertises a message asking its directly-connected neighbors (its four adjoining nodes) to join its cluster. The neighboring nodes that are not themselves clusterheads and receive the advertisement, join the cluster. If a node receives two or more membership messages, it joins the cluster for which the difference between the cluster's number of acquired nodes and the respective clusterhead's RoA is larger. Then the recently-joined nodes advertise membership messages to their own directly-connected neighbors. Ideally, this procedure should continue until every cluster acquires exactly a number of nodes that is equal to its RoA. However, based on simulations (Section 5.5.1), it was determined that the basic algorithm has a number of drawbacks. There might be a number of nodes that are neither clusterheads nor members of a cluster, to be referred to hereafter as isolated nodes. There might be a number of clusters in close proximity to one another in such a way that they cannot grow and accumulate a sufficient number of nodes, i.e., reach their assigned RoAs.

The clusterheads are distributed evenly in a network if they are distributed in a way that all of them are able to acquire their required nodes while staying at the geographical center of their cluster. Uneven distribution of the clusterheads may cause the emergence of entangled isolated nodes which are not connected to any cluster and are trapped among some clusters that have already accumulated a sufficient number of nodes. Some clusters may have constrained surroundings because either they have other clusterheads as their neighbors or they are at the boundary of the rectangular grid network.

In the following subsections, two extensions to the proposed basic clustering algorithm are introduced with the purpose of addressing the above-mentioned issues.

5.4.2 Extension 1: Re-Selection of the Clusterheads

In this extension, the nodes that have been selected as clusterheads by the clustering algorithm are released and the central node of each cluster is chosen as the next candidate clusterhead. If the network is assumed as a two dimensional Cartesian coordinate system, the central node of each cluster is the node whose coordinates' Euclidean distance is closest to the average of the coordinates of all the nodes in the cluster in both x and y axes.

This proposed extended algorithm basically consists of iterations of the basic algorithm. The iterations terminate once the clusterhead locations do not change anymore, i.e., when the sum of the distances between the nodes and their current respective clusterheads and the previous ones in each cluster approaches 0 (the same assumption is made in the Kmeans algorithm [McQueen 66]). Note that 0 is a limit value and, since the network model used in this chapter entails the use of a discrete environment, there may be a case where two or more nodes are qualified for being clusterheads in a cluster and the selection of a clusterhead may switch back and forth between those nodes continually. Therefore, once the locations are changing less than a convergence threshold value, the process is stopped. The convergence threshold is a system parameter whose value is decided based on a trade-off between the algorithms convergence time and the even distribution of the clusterheads.

This extension should provide a better distribution of the clusterheads since it is designed to separate the clusterheads that are so close to each other that they are not able to acquire nodes in all directions. The modified clustering algorithm also places the clusterheads away from the boundary based on the number of nodes in each cluster.

Since the clusterheads are distributed pseudo-randomly in the network, it is possible, despite attempts at re-selection of the clusterheads toward having them away from the boundry, that a number of the clusterheads still get entangled among some other clusterheads and not be able to acquire all of their required nodes. This problem is addressed by pseudo-randomly selecting another not-already-committed node in the network as the clusterhead instead of one of the clusterheads that has not reached a threshold number of nodes in the previous Iteration. The RoA of the previous clusterhead should be assigned to the new one. The threshold value for the number of nodes that each cluster should acquire in each iteration was chosen based on the RoA of that cluster. Choosing threshold values closer to the RoAs of the clusterheads resulted in fewer isolated nodes in the basic algorithm and an undesirably longer convergence time.

Simulation results showed that the problem where a number of nodes may not be members of any cluster still exists. The next extension addresses this issue.

5.4.3 Extension 2: Assigning Isolated Nodes to Clusters

In this extension, two possible alternative improvements are proposed. A comparison of the two alternatives is discussed in more detail in the analysis and simulation section (Subsection 5.5.3).

In the first improvement, before each iteration of re-selecting the clusterheads, the isolated nodes are assigned to the nearest clusterhead. In the situations where there are more than one clusterheads with the same minimum distance to the node, the clusterhead that needs more nodes to reach its specified population size is chosen.

An alternative improvement is introduced in this study. This improvement, to be referred to as the *crawling method*, attaches the isolated nodes to the clusterheads that need more nodes to reach their RoAs, i.e. the neediest clusters. The isolated node should not be added directly to the neediest cluster because it would prevent the protocol from guaranteeing nearest distances between nodes and their corresponding clusterheads. An algorithm is proposed to help the isolated nodes crawl in the network to reach the neediest cluster. The algorithm is outlined below.

1 Each isolated node, marked as the current node, finds the shortest path through the network to the neediest cluster. The shortest path could be found using Dijkstra's shortest path algorithm with the complexity of $O(|E| + |V| \log |V|)$ with E being the set of all links between any pairs of nodes and V being the set of all nodes in the network [Dijkstra 59].

2 In the shortest path, the current node joins the cluster to which the next node on the path belongs.

3 If the next node in the shortest path is not a member of the neediest cluster, set the next node as the current node and go to Step 2.

4 If there is no other isolated node in the network, exit, else go to Step 1.

Table III. An example of calculating the average number of clusters for each cluster size. a) sizes of 5 clusters after the first three executions of the algorithm b) number of clusters of each size after three execution of the algorithm

(a)

Cluster ID	**C1**	**C2**	**C3**	**C4**	**C5**
Cluster sizes after 1st execution	7	7	6	7	9
Cluster sizes after 2nd execution	8	7	9	9	6
Cluster sizes after 3rd execution	9	7	7	7	9

(b)

Cluster Sizes	6	7	8	9
Number of clusters after 1st execution	1	3	0	1
Number of clusters after 2nd execution	1	1	1	2
Number of clusters after 3rd execution	0	3	0	2
Average number of clusters of each size	0.6	2.3	0.3	1.6

After the algorithm runs, the clusterhead re-selection algorithm runs and, as a result, the new clusterheads will be the nodes in the center of the new clusters after the inclusion of the isolated nodes.

Since the clusterheads are initially selected pseudo-randomly, in some cases the algorithm may not converge to minimize the sum of the distances, i.e., the sum of the distances may just keep changing. To deal with this problem, an upperbound value was used for the number of iterations. If within that upperbound number of iterations the algorithm do not converge, the clusterheads are re-selected pseudo-randomly and the algorithm starts anew. The upperbound value could be chosen based on different factors such as the number of nodes or historical data.

5.5 Analysis and Simulation

As mentioned in Section 5.2, the network has a rectangular layout with a hard boundary. The clusterheads are chosen pseudo-randomly and the input list is considered to be the initial distribution of the RoAs (see Subsection 5.3) among the clusters in the network. Netlogo [Wilensky 99] (about 1400 lines of code) and Java (about 1000 lines of code) are used for the simulations in this section.

In this section, the *RoAs' distribution* among the clusterheads is assumed to be as given in Figure 17. The given RoAs' distribution is used for all simulation results in this section. In the example of this section, five cluster types with sizes between 5 and 29 are considered and the granularity factor is 5.

In Figure 17, the number of clusters for each RoA is not a whole number and the reason is explained in 5.3. The results of running the algorithm for 100 different pseudo-randomly generated initial distributions of RoAs among the clusterheads are discussed later in the study (Subsection 5.6.4).

An advertisement protocol was used to assign the nodes to clusters. All clusterheads advertise to take members with the same priority (i.e., all clusterheads advertise at the same time). Based on simulations, it has been observed that it is harder for clusters with larger RoAs to acquire a sufficient number of nodes. To deal with this issue, clusterheads could be non-ascendingly ordered based on their RoAs in order to give the clusterheads with larger RoAs more opportunity to acquire nodes.

To make the results that are provided in this section more reliable, the clustering algorithm was executed *100* times and the average of the results of the executions was used for analysis and depiction. Table III describes how the average of 3 execution of clustering algorithm could be generated. In Table III, 5 clusters are considered in a cluster type with the range of {6, 7, 8, 9}.

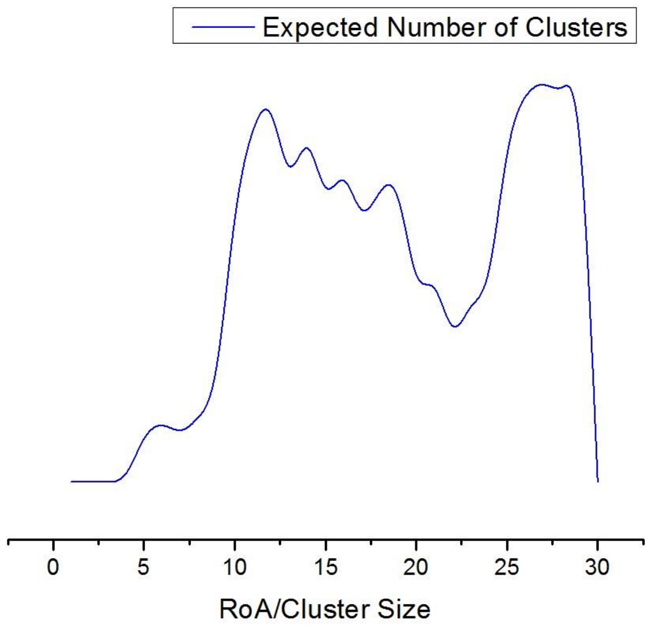


Figure 17. An example of the distribution of the Range of Advertisements (RoA) among clusterheads. RoAs define clusters sizes.

5.5.1 Simulation Results for the Basic Algorithm

The result of the execution of the basic algorithm is depicted in Figure 18. In the basic algorithm, only the advertisement procedure is executed (i.e., the re-selection of the clusters is not applied (extension 1) and the isolated nodes are not taken care of (extension 2)). As it is observable in Figure 18, the number of clusters of sizes 18 and more (except 21) are less than the expected number. The number of clusters of sizes smaller than 17 are more than their expected number. The reason is that a number of the clusters do not acquire a sufficient number of nodes, which is due to the existence of isolated nodes.

Clusterheads are distributed pseudo-randomly therefore a number of them might prevent other clusterheads from acquiring nodes. Also, a number of the clusterheads might fall on the boundary and not be able to advertise in all directions.

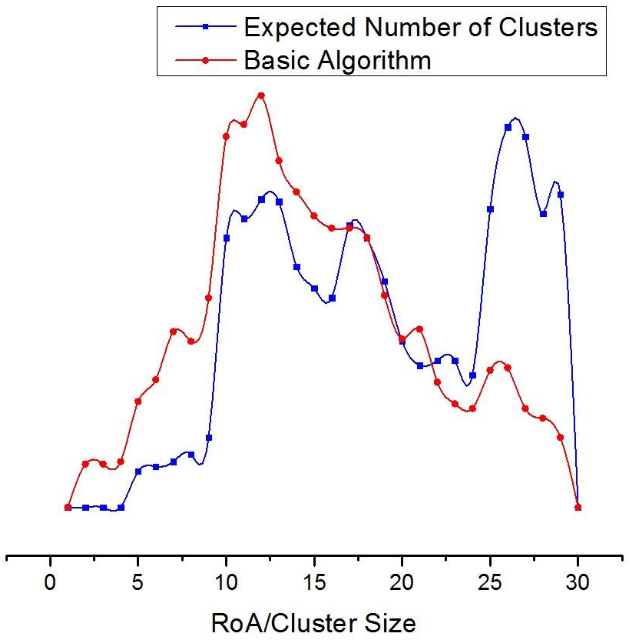


Figure 18. Comparison of the expected number of clusters with each size and the average number of clusters with each size representing the average of the results for 100 executions of the basic algorithm on a given distribution of RoAs

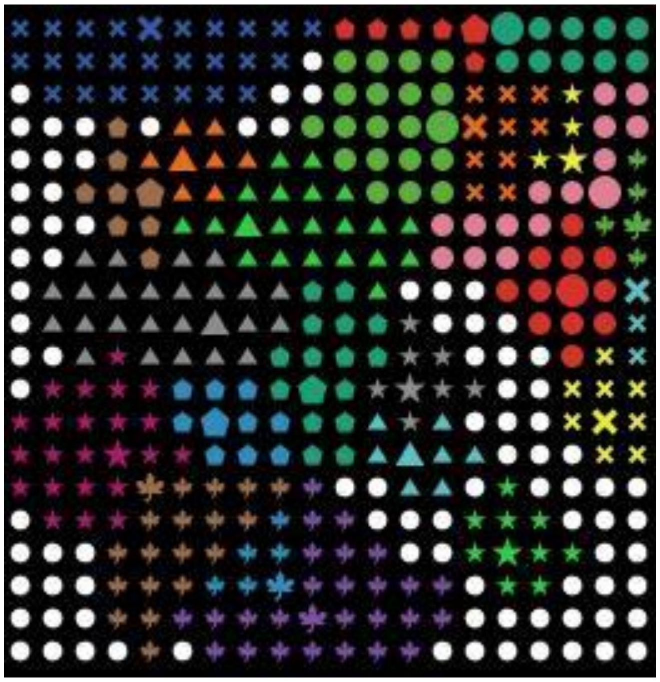


Figure 19. Visual result of the basic algorithm. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes in each cluster

In the example depicted in Figure 19, the distribution of the clusterheads in the network as a result of applying the basic algorithm indicates that several clusterheads may be undesirably close to one another, and a number of them might be on or too close to the boundary.

