Random Distribution for Data Survival in Unattended Wireless Sensor Networks

By

Thi My Y Vo

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

August 2010

Your file Votre référence
ISBN: 978-0-494-71558-1
Our file Notre référence
ISBN: 978-0-494-71558-1

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Random Distribution for Data Survival in Unattended Wireless Sensor Networks

submitted by

Thi My Y Vo, B.Eng.

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical Engineering

Chair, Dr. Howard Schwartz, Department of Systems and Computer Engineering

---

Thesis Supervisor, Dr. Jerome Talim

Carleton University
August 2010

Abstract

In Unattended Wireless Sensor Networks (UWSNs), sensors operate without the supervision of an online sink; instead, an itinerant sink regularly visits the network to acquire data. Thus, there is no real-time communication between the sensors and the sink; therefore, sensors collect data and store them locally. Due to the data accumulation in sensors' storage waiting for the sink to offload, data survival is crucial, as an adversary can visit the network and corrupt the collected data between sink visits.

In this thesis, we propose simple and effective algorithms to increase data survival in a homogeneous UWSN, without implementing cryptography. The key feature of our algorithms is to randomly distribute data within the network. We evaluate their performance by using numerical simulations. Our findings indicate that the proposed algorithms successfully defeat an adversary even when it has enough energy and time to corrupt all the nodes in the entire network.

Acknowledgements

I am heartily thankful to my supervisor, Professor Jerome Talim, who was helpful and offered encouragement, invaluable assistance, support and guidance from the initial to the final level during the completion of the thesis.

Special thanks to Toilers research group for their help in the implementation of the thesis simulation program.

I wish to express my love and gratefulness to my families; for their understanding and dedication, through the duration of my studies.

I would like to offer my thanks to all of those who supported me in any respect to make this thesis possible.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADV	Adversary
DC	Damage Control
DR	Damage Recovery
PRNG	Pseudo random number generator
RaDD	Random Data Distribution
TRNG	True random number generator
UWSN	Unattended Wireless Sensor Network
WSN	Wireless Sensor Network

Chapter 1: Introduction

In the ongoing development of wireless communication, Wireless Sensor Networks (WSNs) are receiving increased attention due to their many useful and popular applications. Features of an WSN include self-management, ease of deployment, low cost and multifunction. In this thesis, we will focus on a particular type of WSN known as an Unattended WSN, or UWSN. This chapter will provide an overview and description of an UWSN and delineate specific issues and challenges surrounding the technology. As well, we will introduce our proposed strategy of maximizing data survival in the face of adversarial attacks.

1.1 Overview

A Wireless Sensor Network is a cooperative network of distributed autonomous nodes to perform the tasks of monitoring physical or environmental conditions. Each node is a small wireless communication device which typically consists of: 1) a sensor to sense data from the environment; 2) a micro control unit which houses the processing capacity and controls all activities of the node; 3) a radio transceiver, which is responsible for wireless communication; 4) an energy source such as solar cells and/or batteries; and 5) a memory to store information. A sensor node is able to process its sensed data, exchange data with other sensors in the network, and communicate with a central operator.

Communications in the network are wireless and, after being deployed, usually sensors are self-management [1], [2]. The terms 'sensor node', 'sensor', and 'node' will be used interchangeably throughout this thesis.

UWSN is an emerging type of Wireless Sensor Networks. UWSN is unattended most of the time, and a sink (a.k.a. a collector) operates in a dynamic mode, visiting the network regularly to collect data. Nodes collect data and store them locally. Sensed data values are accumulated in sensors' storage and wait for the sink to offload [3]-[10]. Figure 1 shows an example of an UWSN