5.5.2 Simulation Results for Extension 1

Figure 20 shows the simulation result after adding the procedure for re-selecting the clusterheads and running the iterations of the algorithm until the change in the clusterheads' locations became at most one unit for each clusterhead. One unit is the convergence threshold value that was discussed in Subsection 5.4.2.

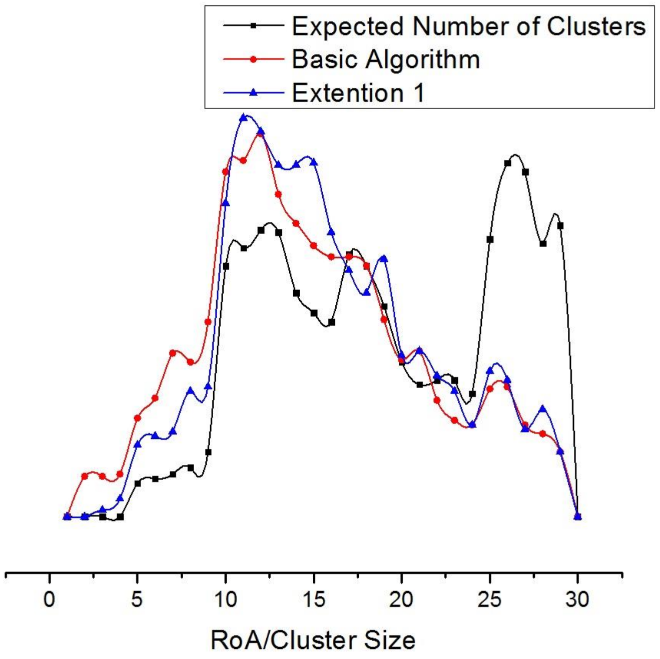


Figure 20. Comparison of the expected number of clusters with each size and the average number of clusters with each cluster size representing the average of the results for 100 executions of the extension 1 algorithm using a given distribution of RoAs

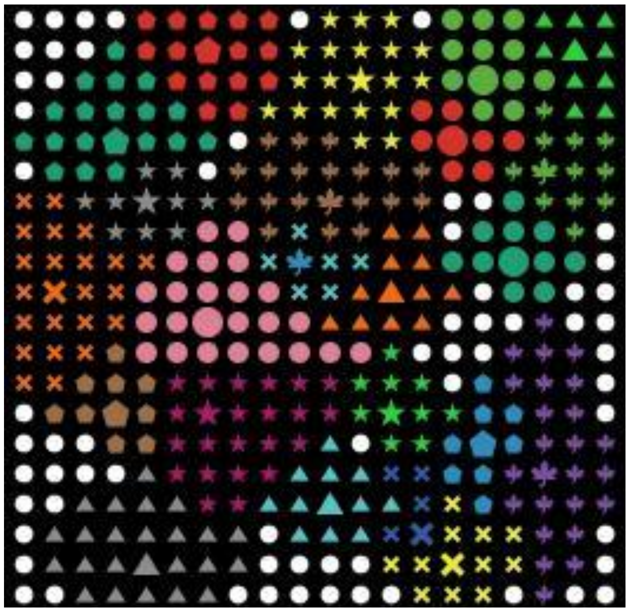


Figure 21. Visual simulation result of extension 1. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes in each cluster

The simulation result of this enhanced version indicated that the number of clusters with different sizes was still not matching the RoAs distribution, and the clusterheads with higher RoAs still could not obtain a sufficient number of nodes to be able to reach their RoAs in spite of all clusterheads' being far enough from the boundary and from each other.

The clusterheads were also close to the centers of their own clusters. Figure 21 depicts the simulation result of this extension.

5.5.3 Simulation Results for Extension 2

To choose between the two methods that were proposed in Section 5.4.3 for attaching isolated nodes to clusters, both methods were implemented and the results were compared. The two methods are: 1) assigning isolated nodes to the nearest cluster, 2) crawling method. The distribution of the resulting clusters in the network was compared with the distribution of the RoAs among clusterheads. It was

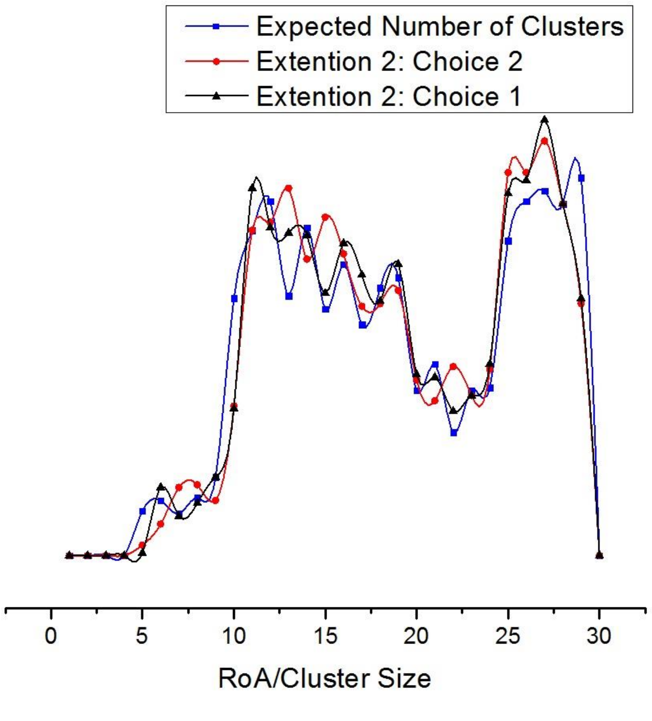


Figure 22. Comparison of the expected number of clusters with each size and the average number of clusters with each size representing the average of the simulation results for 100 executions of extension 2 algorithm on a given distribution of RoAs

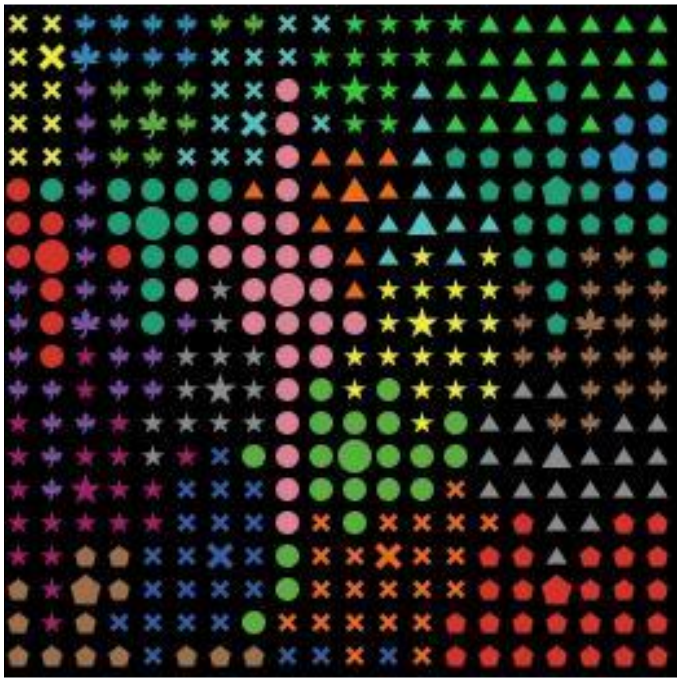


Figure 23. Visual simulation result of extension 2 (crawling method). The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes of each cluster

found that the convergence time for the first method was less than the convergence time for the second method. However, the distribution of clusters with different sizes did not match the distribution of RoAs among the clusterheads for the first method. A better distribution of the clusterheads might be obtained by using an optimization approach, e.g., by using genetic algorithms. This approach is relegated to future work.

There are two options for attaching the isolated nodes to clusters using the crawling method that was introduced in Subsection 5.4.3. One option is to attach the isolated nodes before each re-selection of the clusterheads. The other option is to execute the crawling method once at the end of the last iteration of reclustering. Simulations showed that the second option worked better in that the clusterheads were distributed more evenly and the algorithm converged faster. The later convergence time of the first option might be because of a number of nodes that were not directly attached to their clusters thus causing the crawling procedure to be executed several more times and making the clusters unbalanced. As a result of unbalanced clusters, more iterations were needed in re-selecting the clusterheads in order to stabilize the location of the clusterheads. In the simulation runs, the convergences time for the first option and the second option were *33.73* and *2.34* iterations, respectively. These two numbers are the average number of iterations that each method needs to converge. They are the average number of iterations for *100* executions of the algorithm. The result for the crawling method is given in Figure 22. The results of the algorithm is depicted in Figure 23.

In Figure 21, compared to extension 1 in Figure 19, the number of clusters with each size is closer to the expected number for that cluster size. Only in a few situations, where the expected number of clusters changes from one cluster type to another, was a large difference observed between the number of clusters and the expected number of clusters (e.g., from cluster size 9 to size 10). Depending on the magnitude of change of the expected number of clusters in two adjacent cluster types, the first or the last cluster size in the cluster types' ranges might have more or fewer clusters than what was expected. This inequality was compensated for by other clusters in the cluster type.

5.5.4 Simulation Results for Different Distributions of RoA's Among Clusterheads

In the previous subsections of this section, the simulation results presented were based on a specific

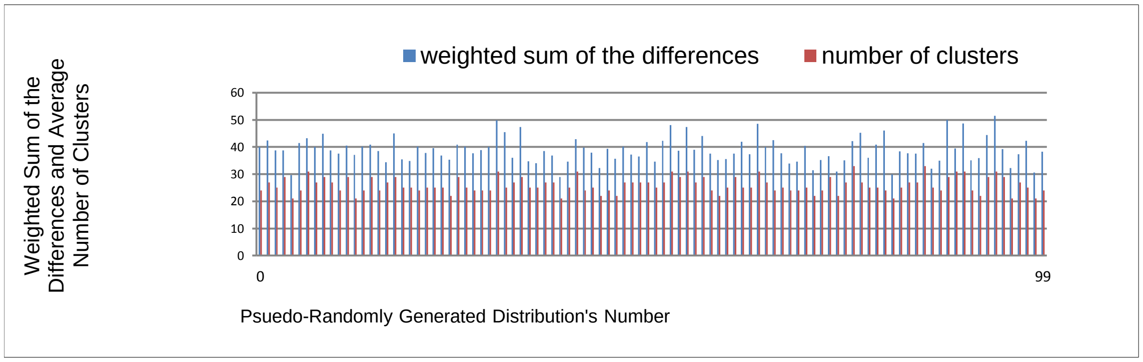


Figure 24. Results for execution of the proposed algorithm for 100 pseudo-randomly generated distributions

distribution of the RoAs. In this subsection, it is shown that the algorithm worked well for 100 different pseudo-randomly generated distributions (i.e., samples from the same statistical distribution).

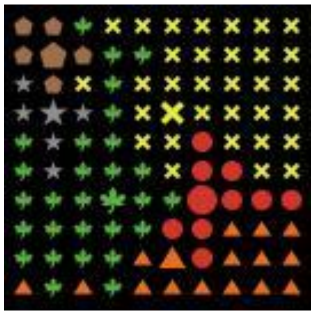
The difference between the number of nodes that a cluster acquires and that cluster's RoA was calculated for all clusters. This number is referred to as the number of missing nodes of a cluster. Note that if the magnitude of the number of nodes acquired by a cluster is greater than its RoA, the magnitude of the number of missing nodes is considered to be a negative number equal to the difference between the number of acquired nodes and the RoA. The following equation calculates the sum of the differences in all clusters for the pseudo-randomly generated distributions.

$$d = \sum_{i=0}^{N_{CH}} |d_i| \qquad (15)$$

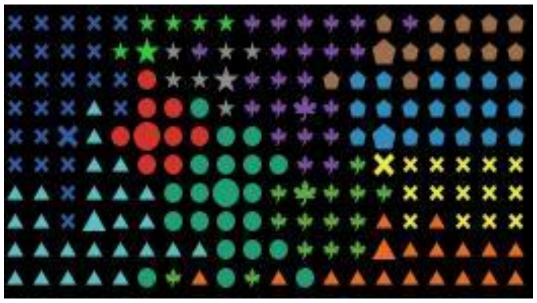
In Equation (15), d is the sum of the differences, di is the number of missing nodes in cluster i, |di| is the absolute value of di which is an integer between 0 and the RoA of the respective cluster, and N_{CH} is the total number of clusterheads in the network.

The ideal case for the result is for the sum of the differences to be equal to 0 or the number of nodes in the clusters to be exactly equal to their respective RoAs.

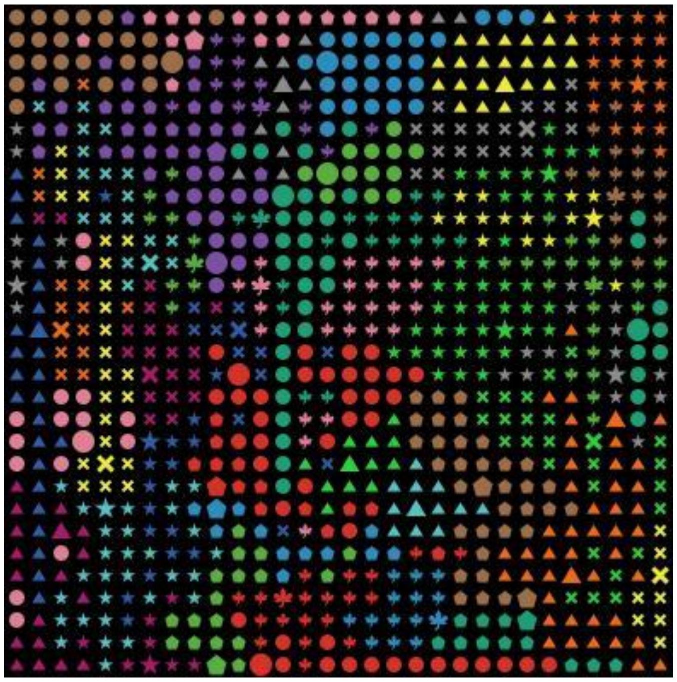
The sum of the differences contains information about the total number of missing nodes in all clusters although it does not provide any information about how the missing nodes are distributed among the clusters. For example, consider the situation where N_{CH} is 10 and the total number of missing nodes in the clustered network is 20. It is possible that the number of nodes in cluster j be in the range [RoAj - 2, RoAj+2], where RoAj is the RoA of cluster j, with $0 < j < N_{CH}$. It is also possible that one cluster's missing nodes be 20 and all other clusters have exactly as many nodes as their RoA. Equation (16) below, in addition to the total number of missing nodes, reflects another parameter that is related to how the missing nodes are distributed in the clusters of the network. In this equation, in each cluster more weight is given to the second missing node than to the first one, still more weight to the third



(a)



(b)



(c)

Figure 25. Results for three network sizes: (a) 10 × 10 nodes (b) 10 × 20 nodes (c) 30 × 30 nodes. The nodes of a cluster have the same color and shape, and the clusterheads are the larger nodes with the same color and shape as the regular nodes of each cluster

missing node than the second one, and so on for the rest of the missing nodes. A geometric progression is utilized to reflect the weights.

$$d = \sum_{i=0}^{N_{CH}} a^{|d_i|} \tag{16}$$

In equation (16), the value of a can be decided based on the criticality of not reaching RoAs in the clusters, i.e., not acquiring enough nodes. In the simulation runs reported in this work a = 2 was used. For instance, assume the situation where there are three clusters in the network and three nodes are missing. If three clusters are missing one node each, the sum of the differences would be 3, but if one cluster is missing three nodes and the others are not missing any node, the sum of the differences would be 23.

Average simulation results for 100 executions of the proposed algorithm for any of the 100 pseudo-randomly generated distributions is shown in Figure 24. It appears that almost all the averages of the sum of the differences for all distributions in Figure 24 is between 30 and 50, and the number of clusters is in the range 20 to 30. In the situation where all the missing nodes were from only one cluster, since the sum of the differences is at most 50, the average of the total number of missing nodes was less than 6 based on Equation (16) which was provided earlier in this section.

If each cluster misses on average at most 2 nodes, a total of fewer than 25 nodes would be unassigned in the network. Considering any of the situations mentioned above, the result of executing the proposed algorithm over 100 different RoA distributions was close to the RoA distribution that was used in the previous sections of the proposed clustering algorithm in this dissertation work.

5.6 Size Scalability

The algorithm was tested based on several options for the number of rows and columns of the rectangular grid network. Because of page number limitation, only the results for three of the cases are shown in Figure 25.

5.7 Performance Analysis

A simulator was implemented to compare the performance of the proposed algorithm with Kmeans [McQueen 66]. The simulator generates a number of applications in each cycle and sets (a) the average number of nodes that each application requires during its lifetime, i.e., its Average Degree of Concurrency (ADoC), and (b) the processing time for each application pseudo-randomly in specific ranges that are input parameters to the simulator. The effect of different lower and upper bounds for these ranges was investigated to avoid accidental results. The applications were placed in a queue to be assigned to the "best-fit" clusters based on the applications' ADoCs and the number of nodes in the available network cluster.

First, the clustering algorithm introduced in this study was compared with Kmeans with the same number of clusters for 1,000,000 applications. Figure 26 illustrates the results of the simulation, where the parallelism parameter is defined as the expected processing time over the actual processing time for applications in an ideal situation. The ideal situation is having as many processors (nodes) as requested for any application at any time. As shown in Figure 26, the new clustering algorithm provides better response time as well as more parallelism compared to Kmeans. Although the overall requested processing time for the new clustering algorithm is more than Kmeans, the total clock cycles used by the new clustering algorithm is less than Kmeans. Based on these results, the proposed clustering algorithm outperforms Kmeans with identical number of clusters in terms of parallelism, average response time, and total clock cycles. Note that the results are for a relatively small period of time (around 3 minutes), the differences between the two algorithms increase as the sampling time increases.

Both of the algorithms are used for another experiment in which applications were generated for 300,000 milliseconds. The two algorithms were compared based on the total number of idle nodes throughout the simulation. Figure 27 depicts the results for this experiment where the effect of changing the maximum bound for ADoC was investigated as well. It is shown that for the proposed clustering

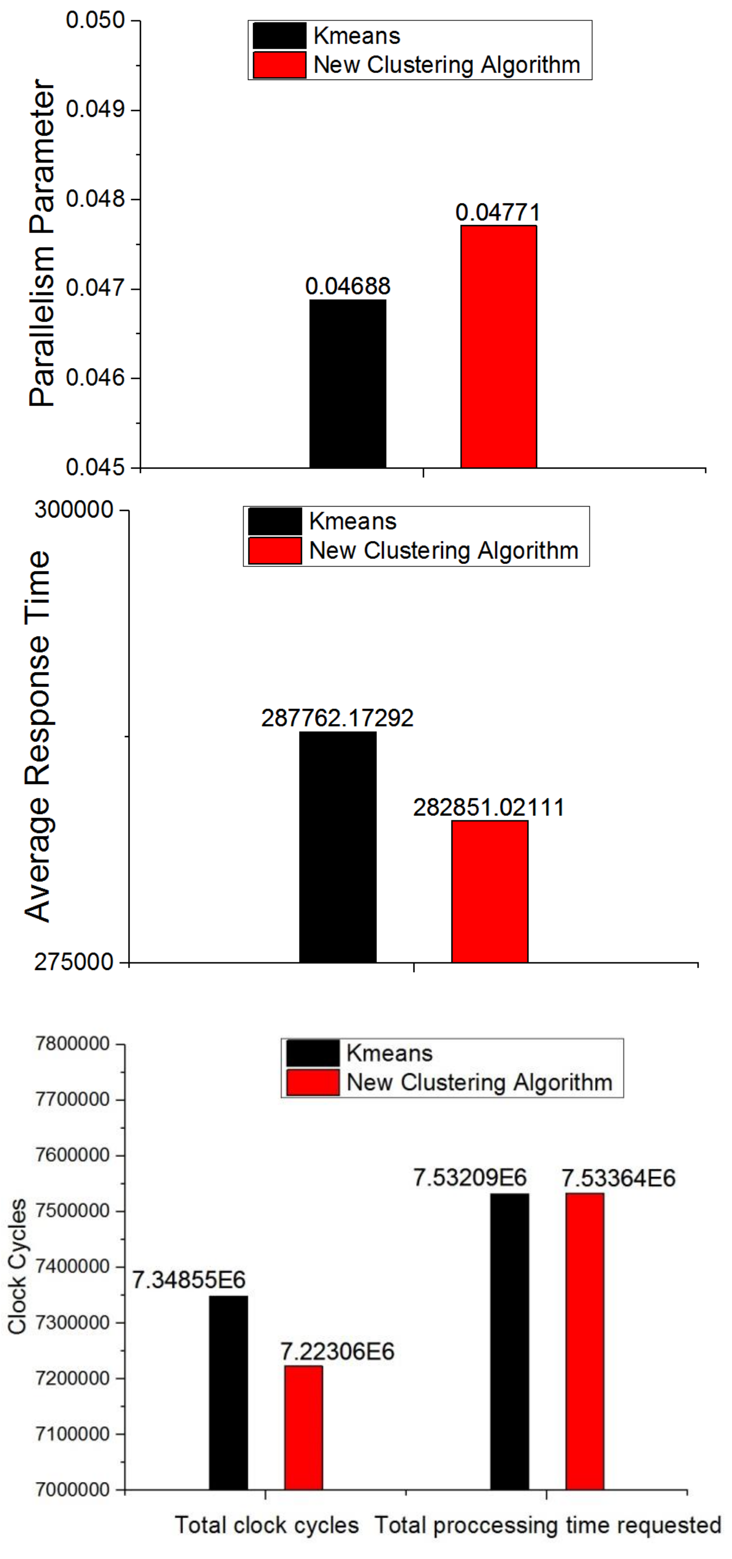


Figure 26. Comparison between Kmeans and the proposed algorithm for a duration of 300,000 milliseconds. The differences between the two algorithms' performance appear to increase with larger time periods

algorithm the total number of idle nodes is always less than Kmeans, meaning that the proposed clustering algorithm is using resources more efficiently.

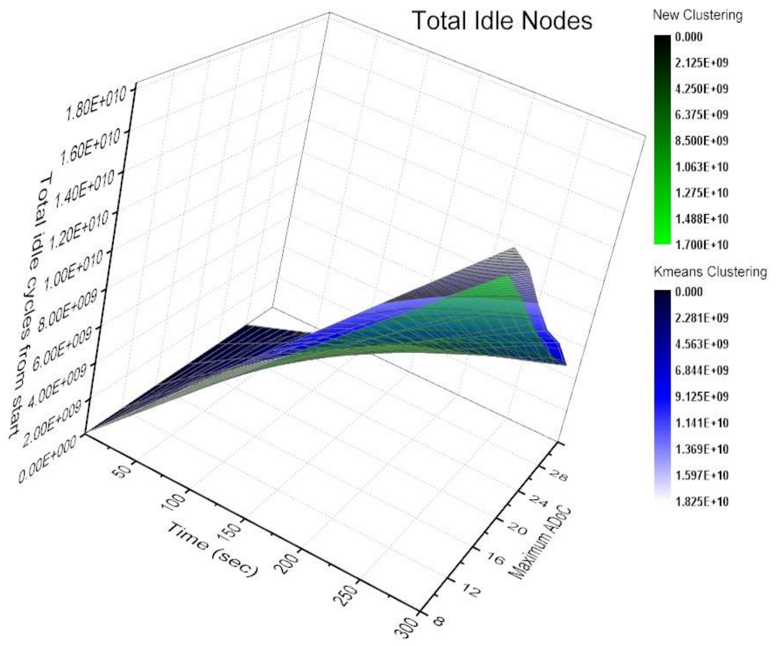


Figure 27. The total number of idle nodes for 300,000 milliseconds

CHAPTER VI

SUMMARY AND FUTURE WORK

6.1 Summary

In Chapter I, the main objective of this thesis was presented. Chapter II provided background knowledge on different aspects of security, Wireless Sensor Networks, and clustering. The Threshold Secret Threshold Secret Sharing Technique, Eshnenaur and Gligor’s scheme, collaborative and parallel processing in energy-sensitive networks, and characteristics of popular clustering algorithms were also discussed in Chapter II.

Chapter III contained the description of the symmetric threshold multipath scheme. In this chapter, a new scheme for highly secure communication in a network was introduced. The proposed scheme is a symmetric key management technique with secure online key distribution. The strength of the proposed techniques was in the enhanced security of the distribution of the symmetric keys that were used to encrypt messages. The symmetric keys were generated for each message and, in order to provide further security, the keys/secrets were split using the threshold secret sharing technique. A multipath approach along with a pre-distributed symmetric key management scheme were utilized to enhance the security of transferring the shares of the secrets to their respective destinations. Based on the analysis provided in Chapter III, confidentiality (i.e., access to the confidential data being restricted only to the authenticated entities) was assured to an acceptable level compared to related studies.

A reasonable level of dependability (i.e., being resilient to compromise and fault tolerant) was also provided by the proposed scheme. In the proposed technique, the shares of the secrets were distributed through different paths, and the reconstruction of the secrets depends on the reception of the shares. The level of dependability could be determined based on the threshold secret sharing technique's parameters and the redundancy they provide for the shares. The proposed scheme reduces the space cost compared to similar studies, which is one of the benefits of the online nature of the proposed key management in this work.

In Chapter IV, a new scheme for storing collected data in Wireless Sensor Networks was provided. The proposed cluster-based scheme worked with a threshold secret sharing technique and a symmetric key-management scheme. Throughout the chapter, several security and performance issues were addressed. In the proposed scheme, the sensed data are transmitted to CHs in a secure manner and disseminated in the respective clusters using the threshold secret sharing technique. Moreover, an adaptive threshold technique was introduced that operate based on historical data and a multicasting protocol. The effect of several system parameters on energy consumption and overhead was investigated using simulations. The results of simulation indicated that the proposed schemes performed better than the basic scheme.

The proposed protocol is lightweight and suitable for WSNs while providing security for stored data in the network.