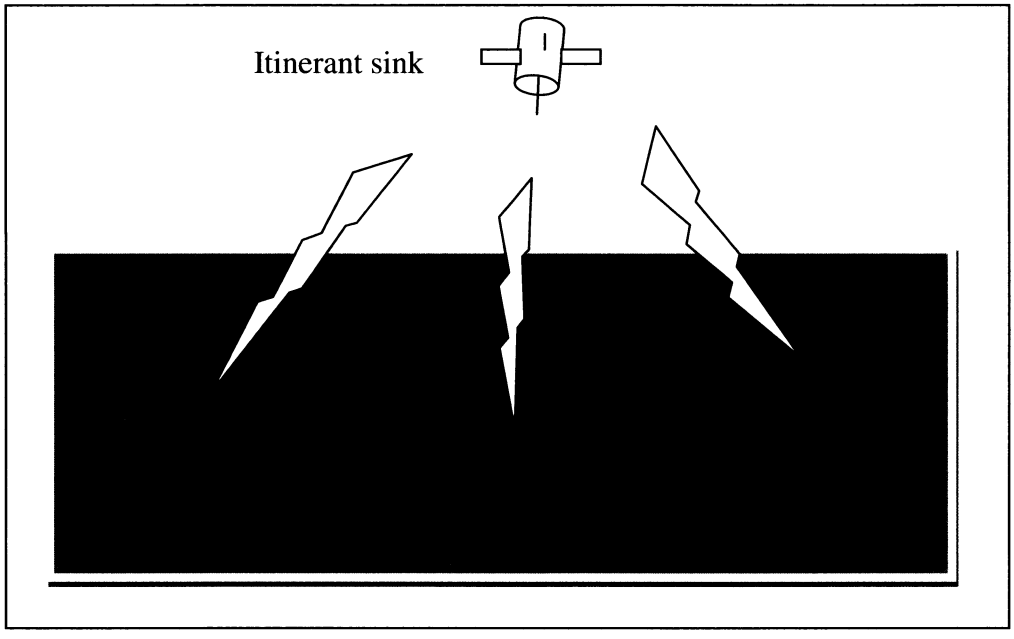

Figure 1: An example of an Unattended Wireless Sensor Network

The motivation of UWSN stems from the desire to avoid the failure or corruption of a central node. Because the sink holds sensed data gathering from sensor nodes in the network, it is an attractive node for enemy to attack. UWSN is also motivated by applications where are unsuitable for an online sink because a sink would be inconvenient to deploy, or needs to be itinerate, or should be turned off periodically to

conserve energy [3]-[10]. For example, a sensor network monitoring the enemy activities in the battlefield would be extremely vulnerable with a static sink. In this hostile environment, the sink is more convenient to be dynamic to evade adversaries. The case of WSN applications deployed in a wide range geographical area would require an itinerant sink to travel around to collect data. Similarly, for certain areas, intermittent sinks are more applicable as they collect data for a long period and turn off the sink to save energy. In the last decade, UWSNs have been used widely in WSN applications such as remote battlefield and tactical applications. In these applications, unattended sensors are deployed to detect underground sound and vibration. Examples of UWSN applications also include military applications monitoring of underground troop movements, border crossings for monitoring illegal activities, monitoring of nuclear emissions, enemy's aircrafts, historical data collection, anti-poaching protection in national parks monitoring firearm discharge, and applications in remote and inaccessible locations [6] [25].

1.2 Problem Statement

Along with the growing popularity of WSN, it continues to be plagued by issues of data security, a situation which has prompted considerable research over the past decade. Because WSNs are vulnerable to many kinds of attacks due to their inherent features such as being self-organized and lacking tamper-resistance [11], [12]. Most of the previous research has focused on data security in the presence of a static/online sink which (1) is a trusted party (i.e., tamper-resistant), (2) can obtain collected data from sensors in real

time (or nearly real time), and (3) can take appropriate action instantly to cease the further effects of an adversary (ADV) if one is detected [13].

However, our focus is on security in UWSNs which is more challenging than for WSNs because most of the time sensors' activities are left unattended. Sensors are not able to communicate with the sink in a real time manner and do not offload data immediately after collecting them. An adversary (ADV) can easily take advantage of the time between sink visits to roam the network with impunity and thereby learn the network topology and security strategy, compromise sensors, alter or delete the collected data in the sensors' storage, and leave the network without leaving a trace to the collector. In light of such potential overwhelming and pervasive threats, the issue here is how to protect data against an ADV or how to maintain data survival until a collector arrives [10].