In Chapter V, a new clustering algorithm was proposed to support Wireless Sensor Networks. The proposed algorithm groups the nodes of a given network into clusters of different sizes. The proposed clustering algorithm was built upon two quantitative aspects of the underlying network. The first aspect is the required number of nodes in each cluster that could be different for each cluster. The number of nodes in each cluster is aimed to match the size expected for that cluster. In this part of the work, the expected size for each cluster was determined based on a pseudo-randomly generated input. The second quantitative aspect of the underlying network was the consideration of the sum of the distances between the nodes in each cluster and that cluster's corresponding clusterhead. Clusterheads were chosen based on their locations, which means that they were chosen in a way that the sum of the distances between nodes and their clusterhead was minimized. In this part of the study, first a basic algorithm was introduced and which was subsequently extended in two stages.

The proposed algorithm was simulated based on pseudo-randomly generated expected number of clusters with different sizes. The results showed that the number of clusters with each size in the clustered network closely correspond to the given expected numbers. The proposed algorithm was executed based on 100 different pseudo-randomly generated "expected number of clusters" with different sizes to prototypically evaluate the applicability of the algorithm for more general cases. The scalability of the proposed algorithm was discussed in Subsection 5.6 with the observation that the algorithm would work for different network sizes.

To the best of our knowledge, the proposed clustering algorithm is the first clustering algorithm that can provide an infrastructure to make task scheduling efficient and provide a solution to the problem of task scheduling without needing to invoke time consuming task scheduling optimization methods.

6.2 Future Work

An extension of this study, a predictive approach that adapts the proposed clustering algorithm to the incoming jobs based on historical data, is an ongoing research thread. The proposed clustering algorithm reduces the distance between the nodes and the clusterheads and thus it is expected that to outperform Kmeans in terms of energy efficiency.

The proposed clustering algorithm could be applicable to several other fields such as Network on Chips and cloud computing. A detailed study in these areas is part of the future work for this research.

REFERENCES

[Abbasi and Younis 07] A**.** A. Abbasi and M. Younis, “A Survey on Clustering Algorithms for Wireless Sensor Networks”, *The Journal of Computer Communications*, Vol. 30, No. 14-15, pp. 2826-2841, October 2007.

[Amis et al. 00] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, “Max-Min D-Cluster Formation in Wireless Ad Hoc Networks”, *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM),* No. 1, pp. 32-41, Tel Aviv, Israel, March 2000.

[Backer and Jain 81] E. Backer and A. K. Jain, “A Clustering Performance Measure Based on Fuzzy Set Decomposition”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-3, No. 1, pp. 66-75, January 1981.

[Bandyopadhyay and Coyle 03] S. Bandyopadhyay and E. Coyle, “An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks”, *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communication (INFOCOM’03)*, Vol. 3, pp. 1713-1723, San Francisco, CA, April 2003.

[Barker and Roginsky 10] E. Barker and A. Roginsky, Recommendation for the Transitioning of Cryptographic Algorithms and Key Lengths, Draft NIST Special Publication 800-131, U.S. Department of Commerce, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, June 2010.

[Boloorchi 11] A. T. Boloorchi, *An Adaptive Clustering Algorithm for Wireless Sensor Networks*, M.S. Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, December 2011.

[Boloorchi and Samadzadeh 13] A. T. Boloorchi and M. H. Samadzadeh, “Energy-Efficient and Secure In-Network Storage and Retrieval for Wsns: An Adaptive Approach”, *The Journal of Supercomputing (Springer)*, Vol. 65, No. 2, pp. 961-977, August 2013.

[Boloorchi et al. 14a] A. T. Boloorchi, M. H. Samadzadeh, and T. Chen, “Symmetric Threshold Multipath (STM): An Online Symmetric Key Management Scheme”, *Information Sciences (Elsevier),* Vol. 268, pp. 489-504, June 2014.

[Boloorchi et al. 14b] A. T. Boloorchi, M. H. Samadzadeh, and N. Rahnavard, “A New Parallelism-Capable Clustering Algorithm for Wireless Sensor Networks”, *14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid’14)*, pp. 660-669, Chicago, IL, May 2014.

[Booch et al. 05] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide*, The second edition (Addison-Wesley Object Technology Series) Addison-Wesley Professional, 2005.

[Boukerche 08] A. Boukerche, *Algorithms and Protocols for Wireless Sensor Networks*, Wiley-IEEE Press, Hoboken, NJ, 2008.

[Casella and Berger 90] George Casella and R. L. Berger , *Statistical Inference,* Duxbury Press, Pacific Grove, 1990.

[Chan et al. 03] H. Chan, A. Perrig, and D. Song, "Random Key Pre-distribution Schemes for Sensor Networks", *Proceedings of IEEE Symposium on Security and Privacy,* pp. 197-213, Berkeley, CA, May 2003.

[Chan et al. 04] H. Chan, A. Perrig, and D. Song, "Key Distribution Techniques for Sensor Networks", a chapter in a book titled *Wireless Sensor Networks,* C. S. Raghavendra, K. Sivalingam, T. Znati (Eds.) Springer, Kluwer Academic Publishers, pp. 277-303, Norwell, MA, 2004.

[Chen et al. 11] C. Chen, W. Guo, and G. Chen, "Real-Time Adaptive Task Allocation Algorithm with Parallel Dynamic Coalition in Wireless Sensor Networks", *Knowledge Engineering and Management*, Vol. 123, pp. 25-32, December 2011.

[Cordeiro et al. 10] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J. Vincent, and F. Wagner, "Random Graph Generation for Scheduling Simulations", *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools '10),* No. 60, pp. 1-10, Brussels, Belgium, March 2010.

[Crosby and Pissinou 07] G. V. Crosby and N. Pissinou, "Cluster-Based Reputation and Trust for Wireless Sensor Networks", *Proceedings of the Consumer Communications and Networking Conference* (*CCNC'07),* pp. 604-608, Las Vegas, NV, January 2007.

[Crossbow 07] Crossbow Technology Inc., *imote2 Hardware Reference Manual,* September 2007.

[Dick et al. 98] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task Graphs For Free", *Proceedings of the Sixth International Workshop on Hardware/Software Codesign,* pp. 97-101, Washington, DC, March 1998.

[Dijkstra 59] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, Vol. 1, pp. 269-271, 1959.

[Du et al. 04] W. Du, J. Deng, Y. Han, S. Chen, and P. Varshney, "A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge", *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, Vol. 1, pp. 586-597, Hong Kong, PR China, March 2004.

[Du et al. 07] X. Du, Y. Xiao, M. Guizani, and H. Chen, "An Effective Key Management Scheme for Heterogeneous Sensor Networks", *The Journal of ad hoc Networks,* Vol. 5, No. 1, pp. 24-34, January 2007.

[Deng et al. 04] H. Deng, A. Mukherjee, and D. P. Agrawal, "Threshold and Identity-Based Key Management and Authentication for Wireless ad hoc Networks", *Proceedings of the*

International Conference on Information Technology: Coding and Computing (ITCC'04), Vol. 2, pp. 107-111, Las Vegas, NV, April 2004.

[Diffie and Hellman 76] W. Diffie and M. Hellman, “New Directions in Cryptography”, *IEEE Transactions on Information Theory,* Vol. 22, No. 6, pp. 644-654, November 1976.

[Eades and Sugiyama 91] P. Eades and K. Sugiyama, “How to Draw a Directed Graph”, *Journal of Information Processing,* Vol. 13, No. 4, pp. 424-437, April 1991.

[Edalat et al. 09] N. Edalat, W. Xiao, C. Tham, E. Keikha, and L. Ong, “A Price-Based Adaptive Task Allocation for Wireless Sensor Network”, *Proceedings of the Sixth IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS '09),* pp. 888-893, Las Vegas, NV, October 2009.

[Erdos and Renyi 59] P. Erdos and A. Renyi, “On Random Graphs I”, *Publicationes Mathematicae Debrecen,* Vol. 6, pp. 290-297, 1959.

[Eschenauer and Gligor 02] L. Eschenauer and V. D. Gligor, “A Key-Management Scheme for Distributed Sensor Networks”, *Proceedings of the 9th ACM Conference on Computer and Communications Security,* pp. 41-47, Washington, DC, November 2002.

[Feitelson 09] D. Feitelson, “The Parallel Workloads Archive Logs”, URL:
http://www.cs.huji.ac.il/labs/parallel/workload/logs.html,
date created: October 2009, date accessed: October 2012.

[Gangishetti et al. 07] R. Gangishetti, M. C. Gorantla, M. L. Das, and A. Saxena, ”Threshold Key Issuing in Identity-Based Cryptosystems”, *The Journal of Computer Standard Interfaces,* Vol. 29, No. 2, pp. 260-264, February 2007.

[Garay et al. 02] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin, “Secure Distributed Storage and Retrieval”, *The Journal of Theoretical Computer Science,* Vol. 243, No. 1-2, pp. 363-389, July 2002.

[Gentry 03] C. Gentry, “Certificate-Based Encryption and the Certificate Revocation Problem”, *Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT’03),* pp. 272-293, Warsaw, Poland, May 2003.

[Grama et al. 03] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing,* Second Edition, Addison Wesley, Boston, MA, 2003.

[Gupta and Shrivastava 13] N. Gupta and M. Shrivastava, “Securing Routing Protocol by Distributed Key Management and Threshold Cryptography in Mobile ad hoc Network”, *International Journal of Advanced Computer Research,* Vol. 3, No. 9, pp. 13-18, March 2013.

[Heinzelman et al. 00] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy- Efficient Communication Protocol for Wireless Microsensor Networks”, *Proceedings of the Hawaii International Conference on System Sciences (HICSS’00),* Vol. 2, pp. 1-10, Wailea Maui, HI, January 2000.

[Ibriq and Mahgoub 12] J. Ibriq and I. Mahgoub, HIKES: Hierarchical Key Establishment Scheme for Wireless Sensor Networks, *International Journal of Communication Systems*, November 2012.

[Ito et al. 05] T. Ito, H. Ohta, N. Matsuda, and T. Yoneda, "A Key Pre-distribution Scheme for Secure Sensor Networks using Probability Density Function of Node Deployment", *Proceedings of the 3rd ACM Workshop on Security of ad hoc and Sensor Networks (SASN '05)* , pp. 69-75, Alexandria, VA, November 2005.

[Jøsang and Ismail 02] A. Jøsang and R. Ismail, "The Beta Reputation System", *Proceedings of the 15th Bled Electronic Commerce Conference*, pp. 324-337, Bled, Slovenia, June 2002.

[Kumar et al. 12] A. Kumar, A. Aggarwal, and Charu, "Efficient Hierarchical Threshold Symmetric Group Key Management Protocol for Mobile ad hoc Networks", *Communications in Computer and Information Science*, Vol. 306, pp. 335-346, 2012.

[Lu et al. 09] Y. Lu, J. Li, and J. Xiao, "Threshold Certificate-Based Encryption", *The Journal of Software*, Vol. 4, No. 3, pp. 210-217, May 2009.

[Lu et al. 12] Y. F. Lu, C. F. Kuo, and A. C. Pang, "A Novel Key Management Scheme for Wireless Embedded Systems", *SIGAPP Applied Computing Review*, Vol. 12, No. 1, pp. 50-59, April 2012.

[McQueen 66] J. B. McQueen, "Some Methods for Classification and Analysis of Multivariate Observations", *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1, pp. 281-297, Berkeley, CA, January 1966.

[Meng and Li 12] X. Meng and Y. Li, "A Verifiable Dynamic Threshold Key Management Scheme Based on Bilinear Pairing Without a Trusted Party in Mobile ad hoc Network", *IEEE International Conference on Automation and Logistics (ICAL)*, pp. 315-320, Zhengzhou, China, August 2012.

[NIST 08] NIST - National Institute of Standards and Technology, Information Technology Laboratory, *Secure Hash Standard (SHS), Federal Information Processing Standards, FIPS PUB* 180-3, Gaithersburg, MD, October 2008.

[Ogata and Kurosawa 96] W. Ogata and K. Kurosawa, "Optimum Secret Sharing Secure Against Cheating", *Lecture Notes in Computer Sience: Advances in Cryptography (Proceedings of EUROCRYPT'96)*, pp. 200-211, Vol. 1070, Springer-Berlin, Heidelberg, Germany, 1996.

[Pissinou and Crosby 07] N. Pissinou and G. V. Crosby, "Cluster-Based Reputation and Trust for Wireless Sensor Networks", *Proceedings of the 4th IEEE Consumer Communications and Networking Conference*, pp. 604-608 Las Vegas, NV, May 2007.

[Qian 12] S. Qian, "A Novel Key Pre-distribution for Wireless Sensor Networks", *Physics Procedia*, Vol. 25, pp. 2183-2189, April 2012.

[Reed and Solomon 60] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields", *The Journal of the Society for Industrial and Applied Mathematics (SIAM)*, Vol. 8, No. 2, pp. 300-304, June 1960.

[Ren et al. 08] W. Ren, Y. Ren, and H. Zhang, "HybridS: A Scheme for Secure Distributed Data Storage in WSNs", *Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC '08),*Vol. 2, pp. 318-323, Shanghai, China, December 2008.

[Ruan et al. 11] N. Ruan, T. Nishide, and Y. Hori, "Threshold ElGamal-Based Key Management Scheme for Distributed RSUs in VANET", *International Conference on Selected Topics in Mobile and Wireless Networking (iCOST),* pp. 133-138, Shanghai, China, October 2011.

[Ruan et al. 12] N. Ruan, T. Nishide, and Y. Hori, "Elliptic Curve ElGamal Threshold-Based Key Management Scheme Against Compromise of Distributed RSUs *for* VANETs", *Journal of Information Processing*, Vol. 20, No. 4, pp. 846-853, October 2012.

[Shamir 79] A. Shamir, "How to Share a Secret", *Commununications of the ACM*, Vol. 22, No. 11, pp. 612-613, November 1979.

[Shamir 84] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes", *Proceedings of Conference on Advances in Cryptology (CRYPTO'84)*, pp. 47-53, Santa Barbara, CA, August 1984.

[Stallings 10] W. Stallings, *Network Security Essentials: Applications and Standards*, Prentice Hall PTR, Upper Saddle River, NJ, 2010.

[Subramanian et al. 07] N. Subramanian, C. Yang , and W . Zhang, "Securing Distributed Data Storage and Retrieval in Sensor Networks", *The Journal of Pervasive Mobile Computing*, Vol. 3, No. 6, pp. 659-676, December 2007.

[Sugiyama et al. 81] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for Visual Understanding of Hierarchical Systems", *IEEE Transaction on Systems, Man, and Cybernetics*, Vol. 11, No. 2, pp. 109-125, February 1981.

[Teo and Tan 05] J.C.M. Teo and C. H. Tan, "Energy-Efficient and Scalable Group Key Agreement for Large ad hoc Networks", *Proceedings the 2nd ACM International Workshop on Performance Evaluation of Wireless ad hoc, Sensor, and Ubiquitous Networks,* pp. 114-121, Quebec, Canada, October 2005.

[Tian and Ekici 07] Y. Tian and E. Ekici, "Cross-Layer Collaborative In-Network Processing in Multihop Wireless Sensor Networks", *IEEE Transactions on Mobile Computing*, Vol. 6, No. 3, pp. 297-310, March 2007.

[Tobita and Kasahara 02] T. Tobita and H. Kasahara, "A Standard Task Graph Set for Fair Evaluation of Multiprocessor Scheduling Algorithms", *Journal of Scheduling*, Vol. 5, No. 5, pp. 379-394, September 2002.

[van Ham 05] Frank van Ham, *Interactive Visualization of Large Graphs*, Ph.D. Thesis, Advanced School for Computing and Imaging, Eindhoven University of Technology, Eindhoven, Netherlands, November 2005.

[Wu et al. 07] B. Wu, J. Wu, E. B. Fernandez, M. Ilyas, and S. Magliveras, "Secure and Efficient Key Management in Mobile ad hoc Networks", *Journal of Network and Computer Applications,*Vol. 30, No. 3, pp. 937-954, August 2007.

[Xiong and Gong 11] W. A. Xiong and Y. H. Gong, "Secure and Highly Efficient Three Level Key Management Scheme for MANET", *WSEAS Transactions on Computers*, Vol. 10, No. 1, pp. 6-15, January 2011.

[Xiao et al. 09] W. Xiao, S. Low, C. Tham, and S. Das, “Prediction-Based Energy-Efficient Task Allocation for Delay-Constrained Wireless Sensor Networks”, *Proceedings of the Sixth Annual IEEE Communications Society Conference on Sensor, Mesh, and ad hoc Communications and Networks Workshop*, pp. 1-3, Rome, Italy, June 2009.

[Xu and Wunsch 05] R. Xu and D. Wunsch, “Survey of Clustering Algorithms”, *IEEE Transactions on Neural Networks,* Vol. 16, No. 3, pp. 645-678, May 2005.

[Younis and Fahmy 04] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for ad hoc Sensor Networks", *IEEE Transactions on Mobile Computing*, Vol. 3, No. 4, pp. 366-379, October 2004.

[Yu and Prasanna 05] Y. Yu and V. K. Prasanna, “Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks”, *Mobile Networks and Applications*, Vol. 10, No. 1-2, pp. 115-131, February 2005.

[Zhang et al. 06] W. Zhang, S. K. Das, and L. Yonghe, “A Trust Based Framework for Secure Data Aggregation in Wireless Sensor Networks”, *Proceedings of the Third Conference on Sensor and Ad Hoc Communications and Networks (SECON '06),* Vol. 1, pp. 60-69, Reston, VA, September 2006.

[Zhou and Haas 99] L. Zhou and Z. Haas, “Securing ad hoc Networks”, *IEEE Network Magazine* Vol. 13, No. 6, pp. 24-30, November/December 1999.

APPENDICES

APPENDIX A

GLOSSARY

Asymmetric Key Algorithm	Cryptographic algorithms that require two separate keys, one of which is *secret* (or *private*) and the other one is *public*.
Average Degree of Concurrency	ADoC is defined as the average of the Degree of Concurrencies (DoC) of all layers of the task graph, i.e., the sum of the DoCs at each layer of an application's task-dependency graph divided by the height of the graph.
Base Station	The nodes that are sinks to which the transferred data from the network are destined and which usually have higher computational and communicational capabilities.
Degree of Concurrency	DoC is defined as the number of tasks that can be executed concurrently at a layer of an application's task-dependency graph.
Directed Acyclic Graph	A DAG is a graph that has no cycles that could be used for a task dependency graph in which the tasks of an application are represented by nodes and the dependencies among tasks are shown by arrows drawn between tasks [Grama 03].
Eschenauer and Gligor Scheme	The EG scheme is a probabilistic scheme in which there is a key pool, the nodes draw keys from that pool, and the keys are subsequently put in their respective key rings where a key ring is the set of keys that a node draws from the pool.

LEACH	One of the popular clustering algorithms for Wireless Sensor Networks is Low Energy Adaptive Clustering Hierarchy (LEACH) [Heinzelman et al. 00]. In LEACH, all sensors in the network are homogeneous and energy constrained.
MEMS	Micro-Electro-Mechanical Systems, very small mechanical devices working with electricity.
Netlogo	A multi-agent programmable modelling environment for network research [Wilensky 99].
NoC	Network-on-Chips is a new approach for designing the communication subsystem of a System-on-Chips (SoC).
SoC	System-on-Chips refers to integrating all components of a computer or other electronic systems into a single integrated circuit (chip).
Symmetric Key Algorithm	Cryptography algorithms that use the same cryptographic keys for both encryption of the plaintext and decryption of the ciphertext.
Threshold Secret Sharing	A cryptographic technique that splits a secret into n shares in a way that with fewer than m shares, $m \leq n$, no information would be disclosed about the secret. The secret could be reconstructed with m or more shares.
WSNs	Wireless Sensor Networks, networks that consist of distributed sensors to monitor physical or environmental conditions such as sound, temperature, and moisture.

APPENDIX B

SOURCE CODE

This appendix contains the source code for the simulation part of Chapter V [Boloorchi 11].

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; AN ADAPTIVE CLUSTERING ALGORITHM FOR WIRELESS SENSOR NETWORKS
;; Author Alireza Boloorchi Tabrizi
;; Comments start with a semicolon ‘;’.
;; ;; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
extensions [array table]

; Declaring global variables

globals [distances mostLikelyRoA RoA numberOfNeededClusters  expectedValues
probabilityOfRoA cluster_heads turtlesArray level numberOfClusterHeads
numberOfClusterTypes overallNumberOfNeededClusters  normalizationFactor
currentMainTurtle  p x advertiseBlock clusterHeadsBlock iterations
totalDistances AVGNOMembers NOMembers lastOne firstOne startingCluster central-
turtle nearestCluster isolatedNode nearestTemp endd check firstNode lastNode
enddd]

;In Netlogo, nodes are called turtles and they contain attributes which are
declared ;here

turtles-own [joined beingCH rangeOfAdvertise number edge clusterType member
flag CHMembers]

;==========================================================================

;Steup is the main procedure in this program and the program starts from this
;procedure.

;==========================================================================

to setup

Let counter 0 ;To count the number of iterations
Let temp array:from-list n-values 6 [0]

Let tempMem array:from-list n-values numberOfClusterHeads [0]
```

```
;After each iteration of the algorithm, the average of the number of members of the
;clusters are updated in the AVGNOMembers array.

set AVGNOMembers array:from-list n-values (ceiling ((granularityfactor *
numberOfClusterTypes) ) + 1) [0]

set AVGNOMembersROA array:from-list n-values (ceiling ((granularityfactor *
numberOfClusterTypes) ) + 1) [0]  let tempFirst 0

set firstOne 0 2

repeat 1[ ; This loop is for executing the algorithm for different RoA's
distributions

  set counter counter + 1

  set temp AVGNOMembers

  set tempFirst firstOne

  start

  set firstOne tempFirst

  set AVGNOMembers temp

 ;set  members tempMem

      printNumberOfMembers ;A

      set firstOne 1

  ]

  let m 0

  repeat 100[ ;This loop is to obtain the average of the number of clusters with
each

              ;size for all the iterations.

    array:set AVGNOMembers m array:item AVGNOMembers m / counter


  Print array:item AVGNOMembers m

  set m m + 1

  ]


 set enddd true

end
```

```
;===========================================================================

; This procedure takes care of the initializations of the environment before the
;algorithm starts.

;===========================================================================

to start

  clear-all

  create-turtles 400 ; There are 400 nodes in the network

  let n 1

  let m 0

  repeat 400 [ ; Initializes the turtles which are the nodes in the network

      ask turtle m [set number m set color white set edge "null" set joined "null"
set beingCH "null" set member "null" set CHMembers "null"]

      set m ( m + 1)

    ]

  let temp 0

;*****************Boundary: Assign the attributes of the nodes that are on the

                  ;boundary of the network layout.

  ask turtle 0 [set edge ("left-down") ]

  while [ n < 19] [

    ask turtle n [set edge "left" ]

    set n (n + 1)

  ]

  ask turtle 19 [set edge "left-up"]

  set n ( 1 )

  while [ n < 19] [

    set temp (n * 20)

    ask turtle temp [set edge "down" ]

    set n (n + 1)

  ]

    set temp (20 * 19 )

  ask turtle temp [set edge "right-down" ]

  set n ( 2 )

  while [ n < 20] [
    set temp (n * 20 - 1 )
    ask turtle temp [set edge "up" ]
    set n (n + 1)
  ]
    set temp (399)

  ask turtle temp [set edge "right-up" ]
  set n ( 1 )
  while [ n < 19] [
    set temp (380 + n)
    ask turtle temp [set edge "right" ]
    set n (n + 1)
  ]
 ;************************
  ask turtles [set shape "dot"]
  ask turtles [set size 1.5]
  let i 0
  let j 0
  set temp 0
  ;setting the location of each node
  while [ i < 20][
    while [j < 20][
      set temp  (i * 20 + j)
      ask turtle temp [setxy (i * 1)- 10 (j * 1)- 10]
      set j (j + 1)
    ]
    set i (i + 1)
    set j ( 0)
  ]
  set granularityFactor 5;
  probablityOfRoA
  let g 0
```

```
end

;===============================================================================

; This procedure determines the number of expected nodes in each cluster.

;===============================================================================

to probablityOfRoA

set numberOfClusterTypes 5

set RoA array:from-list n-values numberOfClusterTypes [0]

set numberOfNeededClusters  array:from-list n-values numberOfClusterTypes [0]

set totalDistances 1

;clustertypes

array:set RoA 0 (1)

array:set RoA 1 (2)

array:set RoA 2 (3)

array:set RoA 3 (4)

array:set RoA 4 (5)

;The numberOfNeededClusters is set for each cluster. It is modified in another
;version of the program to be generated pseudo-randomly.

array:set numberOfNeededClusters  0 (10);

array:set numberOfNeededClusters  1 (97)

array:set numberOfNeededClusters  2 (15)

array:set numberOfNeededClusters  3 (51)

array:set numberOfNeededClusters  4 (70)


sortByNumberOfNeededClusters

  computeExpectedValues

  findNumberOfClusterHeads

  clusterHeads

  ;computeProbabilities

  set iterations 0

  assignDistancesToClusterHeads


   let m 0

   let notFinished true
   while [notFinished = true][

   set notFinished false


   while [m < numberOfClusterHeads][

               if [rangeOfAdvertise] of array:item cluster_Heads m - [member] of
array:item cluster_Heads m > 3

                 [ set notFinished true

                 ]

               set m m + 1

           ]

         clusterInitial


   ]

; The isolatedNodes1 is one of the optional procedures of the algorithm which might

; be enabled for comparisons. This procedure has two versions isolatedNodes and

; isolatedNodes1.

;isolatedNodes1


end


;===============================================================================

; The clusters are sorted by the number of needed clusters where the clusters with
;more needs could be given a higher priority.

;===============================================================================

to sortByNumberOfNeededClusters

let k 0

let i 0

let j 0

let temp 0

let tempArr array:from-list n-values numberOfClusterTypes[0]

set tempArr (RoA)

while [ i < numberOfClusterTypes] [
  set j (i)

  while [j < numberOfClusterTypes] [

    if (array:item numberOfNeededClusters  i < array:item numberOfNeededClusters
j) [

      set temp (array:item numberOfNeededClusters  i)

      array:set numberOfNeededClusters  i array:item numberOfNeededClusters  j

      array:set numberOfNeededClusters  j temp

      set temp (array:item RoA i)

      array:set RoA i array:item RoA j

      array:set RoA j temp

    ]

    set j (j + 1)


  ]

  set i ( i + 1)

]


end


;===============================================================================

; This assigns clusterheads pseudo-randomly.