1.3 Thesis Contribution

In response to the problem for UWSN as stated in Section1.2, the conventional line of defences for WSNs cannot be applied to UWSNs because WSN defence strategies mainly rely on an online sink and real-time detection [6]. Moreover, cryptography does not guarantee data survival because if the secret is exposed, an ADV can take control of the nodes and manipulate data. Thus, to maximize the data survival probability, sensor collaboration is necessary to mislead an adversary [14]. Over the past few years, several mitigating techniques using sensor cooperatives have been proposed to maximize data survival in UWSNs. These techniques will be reviewed in Chapter 2.

In this thesis, we develop random data distribution algorithms to increase the likelihood for data survival in homogeneous UWSNs, without implementing cryptography. The algorithms deploy data redundancy in a random fashion, while sensors cooperate to spread data within the network. Specifically, sensor nodes collect data, store them, and then broadcast them to neighbouring nodes; neighbouring nodes then store and/or forward them based on random decisions. Data are replicated at some arbitrary nodes within the network so that an ADV cannot know or guess where all the data replications are. This increases the rate of data survival when the sink arrives to collect data. The performance of these algorithms is evaluated under simulation results. Results of evaluations show that our algorithms are effective in providing data survival against an adversary even if it has corrupted every single node in the network.

Our contribution has been published in SENSORCOMM, 2010 Fourth International Conference on Sensor Technologies and Applications, "Random Distribution for Data Survival in Unattended Wireless Sensor Networks," pages 468-471, 2010.

Chapter 2: Literature review

A UWSN has distinctive characteristics, in that it operates without the supervision of a sink and stores accumulated data in local sensors' storage for a long time. This has been attracted to a new kind of adversary – a mobile ADV [16],[17]. This ADV can stealthily learn a network's strategies, compromise sensor nodes and ultimately take control of them. Moreover, it is able to roam the network from node to node, corrupting and taking over one, and then moving onto another.

Research into UWSN strategies to counteract the effects of adversaries is as urgent as it is timely. The following section overviews some research recently undertaken into maximizing data survivability against a mobile ADV.

2.1 Cryptography

Cryptography is well-known as an effective strategy against many kinds of attacks. It is a mean to hide information such as data content, data origin, and the time of collection [14]. Cryptography can be divided into two types: symmetric-key cryptography and public-key cryptography

Symmetric-key Cryptography

In a symmetric-key algorithm, both sender and receiver use the same key for encryption and decryption [18]. Therefore, when a sensor is compromised by a mobile ADV, the key is exposed, and the ADV can then decrypt the information.

Public-key Cryptography

In public-key cryptography or asymmetric cryptography algorithms, sender and receiver use two different keys for encryption and decryption: a public key and a private key [18].

Public-key cryptography is known as a high-cost algorithm. Though previously not recommended for wireless sensor networks, it has been recently revised because of its advantages[14]. In public-key cryptography, the sink's public key is distributed freely to all sensors in the network, while the sink's private key is secret and unknown to sensors. Sensors encrypt data using the public key. Only the sink can decrypt data using its secret private key. When an ADV corrupts the network, it is only able to learn the public key; it does not know the sink's private key and thus cannot decrypt the data. However, an ADV can detect target data by attempting to encrypt its duplicate using the sink's public key [14]. If an ADV's purpose is to try deleting target data before the sink arrives, it may achieve its purpose.

Therefore, in protecting nodes against an ADV attack in UWSN, the use of basic cryptography, or cryptography in its current state, is not effective. What is required is something far more advanced than conventional cryptography to protect data against mobile ADVs. If an ADV corrupts a node, it may learn the current secret, but it would

not be able to derive any other secrets further than that. Pietro *et al.* in [19] and [20] suggest that a successful security for a UWSN can be defined as the ability to achieve both forward and backward security, as can be viewed in the following:

Assume that an ADV has a certain amount of time to corrupt a given sensor node. The authors define data security as being based on the time a sensor is compromised and where:

(1) data is collected before the sensor is compromised,

(2) data is collected during the time the sensor is compromised, and

(3) data is collected after the sensor is compromised.