;===============================================================================

to clusterHeads

  let i 0

  set cluster_heads array:from-list n-values numberOfClusterHeads [0]

  while [i < numberOfClusterHeads] [

    array:set cluster_heads  i one-of turtles

      ask array:item cluster_heads i [set member 1 ]

    if [beingCH] of array:item cluster_heads i != 1[

      ask array:item cluster_heads i [set beingCH 1 set size 2 ]

      set i (i + 1)

    ]

  ]
```

```
 ; print cluster_heads

end

;===============================================================================

; This procedure is for future work and uses a method to estimate the number of
;nodes needed in the clusters based on historical data.

;===============================================================================

to computeExpectedValues

  set expectedValues array:from-list n-values numberOfClusterTypes [0]

  let i 0

  let temp 0

  while [i < numberOfClusterTypes][

    set overallNumberOfNeededClusters  (array:item numberOfNeededClusters  i +
overallNumberOfNeededClusters )

    set i (i + 1)

  ]

  set i (0)

  while [i < numberOfClusterTypes][

    set temp precision ((array:item numberOfNeededClusters
i)/(overallNumberOfNeededClusters )) 3;; expected value of each RoA

    array:set expectedValues i temp

    set i (i + 1)

  ]

 ;print expectedValues

end

;===============================================================================

; This method computes the number of clusterheads based on the input.

;===============================================================================

to findNumberOfClusterHeads

  let temp 1

  let i 0

  let ch 0

while [i < numberOfClusterTypes ][

    set temp (precision ((array:item expectedValues i) * (array:item RoA i) *
normalizationFactor) 0) + temp
```

```
    set i i + 1

  ]

  set numberOfClusterHeads precision (400 / temp) 0

end


;================================================================================

; This assigns the respective RoA to each cluster.

;===============================================================================

to assignDistancesToClusterHeads

  let clusterNumber 0

  let temp 0

  let i 0

  let j 0

 while [ i < numberOfClusterTypes] [

    set temp (precision ((array:item expectedValues i) * numberOfclusterHeads) 0)

    repeat temp [

   ; print [member] of array:item  cluster_heads j

      ask array:item  cluster_heads j [set rangeOfAdvertise (array:item RoA i) *
granularityFactor + (random granularityFactor -

       (granularityFactor / 2) + 0.5) set member 1  set CHMembers array:from-list
n-values (30) ["null"]]

      set j (j + 1)

     ; ask array:item cluster_heads i  [print (array:item RoA i) *
granularityFactor ;]

    ]

    set i ( i + 1)

   ]

clusterInitial

end

;===============================================================================

;This procedure initializes the clusterheads.

;===============================================================================

to clusterInitial
```

```
  set p 0  set x 1

  set level array:from-list n-values numberOfClusterHeads[0]

  set clusterHeadsBlock table:make

  while [p < numberOfClusterHeads ][

      table:put clusterHeadsBlock p array:from-list n-values 9[false]

    set p p + 1

  ]

  ;print clusterHeadsBlock

  set p 0

  set startingCluster 0 ; The variable starting cluster is used to simulate the
                             ;parallel advertisement

  joinToClusters3

end

;==============================================================================

; This is the last version of the procedure that is used to join the nodes to the
;clusters.

;==============================================================================

to joinToClusters3

  let temp 0

  let tempBlock 0

  let nextAdvertise array:from-list n-values numberOfClusterHeads["right"]

  let currentAdvertise "right"

  let turtleNumber 0

  let clusterNumber startingCluster

  let clusterNumberTemp 0

  let allClustersZero 0

  let level' 0

  let notfinished true

while [clusterNumberTemp < numberOfClusterHeads][

    set level' array:item level clusterNumber  ; The layers of advertisement

    if 0 < ([number] of array:item cluster_heads clusterNumber + level')

    and([number] of array:item cluster_heads clusterNumber + level') < 400 [

    if [joined] of turtle ([number] of array:item cluster_heads clusterNumber +
level') = clusterNumber
```

```
    or [beingCH] of turtle ([number] of array:item cluster_heads clusterNumber +
level') = 1

    [

    set currentMainTurtle turtle ([number] of array:item cluster_heads
clusterNumber + level')

    set turtleNumber ([number] of currentMainTurtle)

    if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member] of
array:item cluster_heads clusterNumber [

    set currentAdvertise (array:item nextAdvertise clusterNumber )


    ;;right

    if currentAdvertise = "right" ;Advertisement to the node on the right



    ifelse [edge] of currentMainTurtle != "right" ; Check for the boundary

         and [edge] of currentMainTurtle != "right-up"

         and [edge] of currentMainTurtle != "right-down"

         [


         set temp ([number] of currentMainTurtle + 20 )


           if [joined] of turtle temp = "null"

             and [beingCH] of turtle temp = "null"[

             ;;print ["right " ]


             ask turtle temp [set joined clusterNumber set color clusterNumber * 10
+ 5]


             ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]

]

         ][

           set tempBlock table:get clusterHeadsBlock clusterNumber

           array:set tempBlock 1 true

           array:set tempBlock 5 true
           array:set tempBlock 8 true

           table:put clusterHeadsBlock clusterNumber tempBlock

           ]

        set currentAdvertise ("up")

        ]

        ]


      if [rangeOfAdvertise] of array:item cluster_heads clusterNumber  > [member]
of array:item cluster_heads clusterNumber[


      if currentAdvertise = "up"       ;Advertisement to the node on the right

         ifelse [edge] of currentMainTurtle != "up" ; check for the boundary

         and [edge] of currentMainTurtle != "right-up"

         and [edge] of currentMainTurtle != "left-up"

        [

        set temp ( [number] of currentMainTurtle + 1)

         if [joined] of turtle temp = "null"

             and [beingCH] of turtle temp = "null"[

            ;; print "up"

             ask turtle temp [set joined clusterNumber set color ((clusterNumber *
10 + 5))]

             ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]

        ;   ask array:item cluster_heads clusterNumber [set rangeOfAdvertise
[rangeOfAdvertise] of  array:item cluster_heads clusterNumber - 1]

           ]

          ][

          set tempBlock table:get clusterHeadsBlock clusterNumber

           array:set tempBlock 2 true

           array:set tempBlock 5 true

           array:set tempBlock 6 true

           table:put clusterHeadsBlock clusterNumber tempBlock

          ]

         ]
      set currentAdvertise ("left")

      ]

      if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member] of
array:item cluster_heads clusterNumber[

      if currentAdvertise = "left" [;Advertisement to the node on the left


         ifelse [edge] of currentMainTurtle != "left" ; Check for the boundary

         and [edge] of currentMainTurtle != "left-up"

         and [edge] of currentMainTurtle != "left-down"

         [

         set temp ( [number] of currentMainTurtle - 20)

           if [joined] of turtle temp = "null"

             and [beingCH] of turtle temp = "null"[

             ;;print "left"

             ask turtle temp [set joined clusterNumber set color clusterNumber * 10
+ 5]

             ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]

            ; ask array:item cluster_heads clusterNumber [set rangeOfAdvertise
[rangeOfAdvertise] of  array:item cluster_heads clusterNumber - 1]

             ]

         ][

         set tempBlock table:get clusterHeadsBlock clusterNumber

           array:set tempBlock 3 true

           array:set tempBlock 6 true

           array:set tempBlock 7 true

           table:put clusterHeadsBlock clusterNumber tempBlock

         ]

       ]

        set currentAdvertise ("down")

       ]

       if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member]
of array:item cluster_heads clusterNumber[

              let m 0

       if currentAdvertise = "down" [;Advertisement to the node below
```

```
         ifelse [edge] of currentMainTurtle != "down" ; check for the boundary
         and [edge] of currentMainTurtle != "left-down"
         and [edge] of currentMainTurtle != "right-down"
         [
         set temp ( [number] of currentMainTurtle - 1)
        if [joined] of turtle temp = "null"
             and [beingCH] of turtle temp = "null"[
            ;;print "down"
            ask turtle temp [set joined clusterNumber set color clusterNumber * 10
+ 5]
            ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]
          ; ask array:item cluster_heads clusterNumber [set rangeOfAdvertise
[rangeOfAdvertise] of  array:item cluster_heads clusterNumber - 1]
           ]


         ][
         set tempBlock table:get clusterHeadsBlock clusterNumber
           array:set tempBlock 4 true
           array:set tempBlock 7 true
           array:set tempBlock 8 true
           table:put clusterHeadsBlock clusterNumber tempBlock
           ]
       ]
              set currentAdvertise ("right")
      ]
      ]
      ]
              set clusterNumber (clusterNumber + 1)
         set clusterNumbertemp (clusterNumbertemp + 1)
     if clusterNumber >= numberOfClusterHeads [
       set clusterNumber (clusterNumber - numberOfClusterHeads)
     ]

      ]

;*****************************************************************************

; This is the recursive part of the joiningCluster3 procedure that advertises the
;needed number of nodes after the first round of advertisement.

;*****************************************************************************


      set clusterNumber startingCluster

      set clusterNumbertemp 0

        set p p + 1

         while [clusterNumberTemp < numberOfClusterHeads][

           set currentMainTurtle turtle ([number] of array:item cluster_heads
clusterNumber)

         set tempBlock table:get clusterHeadsBlock clusterNumber

         if p = 1 and array:item tempBlock 1 = false[

         set temp [number] of currentMainTurtle + 20 * (x - 1)

         ifelse [edge] of turtle temp != "right"

         and [edge] of turtle temp != "right-up"

         and [edge] of turtle temp != "right-down"


            [

            array:set level clusterNumber 20 * x


            ][

           array:set tempBlock 1 true


           ]

           ]

         if p = 2 and array:item tempBlock 2 = false[

         set temp ([number] of  currentMainTurtle + 1 * (x - 1))

         ifelse [edge] of turtle temp != "up"

         and [edge] of turtle temp != "right-up"

         and [edge] of turtle temp != "left-up"
```

```
   [array:set level clusterNumber  1 * x ][
   array:set tempBlock 2 true

  ]
   ]

if p = 3 and array:item tempBlock 3 = false[
set temp ([number] of  currentMainTurtle + -20 * (x - 1))
ifelse [edge] of turtle temp != "left"
and [edge] of turtle temp != "left-up"
and [edge] of  turtle temp != "left-down"
   [ array:set level clusterNumber -20 *  x]
   [array:set tempBlock 3 true

  ]
  ]

if p = 4 and array:item tempBlock 4 = false[
ifelse [edge] of  turtle temp != "down"
and [edge] of turtle temp != "left-down"
and [edge] of turtle temp != "right-down"
   [array:set level clusterNumber -1 * x  ][
   array:set tempBlock 4 true

  ]
  ]
if p = 5 and array:item tempBlock 5 = false[
set temp ([number] of  currentMainTurtle + 20 * (x - 1) + 1)
ifelse [edge] of turtle temp != "right"
and [edge] of turtle temp != "right-up"
and [edge] of turtle temp != "right-down"
and [edge] of turtle temp != "left-up"
```

```
 and [edge] of turtle temp != "up"
 [
  array:set level clusterNumber 20 * x + 1
  ][
  array:set tempBlock 5 true
   ]
  ]
 if p = 6 and array:item tempBlock 6 = false[
 set temp ([number] of currentMainTurtle + -20 * (x - 1) + 1)
; print temp
 ifelse [edge] of  turtle temp != "left"
 and [edge] of turtle temp != "left-up"
 and [edge] of turtle temp != "left-down"
 and [edge] of turtle temp != "up"
 and [edge] of turtle temp  != "right-up"
 [
 array:set level clusterNumber -20 * x + 1
  ][
  array:set tempBlock 6 true
   ]
   ]
 if p = 7 and array:item tempBlock 7 = false[
 set temp ([number] of  currentMainTurtle  + -20 * (x - 1) - 1)
 if temp > -1[

 ifelse [edge] of turtle temp != "left"
 and [edge] of turtle temp != "left-down"
 and [edge] of turtle temp != "right-down"
 and [edge] of turtle temp != "left-up"
 and [edge] of turtle temp != "down"
 [
  array:set level clusterNumber -20 * x - 1
```

```
        ]
        [
        array:set tempBlock 7 true
        ]
        ]
       ]
     if p = 8 and array:item tempBlock 8 = false[

      set temp ([number] of  currentMainTurtle + 20 * (x - 1) - 1)
     if temp > -1[
     ifelse [edge] of turtle temp != "right"
     and [edge] of turtle temp != "left-down"
      and [edge] of turtle temp != "right-down"
      and [edge] of turtle temp != "right-up"
      and [edge] of turtle temp != "down"
      [
      array:set level clusterNumber 20 * x - 1
      ][array:set tempBlock 8 true
        ]
       ]
      ]
       table:put clusterHeadsBlock clusterNumber tempBlock
       set clusterNumber (clusterNumber + 1)
      set clusterNumbertemp (clusterNumbertemp + 1)
  if clusterNumber >= numberOfClusterHeads [
    set clusterNumber clusterNumber - numberOfClusterHeads
  ]
      ]
;***************************************************************
      set clusterNumber 0

      set notFinished false
```

```
         while [clusterNumber < numberOfClusterHeads][

            ;print [rangeOfAdvertise] of array:item cluster_heads clusterNumber

          if [rangeOfAdvertise] of array:item cluster_heads clusterNumber >
[member] of array:item cluster_heads clusterNumber [

              set notFinished  true

              ]

          set clusterNumber clusterNumber + 1

          ]

          ifelse  p < 9

            [

            ifelse startingCluster < numberOfClusterHeads - 1[

            set startingCluster startingCluster + 1

            ][

            set startingCluster 0

            ]

            joinToClusters3]

            [

            ifelse notFinished = true and x < 5[

              set p  0

              set x x + 1

              ifelse startingCluster < numberOfClusterHeads[

            set startingCluster startingCluster + 1

            ][

            set startingCluster 0

            ]

              joinToClusters3

            ]

            [

          ;isolatedNodes   ; should be uncommented for extension 2, method 1

          reclustering1    ; re-selecting clusterheads

            ]

       ]

end
```

```
;================================================================================

; This is one version of the reclustering procedures. In this procedures, after one
clustering, the center of the clusters are found and assigned as new clusterheads.
The clustering algorithm is executed again for the new clusterheads.