We will not consider data security for (2) because, in this case, the ADV is in full control of the node. To provide data security for (1), we need **forward secrecy**. Specifically, a cryptographic protocol is considered as forward secrecy if an ADV reveals the current secret, it cannot derive the secrets before the compromise time. To provide data security for (3), we need **backward secrecy**, whereby if the current secret is exposed, it does not lead to exposure of secrets after the compromise time.

Forward secrecy

One effective technique for obtaining forward secrecy is to update secret keys periodically using key evolution with one-way hash function [21]. Assume that time is divided into time intervals (also called rounds) in which data is collected per round. The key for the next round is computed using a one-way hash function of the current key as being represented in the formula

$$K_{r+1} = F(K_r). \quad (1)$$

This work is done at the end of each round and the current key is deleted after computing the next round's key. With this strategy, an ADV can only learn the current round's key and later rounds' keys but it cannot derive previous rounds' keys. Thus, the target data remains secure if it is collected before an ADV arrives.

Some recent research presents options for forward security as in [22], [23] and [24]. [22] and [23] make use of forward-secure sequential aggregate authentication in UWSN to minimize storage and bandwidth overhead. The idea is to allow a signer to aggregate multiple signatures at different rounds into a single fixed size signature. To verify a signature, it requires the verification of every component aggregated in that signature. Thus, if the current key is compromised, it does not lead to the invalidation the authenticity of pre- compromise data. [22] exploits hash chains together with BLS based signature and MAC function in order to compute and validate aggregated signatures. Hash Based Sequential Aggregate and Forward Secure Signature (HaSAFSS) schemes in [23] utilize MAC-based aggregate signatures and the public verifiability of bilinear map based signatures via Timed-Release Encryption to provide forward security. Specifically, HaSAFSS not only allows a signer to aggregate multiple signatures but also allows the signatures to be publicly verifiable. Compare to [22], these schemes are lower storage overhead and higher computational efficiency. Similarly, Shao *et al.* in [24] investigate data location privacy for data-centric UWSNs. Here, the notion is that data be sent to

designated nodes for storage. A secret key, regularly updated, pinpoints these nodes to prevent the ADV from finding out which are the storage nodes.

The aforementioned techniques offer forward security and successfully prevent adversaries from learning secrets before the compromised time, however, they do not provide backward security and therefore cannot protect data against a proactive ADV which compromises the network before the target data is collected [17]. The following discusses how to achieve backward security in UWSNs.

Backward secrecy

Backward secrecy can be obtained by using a trusted third party, but, as the sink in UWSN is not online, we cannot apply this technique here. Pietro *et al.* in [19] and [20] propose backward secrecy protection by using sensor cooperation among sensors in the network. The next round's key is generated using the cooperation of other sensors in the network. With this strategy, backward secrecy is protected if at least one of the cooperation sensors is not compromised.

A combination of key evolution and sensor cooperation would provide both forward and backward secrecy, thereby providing security for the UWSN. Figure 2 shows the next round's key calculation using the current round's key with the help of other sensors. This key calculation satisfies both forward and backward secrecy [19].