;================================================================================

to reclustering1



ifelse totalDistances > 0 and (totalDistances < 500  and iterations < 200)[

           ;isolatedNodes



           let n 0

           let m 0

           ; These matrices are used to calculate the central nodes

           let maxRow-turtle array:from-list n-values numberOfClusterHeads[0]

           let minRow-turtle array:from-list n-values numberOfClusterHeads[20]

           let maxColumn-turtle array:from-list n-values numberOfClusterHeads[0]
```

```
            let minColumn-turtle array:from-list n-values numberOfClusterHeads[19]

            let tempNum 0

            while [n < 400] [


               if [joined] of turtle n != "null" [

               set tempNum precision (([number] of turtle n) / 20) 0

               if tempNum > ([number] of turtle n) / 20

               [

                 set tempNum tempNum - 1

               ]


               if tempNum > (array:item maxColumn-turtle [joined] of turtle n) [

                 array:set maxColumn-turtle ([joined] of turtle n) tempNum

               ]

               if tempNum  < (array:item minColumn-turtle [joined] of turtle n)[

                 array:set minColumn-turtle [joined] of turtle n tempNum

               ]


               if (([number] of turtle n) - tempNum * 20)   > (array:item maxRow-
turtle [joined] of turtle n) [

                 array:set maxRow-turtle [joined] of turtle n ([number] of turtle n
- tempNum * 20)

               ]

               if (([number] of turtle n) - tempNum * 20)   < (array:item minRow-
turtle [joined] of turtle n) [

                 array:set minRow-turtle [joined] of turtle n ([number] of turtle n
- tempNum * 20)

               ]

             ]


             ask turtle n [set member "null" set joined "null"   set color white set
size 1.5 set beingCH "null"]

             set n n + 1

            ]

            set level array:from-list n-values numberOfClusterHeads[0]
```

```
          set p 0
          while [p < numberOfClusterHeads ][
            table:put clusterHeadsBlock p array:from-list n-values 9[false]
            set p p + 1
          ]
           set x 1 set p 0
           set m 0
           let tempClusterHeads array:from-list n-values numberOfClusterHeads[0]
           while [m < numberOfClusterHeads][
               array:set tempClusterHeads m array:item cluster_heads m
               set m m + 1
           ]



           set m 0
           while [m < numberOfClusterHeads][

             set tempNum (precision ((array:item maxColumn-turtle m +  array:item
minColumn-turtle m) / 2) 0)
             ;print tempNum
             if tempNum > (array:item maxColumn-turtle m +  array:item minColumn-
turtle m) / 2
              [
                set tempNum tempNum - 1
              ]

             let tempNum1 tempNum * 20 + precision ((array:item maxRow-turtle m +
array:item minRow-turtle m) / 2) 0

             if tempNum1 > tempNum * 20 + (array:item maxRow-turtle m +  array:item
minRow-turtle m) / 2
              [
                set tempNum1 tempNum1 - 1
              ]
             ; print tempNum1
             ;print minColumn-turtle
             ifelse [beingCH] of  turtle tempNum1 != 1[
             array:set cluster_heads m turtle tempNum1



              ][
              ifelse tempNum < 399 [
              array:set cluster_heads m turtle (tempNum1  + 1)
              ][array:set cluster_heads m turtle (tempNum1  - 1)
              ]
              ]

             ask array:item cluster_heads m [set member 1 set beingCH 1 set Color
green set size 2]
             set m m + 1
           ]
           set m 0
            set iterations iterations + 1
            while [m < numberOfClusterHeads][
               if [rangeOfAdvertise] of array:item cluster_Heads m > 5
                 [ask array:item cluster_Heads m [set beingCH "null"]
                   let tempTurtle one-of turtles
                   while [[beingCH] of tempTurtle != "null" ][
                     set tempTurtle one-of turtles
                     ]
                      array:set cluster_heads m  tempTurtle
                      ask array:item cluster_heads m [ set beingCH 1 set Color
green set size 2 set member 1]
                 ]
               set m m + 1
           ]
          set-current-plot "CHChanges"
          let i  0
          let tempX 0
          let tempY 0
          while [i < numberOfClusterHeads][
            set tempX [xcor] of array:item cluster_Heads i
            set tempY [ycor] of array:item cluster_Heads i

            ifelse tempX > [xcor] of array:item tempClusterHeads i [
              set tempX tempX -  [xcor] of array:item tempClusterHeads i
            ][
               set tempX [xcor] of array:item tempClusterHeads i - tempX
            ]
            ifelse tempY > [ycor] of array:item tempClusterHeads i [
             set tempY tempY -  [ycor] of array:item tempClusterHeads i
            ][
               set tempY [ycor] of array:item tempClusterHeads i - tempX
            ]
            set totalDistances totalDistances + tempX + tempY
            set i i + 1
          ]
          plot totalDistances
          assignDistancesToClusterHeads

          ][if totalDistances > 500 or iterations > 200[

            start
            ]
          ]
 end
;==============================================================================
; This is another version of the reclustering method.
;==============================================================================
to reclustering
ifelse totalDistances > 0 and (totalDistances < 500  and iterations < 200)[
;Checks for the threshold values
           let n 0
           let m 0
      ; These matrices are used to calculate the central nodes
           let maxRow-turtle array:from-list n-values numberOfClusterHeads[0]
           let minRow-turtle array:from-list n-values numberOfClusterHeads[20]
           let maxColumn-turtle array:from-list n-values numberOfClusterHeads[0]
           let minColumn-turtle array:from-list n-values numberOfClusterHeads[19]
           set central-turtle array:from-list n-values numberOfClusterHeads[1000]
           let centerXCors array:from-list n-values numberOfClusterHeads[0]
           let centerYCors array:from-list n-values numberOfClusterHeads[0]
           let tempNum 0
          ; centralNodes
           set n 0
           while [n < 400] [

           if [joined] of turtle n != "null" [
           set tempNum precision (([number] of turtle n) / 20) 0
              if tempNum > ([number] of turtle n) / 20
            [
             set tempNum tempNum - 1
             ]

             if tempNum > (array:item maxColumn-turtle [joined] of turtle n) [
               array:set maxColumn-turtle ([joined] of turtle n) tempNum
              ]
             if tempNum  < (array:item minColumn-turtle [joined] of turtle n)[
                array:set minColumn-turtle [joined] of turtle n tempNum
             ]
           if (([number] of turtle n) - tempNum * 20)   > (array:item maxRow-turtle
[joined] of turtle n) [

               array:set maxRow-turtle [joined] of turtle n ([number] of turtle n -
tempNum * 20)

             ]

             if (([number] of turtle n) - tempNum * 20)   < (array:item minRow-
turtle [joined] of turtle n) [

              array:set minRow-turtle [joined] of turtle n ([number] of turtle n -
tempNum * 20)

            ]

           ]



          ask turtle n [set joined "null"   set color white set size 1.5 set
beingCH "null" set member "null" set CHMembers "null"]

           set n n + 1

         ]

           set level array:from-list n-values numberOfClusterHeads[0]

          set p 0

          while [p < numberOfClusterHeads ][

            table:put clusterHeadsBlock p array:from-list n-values 9[false]

            set p p + 1

          ]

           set x 1 set p 0

           set m 0

           let tempClusterHeads array:from-list n-values numberOfClusterHeads[0]

           while [m < numberOfClusterHeads][

               array:set tempClusterHeads m array:item cluster_heads m

               set m m + 1

           ]

           set m 0




while [m < numberOfClusterHeads][
             set tempNum (precision ((array:item maxColumn-turtle m +  array:item
minColumn-turtle m) / 2) 0)

             ;print tempNum

             if tempNum > (array:item maxColumn-turtle m +  array:item minColumn-
turtle m) / 2

              [

                set tempNum tempNum - 1

              ]



             let tempNum1 tempNum * 20 + precision ((array:item maxRow-turtle m +
array:item minRow-turtle m) / 2) 0



             if tempNum1 > tempNum * 20 + (array:item maxRow-turtle m +  array:item
minRow-turtle m) / 2

              [

                set tempNum1 tempNum1 - 1

              ]

             ; print tempNum1

             ;print minColumn-turtle

             ifelse [beingCH] of  turtle tempNum1 != 1[

             array:set cluster_heads m turtle tempNum1



              ][

              ifelse tempNum < 399 [

              array:set cluster_heads m turtle (tempNum1  + 1)



              ][

              array:set cluster_heads m turtle (tempNum1  - 1)

                          ]

              ]

             ;set cluster_heads central-turtle

             ask array:item cluster_heads m [ set beingCH 1 set Color m * 10 + 5
set size 2 set member 1 set rangeOfAdvertise [rangeOfAdvertise] of array:item
tempClusterHeads m

             set CHMembers array:from-list n-values (30) ["null"]]
```

```
              set m m + 1


            ]

            set m 0

             set iterations iterations + 1

             while [m < numberOfClusterHeads][

                if [rangeOfAdvertise] of array:item cluster_Heads m - [member] of
array:item cluster_heads m > 3

                  [ask array:item cluster_Heads m [set beingCH "null"]

                    let tempTurtle one-of turtles

                    while [[beingCH] of tempTurtle != "null"][

                      set tempTurtle one-of turtles


                      ]

                       array:set cluster_heads m  tempTurtle

                       ask array:item cluster_heads m [ set beingCH 1 set Color m *
10 + 5 set size 2 set member 1 set rangeOfAdvertise [rangeOfAdvertise] of
array:item tempClusterHeads m

                       set CHMembers array:from-list n-values (30) ["null"]]

                  ]

                set m m + 1

            ]

           set-current-plot "CHChanges"

           let i  0

           let tempX 0

           let tempY 0

           while [i < numberOfClusterHeads][

             set tempX [xcor] of array:item cluster_Heads i

             set tempY [ycor] of array:item cluster_Heads i


             ifelse tempX > [xcor] of array:item tempClusterHeads i [

               set tempX tempX -  [xcor] of array:item tempClusterHeads i

             ][

                set tempX [xcor] of array:item tempClusterHeads i - tempX
             ]
             ifelse tempY > [ycor] of array:item tempClusterHeads i [
              set tempY tempY -  [ycor] of array:item tempClusterHeads i
             ][
                set tempY [ycor] of array:item tempClusterHeads i - tempX
             ]
             set totalDistances totalDistances + tempX + tempY
             set i i + 1
           ]
           plot totalDistances
           ;print totalDistances
              ;print "hello"
           let z 0
           while [z < 400] [

             if [beingCH] of turtle z != 1[
               ask turtle z [set size 1.5]
             ]
             set z z + 1
             ]
         if enddd != true[
         clusterInitial
         ]
           ][if totalDistances > 500 or iterations > 200[
             if enddd != true[
              start
              ]
             ]


           ]
end
;==============================================================================
; This procedure takes care of the nodes that have not already joined any cluster.