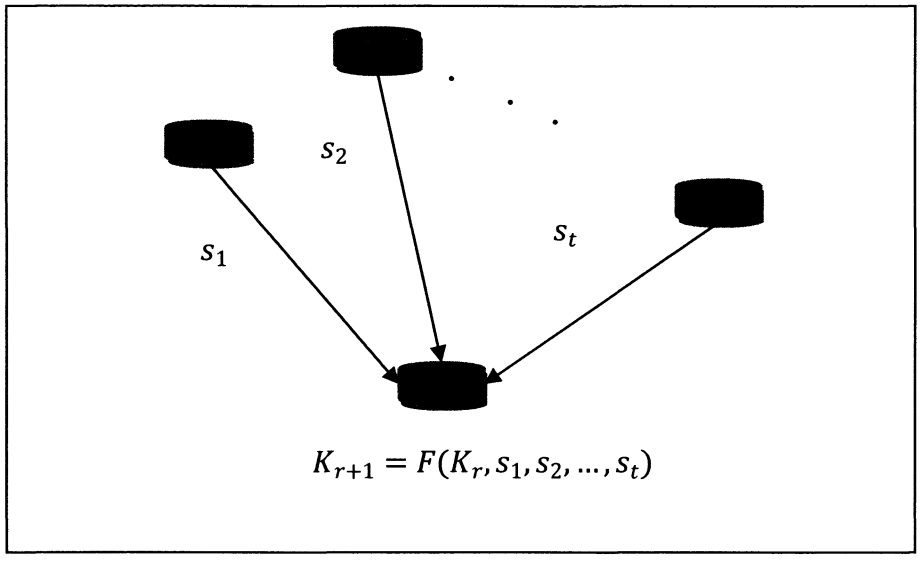

Figure 2: Key evolution with Sensor Cooperation

Pietro *et al.* in [7], [19] and [20] proposed several schemes to leverage sensor co-operation and key evolution as described above. In the [19] scheme, each sensor collects data, encrypts them and commits them to its local storage. For key updating process, at each round, a sensor picks a certain number of random sensors in the network then it sends requests to these chosen peers asking for their co-operation. Upon receiving a cooperation request, each sponsored sensor generates a HELP secret and sends it to the requested sensor. The original sensor then use all HELP secrets along with its current key to compute the next round key using one way function. The current key and HELP secrets are then deleted at the end of the round. The scheme in [20] is similar to the one in [19], however, instead of sending requests for help, each sensor here actively provides help to randomly chosen peers, offering its co-operation. These two schemes are sufficient to provide security for a UWSN, but if an ADV visits all nodes, then all the secrets are revealed.

Two other collaboration authentication schemes, called Co-MAC and ExCo, are proposed in [7]. In these two scheme, each single data item is authenticated by several sensors. In the Co-MAC scheme, at any round, each sensor computes a message authentication code (MAC) of its sensed data and sends the data to a certain number (denoted by t) of randomly chosen co-authenticated sensors in the network. Upon receiving the data, the received sensor computes a MAC using its own key and saves it to its storage. At the end of each round, the sensor computes the next round's key using its current key in addition to the information from the sensors it receives data from. In this scheme, t+1 MACs are needed to verify one data item. The ExCo scheme enhances the Co-MAC scheme by bundling all received MACs into one authentication tag. In order to win against ExCo, an ADV needs to authenticate not only t+1 MACs but also all the MACs from the sensors that send data to itself and to its co-authenticators. From the simulation results, Co Mac and ExCO are successful to provide both forward and backward security, however, they are most effective when only a portion of the sensors in the network are corrupted. The data survival probability is decreased sharply when more than 80% of sensors in the network are corrupted, and the survival probability drops to zero when ADV visits every single node.

Moreover CoMAC and ExCo suffer from Path-Based DoS (PDoS) attacks. Specifically, if an ADV compromises sensors, then it can inject traffic as fake MACs into compromised sensors to waste energy into useless traffic [25]. As a result, the lifetime of UWSN is significantly decreased. Acquiring Authentic Data (AAD) method was proposed in [25] to fight against these attacks. AAD exploits Bloom filter to ensure

the query result is correct. It also limits the number of hops of cooperated sensors that sensor will send the authentication tag to mitigate the energy consumption in case of PDoS attacks. The authors proved that AAD can provide superiority over ExCo in term of sensor compromises and energy consumption. However, AAD scheme performs mainly depending on the static information of sensors' locations.

Another research in [8] examined intrusion-resilience in Mobile Unattended WSNs. In mobile UWSNs, sensors move randomly with a certain mobility model. The purpose of sensors' moving with a mobility model is to provide uniform coverage and the variability in sensors' neighbours. The authors proposed a cooperative protocol that makes use of sensor mobility such that sensors are able to recover security after being compromised. The protocol obtains forward secure by using key evolution via one -way function. The backward secrecy is obtained by exploiting the randomness for sensor peers. At any round, each sensor moves according to the chosen mobility model. When sensor arrives at its new location, it picks a new secret and broadcasts it to its neighbours. Then sensor generates a new key based on its new secret. After that, sensor senses data, encrypts it using its current key and stores it to its local buffer. During the round time, sensor receives several cooperation secrets from random peers. At the end of any round, sensor generates the next round key using the current one along with the cooperated secrets from peers, and then it deletes the obsolete key. In this protocol, a sensor can also recover its secure by moving to a new state if it receives at least one cooperation secret from a non-compromised peer. The results show that the protocol exploited the distributed fashion to provide the network secrecy with very low over head. This protocol

is more efficiency than DISH and POSH protocols as presented in [19] and [20] respectively.

2.2 Secret sharing

Secret sharing is a way to distribute a secret among a group of participated nodes, each of them holds a share of the secret. A sufficient number of shares are needed to be combined together in order to recover the secret; each single share is useless on its own.

Particularly, a (m,n) secret sharing scheme divides some data into n shares such that at least m out of n shares are required to restore the data; n is the number of participated nodes and m is a threshold number [26] [27]. If an ADV has not corrupted enough shares then it cannot reveal the data. Even ADV has complete knowledge of m-1 shares, it still cannot reconstruct the original data. A well-known secret-sharing scheme is Shamir's Secret Sharing. It is an (m, n) secret-sharing scheme based on polynomial interpolation.

Recently, several techniques based on secret sharing have been proposed to defeat ADV in UWSNs. The common idea is using sensor cooperation in the network, or creating a dependency of data. For instance, if a node collects data and stores it individually and an ADV visits that node, there is a high risk that the data will be corrupted. But if sensors share a dependency of data with other sensors, then it increases the chance for data to survive an ADV attack.

[9] proposed a scheme for secure distributed data storage in a UWSN that makes use of a secret-sharing scheme and the Reed Solomon code to encode the sensed data into many shares. The idea of the scheme is as follows: at each time interval, a node generates

a random session key. It then uses this session key to compute the keyed hash value of the collected data to protect data integrity and encrypt data to protect data confidentiality. The node also encrypts the session key, then employs the (m, n) secret-sharing scheme to encode this secret key into n shares, and employs the (m, n) Reed Solomon scheme to encode secret data into n shares. Next, the node randomly select n neighbours and distributes a randomly chosen distinct share of secret keys and a randomly chosen distinct share of secret data. The packet is encrypted using pair-wise keys between the two nodes. In this scheme, if a node is compromised, the data is not revealed. One data only is revealed when the number of captured shares is greater than the threshold m. Similarly, [28] exploits computational secret sharing to attain data survivability. It also utilizes network coding approach to improve communication efficiency.

Other secret-sharing schemes are proposed in [10] and [29]. [10] makes use of hop-bounded secret-sharing to save energy consumption. Nodes move shares to a certain bounded hops to limit unnecessarily moving. Usually, shares are moved within the networks' radius. The authors also enhance the scheme by employing multi-level secret-sharing to reduce communication overhead and improve network security. The idea is to move the shares to different hop levels. For example, in a reputation evaluation scheme (e.g., [30]), there are different levels of nodes: some nodes are of a higher reputation than others in terms of security. The incentive is to move more shares to higher reputation nodes and fewer shares to lower reputation nodes. [29] uses homomorphic secret-sharing to achieve secure and efficient data aggregative retrieval for a cluster-based UWSN. In the proposed scheme, a secret is encrypted via homomorphic encryption. The purpose of

homomorphic encryption is that if there is any manipulation of the original data, there is a corresponding manipulation of the transformed data.

[13] utilizes a secret-sharing technique and Discrete Logarithm Problem (DLP) in heterogeneous UWSNs. The authors used two tier architecture network. One is lower tier which consists of many constrained-resource sensor nodes, the other is upper tier which contains master nodes with higher resource. Sensor nodes perform sensing data from environment, whereas master nodes play a role as a cluster head in a cluster network, collecting data from sensors and providing data to the collector. Master nodes are assumed to be tamper resistant. Thus, the authors only consider data storage protection for sensor nodes in UWSN. At each round, sensor employs secret sharing scheme together with DLP to provide integrity of the distributed shares. In this storage scheme, the data is divided into n shares and then additional n verification messages are generated. If ADV have not corrupted enough sensors to reveal sufficient shares (threshold m) then data is not damage.