;================================================================================

to isolatedNodes

   set isolatedNode 0

   let clusterNumber 0

   let clusterToJoin 0

   let rowDistance 0

   let columnDistance 0

   let temp 0

   let distancee 400

   set nearestCluster 0

   let g 0

   let counter 0

  let firstNearest 0

  while [ isolatedNode < 400][

    while [g < numberOfCLusterHeads][

          ask array:item cluster_heads g [set flag 0]

     set g g + 1

   ]


    if [joined] of turtle isolatedNode = "null"  and [beingCH] of turtle
isolatedNode = "null"[

              set nearestTemp isolatedNode

              findeNearestClusterHead

             set firstNearest nearestCluster

             ask array:item cluster_heads nearestCluster [set flag 1]


         set counter 0

        set endd false

        ifelse [rangeOfAdvertise] of array:item cluster_Heads nearestCluster -
[member] of array:item cluster_Heads nearestCluster > -3[

          ask turtle isolatedNode [ set joined nearestCluster set color
nearestCluster * 10 + 5]
```

```
          ask array:item cluster_heads nearestCluster [set member member + 1
array:set CHMembers member - 1 isolatedNode]

          set endd  true

        ][

 ;""""""""""""""""""""""""""""""



        while [[rangeOfAdvertise] of array:item cluster_Heads nearestCluster -
[member] of array:item cluster_Heads nearestCluster <= -3 and endd != true][

           ; type  [rangeOfAdvertise] of array:item cluster_Heads nearestCluster
type "    ," type [member] of array:item cluster_Heads nearestCluster print ""

            set counter counter + 1

            set nearestTemp nearestCluster

            set nearestCluster 0

            findeNearestClusterHead

          print [rangeOfAdvertise] of array:item cluster_Heads nearestCluster -
[member] of array:item cluster_Heads nearestCluster

            ask array:item cluster_heads nearestTemp [set flag 1]

         let i 0

         let nearestNode 0

         while [[CHMembers] of array:item cluster_heads nearestTemp != "null" and i
< [member] of array:item cluster_heads nearestTemp][


             set columnDistance [ycor] of array:item cluster_heads nearestCluster

             set temp [ycor] of turtle i

             set columnDistance abs (columnDistance - temp)

             set temp [xcor] of array:item cluster_heads clusterNumber

             set rowDistance [xcor] of turtle nearestTemp

             set rowDistance abs(rowdistance - temp )

             set temp rowDistance + columnDistance

            if temp < distancee [

              set distancee temp

               set nearestNode i

                 ]


           set i i + 1
         ]



          ask turtle nearestNode [set joined nearestCluster set color
nearestCluster * 10 + 5]

          ask array:item cluster_heads nearestTemp [set member member - 1 array:set
CHMembers member - 1 "null"]

          ask array:item cluster_heads nearestCluster [set member member + 1
array:set CHMembers member - 1 nearestNode]



           set distancee 400

         ]



        ]

     ]

      set distancee 400

       set temp 0

       set nearestCluster 0

      set clusterNumber 0

            findeNearestClusterHead

           set isolatedNode isolatedNode + 1

   ]

;******************************

  end


;===============================================================================

; This procedure is used to output the results.

;===============================================================================

to printNumberOfMembers

let m 0



set NOMembers array:from-list n-values 6 [0]

    set m 0

while [m < numberOfClusterHeads ][

ifelse [member] of array:item cluster_heads  m < 5 [
  array:set NOMembers 1 array:item NOMembers 1 + 1
  ][ifelse [member] of array:item cluster_heads m < 10 [
    array:set NOMembers 2 array:item NOMembers 2 + 1
    ][ifelse [member] of array:item cluster_heads m < 15 [
      array:set NOMembers 3 array:item NOMembers 3 + 1
      ][
        ifelse [member] of array:item cluster_heads m < 20 [
            array:set NOMembers 4 array:item NOMembers 4 + 1
        ][
          array:set NOMembers 5 array:item NOMembers 5 + 1
    ]
      ]
    ]
]
set m m + 1
]
test
end
;===============================================================================
; This procedure checks to see if the clustering algorithm is working correctly. In
;other words, it checks is the number of nodes taken by each cluster against the
;number of needed nodes in that cluster.
;===============================================================================
to test
set NOMembers array:from-list n-values 13 [0]
let m 0
while [m < numberOfClusterHeads ][
ifelse [member] of array:item cluster_heads m < 7 [
  array:set NOMembers 1 array:item NOMembers 1 + 1
  ][ifelse [member] of array:item cluster_heads m < 10 [
    array:set NOMembers 2 array:item NOMembers 2 + 1


    ][ifelse [member] of array:item cluster_heads m < 13 [
      array:set NOMembers 3 array:item NOMembers 3 + 1
      ][
        ifelse [member] of array:item cluster_heads m < 16 [
            array:set NOMembers 4 array:item NOMembers 4 + 1
        ][
        ifelse [member] of array:item cluster_heads m < 19[
            array:set NOMembers 5 array:item NOMembers 5 + 1
        ]
        [
        ifelse [member] of array:item cluster_heads m < 22 [
            array:set NOMembers 6 array:item NOMembers 6 + 1
        ]
        [
        ifelse [member] of array:item cluster_heads m < 25 [
            array:set NOMembers 7 array:item NOMembers 7 + 1
        ][ ifelse [member] of array:item cluster_heads m < 28 [
            array:set NOMembers 8 array:item NOMembers 8 + 1
       ; ][ ifelse [member] of array:item cluster_heads m < 21 [
      ;      array:set NOMembers 9 array:item NOMembers 9 + 1
       ; ][ ifelse [member] of array:item cluster_heads m < 23 [
      ;      array:set NOMembers 10 array:item NOMembers 10 + 1
      ;  ][ ifelse [member] of array:item cluster_heads m < 25 [
     ;       array:set NOMembers 11 array:item NOMembers 11 + 1
     ;   ]
     ][


            array:set NOMembers 9 array:item NOMembers 9 + 1
        ]
      ]
      ]
    ]
    ]
```

```
    ]
    ]
]
set m m + 1
]
  set m 1
if firstOne = 0[
 repeat 10[



 ; array:set AVGNOMembers m array:item NOMembers m
  set m m + 1
]
]
set m 1
repeat 10[

  array:set AVGNOMembers m ((array:item NOMembers m + array:item AVGNOMembers m))
  set m m + 1
]
set m 0
repeat numberOfClusterHeads [
type [member] of array:item cluster_heads m type ", "

set m m + 1
]

print " "
set m 0
repeat numberOfClusterHeads [
type [rangeOfAdvertise] of array:item cluster_heads m type ", "
```

```
set m m + 1

]

print " "

print "===============================; "

Let shapeArr array:from-list n-values numberOfClusterHeads [0]

array:set shapeArr 0 "star"

array:set shapeArr 1 "circle"

array:set shapeArr 2 "triangle"

array:set shapeArr 3 "pentagon"

array:set shapeArr 4 "x"

array:set shapeArr 5 "leaf"

array:set shapeArr 6 "plant"

set m 0

repeat 400 [

if [joined] of turtle m != "null"

[ask turtle m [set shape array:item shapeArr (joined mod 6)set size 0.7 set color
([joined] of turtle m) * 10 + 5]]

set m m + 1

]

set m 0

repeat numberOfClusterHeads [

ask array:item cluster_heads m  [set shape array:item shapeArr (m mod 6) set size 1
set color m * 10 + 5]


set m m + 1

]

end


;================================================================================

; This is a procedure that is used in the crawling method. Two different methods
are ;tested for finding the nearest clusterhead.

;==============================================================================

to findeNearestClusterHead

     let clusterNumber 0
```

```
   let distancee 400
   let rowDistance 0
   let columnDistance 0
   let temp 0

   set nearestCluster 0

        while[clusterNumber < numberOfClusterHeads][

          ;   type nearestCluster print " "
       ; type clusterNumber type "       " print nearestTemp
         ;test
         if clusterNumber != nearestTemp[

         set columnDistance [ycor] of array:item cluster_heads clusterNumber
         set temp [ycor] of turtle nearestTemp
          set columnDistance abs (columnDistance - temp)
             set temp [xcor] of array:item cluster_heads clusterNumber
           set rowDistance [xcor] of turtle nearestTemp
              set rowDistance abs(rowdistance - temp )

           set temp rowDistance + columnDistance

         ifelse temp < distancee and ([flag] of array:item cluster_heads
clusterNumber) = 0[
                if check = true[
         ]
             set distancee temp
             set nearestCluster clusterNumber

             type nearestCluster type " , "
           ][if temp = distancee[
               if  ([rangeOfAdvertise] of (array:item cluster_Heads nearestCluster)
- ([member] of array:item cluster_heads nearestCluster)) <

                   ([rangeOfAdvertise] of array:item cluster_Heads clusterNumber -
[member] of array:item cluster_heads clusterNumber) and

                      [flag] of array:item cluster_heads clusterNumber = 0 [

                          set nearestCluster clusterNumber

                ]

           ]

         ]

       ]

        set clusterNumber ClusterNumber + 1

       ]

       if nearestCluster = nearestTemp[

   set endd true

               ]

      set clusternumber 0

       while[clusterNumber < numberOfClusterHeads][

      ; type [flag] of array:item cluster_heads clusterNumber type " , "

       set clusterNumber ClusterNumber + 1

       ]

       print " "

             print nearestCluster

end

;================================================================================

; This is another version of attaching the isolated nodes to the clusters.

;==============================================================================

to unjoinedNodes1

let shortestPaths array:from-list n-values numberOfClusterHeads ["null"]

let nodee 0

let fin false

while [fin = false][

set fin  true
```

```
while [nodee < 400][


if ([joined] of turtle nodee = "null")[

;:::::::::::::::::::::::::::::::::::::finding the neediest clusterhead

 set fin false

 let clusterNumber 0

 let neediest 0

 while [clusterNumber < numberOfClusterHeads][

    ask array:item cluster_heads clusterNumber [set joined clusterNumber ]

    if (  [rangeOfadvertise] of array:item cluster_heads clusterNumber - [member]
of array:item cluster_heads clusterNumber

              > [rangeOfadvertise] of array:item cluster_heads neediest - [member]
of array:item cluster_heads neediest )[


           set neediest clusterNumber



       ]



    set clusterNumber clusterNumber + 1


 ]

;:::::::::::::::::::::::::::::::::::::find the nearest cluster to the node

let tempNode nodee

findeNearestClusterHead1

;::::::::::::::::::::::::::::::::::::find the shortest path from nearest clusterhead
to the neediest cluster (greedy approach)

set firstNode nodee

set lastNode [number] of array:item cluster_heads neediest

shortestpath

]

set nodee nodee + 1

]

]

end
```

```
;=================================================================================
; This is a procedure that is used in the crawling method.
;===============================================================================
to findeNearestClusterHead1


   let clusterNumber 0
   let distancee 400
   let rowDistance 0
   let columnDistance 0
   let tempNode 0
   let temp 0
   set nearestCluster 0
   while[clusterNumber < numberOfClusterHeads][
         if clusterNumber != tempNode[
              set columnDistance [ycor] of array:item cluster_heads clusterNumber
              set temp [ycor] of turtle nearestTemp
              set columnDistance abs (columnDistance - temp)
              set temp [xcor] of array:item cluster_heads clusterNumber
              set rowDistance [xcor] of turtle nearestTemp
              set rowDistance abs(rowdistance - temp )
              set temp rowDistance + columnDistance
               if temp < distancee and ([flag] of array:item cluster_heads
clusterNumber) = 0[
                 set distancee temp
                 set nearestCluster clusterNumber
                 ]
               ]
       set clusterNumber ClusterNumber + 1
              ]
      set clusterNumber 0
end
;===============================================================================
;This procedure finds the shortest path between the node that is crawling and the
cluster ;that needs the node.

;================================================================================

to shortestPath

let Hdirection 0

let Vdirection 0


let rowDistance abs ( [ycor] of turtle firstNode - [ycor] of turtle lastNode)

let columnDistance abs ( [xcor] of turtle firstNode - [xcor] of turtle lastNode)

let i 0

let j 0

let currentNode firstNode

let nextNode 0

;ask turtle firstNode [set color red]

;ask turtle lastNode [set color red]

ifelse [xcor] of turtle firstNode > [xcor] of turtle lastNode[

  set Hdirection -1

][

    set Hdirection 1

]

ifelse [ycor] of turtle firstNode > [ycor] of turtle lastNode[

  set Vdirection -1

][

    set Vdirection 1

]

set i 0

;print lastNode

while [i < columnDistance and [joined] of turtle currentNode != [joined] of turtle
lastnode][

    set nextNode currentNode + hdirection * 20

    if [joined] of turtle nextNode != [joined] of turtle currentNode[

      ask turtle currentNode [set joined [joined] of turtle nextNode set color
[color] of turtle nextNode]

    ]
    set currentnode nextNode


set i i + 1

]

set i 0

while [i < rowDistance and [joined] of turtle currentNode != [joined] of turtle
lastnode][

    set nextNode currentNode + vdirection

            ask turtle currentNode [set joined [joined] of turtle nextNode set
color [color] of turtle nextNode]

        set currentnode nextNode

set i i + 1

]

ask turtle lastNode [set member member + 1 ]end
```

VITA

Alireza Boloorchi Tabrizi

Candidate for the Degree of

Doctor of Philosophy

Thesis: ON THE CONJUNCTION OF NETWORK SECURITY REQUIREMENTS AND CLUSTERING: A NEW FRAMEWORK FOR RELIABLE AND ENERGY-EFFICIENT COMMUNICATION

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Doctor of Philosophy degree in Computer Science at Oklahoma State University, Stillwater, Oklahoma in July 2014.

Completed the requirements for the Master of Science degree in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December 2011.

Completed the requirements for the Bachelor of Science degree in Computer Engineering at Amirkabir University of Technology, Tehran, Iran in December 2008.

Experience:

Worked at Oklahoma State University as a research and teaching associate 2009-2013. Collaborated with The Crosby Group for research and development 2013-2014. Worked as Software Developer at Central Rural Electric Cooperative 2011-2012. Served as the president and vice president of ACM chapter at OSU 2011-2013.