In conclusion, while secret-sharing schemes improve data security and increase data survival rates, their drawback is the introduction of computation overhead.

2.3 Data moving

In WSNs, an ADV can capture node, take over it and spy on the stored data. If the collected data remains at a fixed location in the network then the Adversary can easily achieve its goal. Data moving is to travel data around the network to decrease the probability that an ADV can find the target data.

[31] investigated simple evasive data storage in sensor networks. The idea is to move data around according to a Choice function to improve data security. Specifically, node A senses data DAT and then chooses another node in the network to become a new home for DAT. First, node A sends an evasion request to its neighbours. Upon receiving the request, each neighbour node sends a reply to A indicating that it participates into the data evasion process. Node A receives responds and executed the Choice Function to choose a node to move DAT to. Finally, node A sends DAT to its new home. Several Choice functions are proposed in the paper, one example of them is UVicinity, i.e. node A chooses node B among evasion participated nodes with uniform probability. At the end of the data evasion process, all data items in the network are displaced and an adversary cannot know where a specific data item is relocated. In this strategy, if an ADV has accessed data at a node, it must corrupt other sensors in order to maintain its access to the sensor data. As a result, Evasive Data Storage effectively increases data security by mitigating the effect of an adversary even it has accessed the data once. However, this strategy introduces energy consumption and network overhead due to the demand for diligent choices of moving data process.

[6] proposed some simple protocols that leverage data migration and distribution to provide data survivability against an Eraser ADV in UWSNs, without using cryptography. An Eraser ADV is an ADV that aims to prevent certain target data from reaching the sink. It corrupts sensors and tries to delete data before the sink visits. Simple protocols to fight this type of ADV are Move-once (MO) and Keep-moving (KM). In MO, immediately after a node collects the data, it moves data to a randomly picked

sensor, where the data stays until the sink visits. In KM, the collected data is moved continuously; the data stays at each randomly chosen sensor for a fixed period of time (a time interval) and is then moved to a new one. The authors also applied cryptography on these two strategies to provide data confidentiality (i.e., the collected data is encrypted before it travels around the network) [14]. MO is more effective than KM when an ADV searches and erases less than about 80% of data, whereas KM can increase data survival for a longer time. These two schemes are simple and effective but they can only protect data against Eraser ADVs.

2.4 Data replication

Data replication is to create more than one copy of the information. By so doing, it introduces redundancy into the network. The purpose is to provide data reliability.

It is apparent that WSNs and Mobile Ad Hoc Network (MANETs) share several common features. Hence, many related works of WSN are inspired from MANETs research results [4]. Data replication has been studied in MANETs to maintain data availability and shorten response time for database operations [37].

Hara, et al. in [32]-[37] proposed simple replica algorithms based on the access frequency and the network topology to improve data accessibility in MANETs. In these algorithms, each mobile node allocates the original data and replicas with the order of access frequencies. The result shows that any node in the network is able to access the closest replica.

Another data replication scheme in MANETs was proposed in [38]. Here, the authors introduced a game theoretic mechanism technique which is developed with efficient and

effective data replication in order to overcome the ad hoc data replication problem in which each server allocates replicas for its own selfish by misrepresenting their preferences. The authors derived a polynomial-time allocation such that replicas are allocated to mobile host according to reported valuations.

[39] evaluated the accessibility of data via replication when a MANET is partitioned. It shows that the number of its replicas, the network density, and the transmission range have an significant effect on the ability to access the data.

In the most recent research result, [40] presented robust data replication algorithm (RRA) to guarantee the constant data availability. The idea of RRA is to duplicate data over replica- nodes. In this method, a node which has the recent updated of the critical data is referred as the replication manager. The replication manager runs the RRA algorithm regularly to ensure the availability of data.

Inspired from the researches of data replication in MANETs above, data replication is used in WSNs to provide network reliability and to increase chances of data survival. If a node only stores its collected data by itself and that node is compromised, then the data is corrupted. One solution is to replicate data and distribute them throughout the network. Then, if some nodes are compromised, only some of the replications will be corrupted. In such a situation, there is a greater possibility that some nodes will not be corrupted and the data will survive [29]-[32].

Kamra, et at. in [41] proposed a replication scheme for maximizing data persistence in WSNs. The idea is replicating data using Growth Codes to increase data availability and efficiency. Growth Codes is a linear coding technique that encodes data using a

distributed process to guarantee the ability for the sink to recover data from codeword symbols. Furthermore, it is as efficient as the sink is able to decode the data even when it only received a small number of codewords. The result shows that the proposed scheme successfully protect data in the presence of nodes failures and it is extremely useful in WSNs applications for natural disasters such as floods, fires or earthquake .

Replication of plain data in UWSNs was introduced in [4] and [6]. A node senses data, creates multiple copies of it and distributes data copies into the network. An Eraser ADV, as described in section 2.3, must delete all these data copies in order to achieve its objective. The authors proved that the survival probability of at least one copy is increased noticeably with the number of replicas.

[9] presented Replication Based Scheme with encryption to protect data in UWSNs. In this scheme, each node, after collecting data, randomly selects n of its neighbour nodes (n is equal or less than the total number of its neighbours). For each neighbour, it sends one replica of data. The packet is encrypted using the pair-wise key between the two nodes and includes the node ID and sequence number to differentiate the data. The drawback of this scheme is that a node collects data and distributes its replicas only to its neighbours, thus easily enabling an ADV to find all data locations.

In summary of the existing works we have described, Cryptography techniques help to protect data against an ADV, but not as anticipated. Conventional cryptography has too many vulnerabilities and therefore cannot adequately protect data against a powerful ADV. Secret-sharing, while promising, requires a lot of sensor computation. Data moving does not increase the data survival rate because the amount of destroyed data

remains unchanged [10]. While the ADV might not achieve its goal, we still lose data, which could reduce network stability. Sensor cooperation to spread the data within the network is required to increase the chance of data survival. The drawback of this strategy is increased network overhead, storage and energy consumption, and therefore we should use this method appropriately, exploiting it to improve data survival while maintaining the network's workload balance. The following chapters present our proposed strategy which based on sensor cooperation to randomly distribute to data replications within the network.

Chapter 3: Basic Model

In this chapter, we will first describe a model for unattended wireless sensor networks (UWSN), in which we will overview the method whereby sensors are deployed, delineate sensor capabilities, and outline sensor communications. Then, we will describe an adversary (ADV) model and show how an ADV strategy can be used to attack the network. Finally, we will present the random distribution of collected data, indicating how our network may succeed in the field and detailing how a basic proposed strategy can protect data against ADV.

3.1 Network Model

In this thesis, our assumptions are derived from those in [6] and [7]. Specifically, we will be examining a homogeneous UWSN, which consists of N sensor nodes randomly distributed over a geographical area. Nodes are stationary and each one has a unique ID. All of them are identical in terms of processing power and have a common transmission radius. Immediately following network deployment, each sensor creates a list of its neighbours which contains all nodes in its transmission range. Communication mode between sensors is local broadcast, with each sensor being able to communicate only with its neighbours.

Sensor nodes collect data periodically, with a global pre-defined parameter t_{col} as the time interval between successive data collections. All sensors' clocks are synchronized via any well known technique in [43]. Data sensed by each node at a time is organized in one data unit (also called one data item, or one packet) of a fixed size and has a unique ID. Each node obtains data once per time interval, stores it, and may broadcast it to neighbouring nodes within its transmission range. When receiving the broadcasted data, the neighbouring nodes may store it and/or continue forwarding it into the network. Data can be distributed in the network without bounded hops. The rules for storing and forwarding data will be defined in our proposed algorithm in section 3.4.

The network is left unattended for the majority of the time. It is continuously connected and does not experience failure. An iterant sink (or collector) visits the network periodically, with t_{sink} as the period of time between sink visits. The sink is assumed as a trusted entity. When visiting the network, it acquires collected data from every sensor and then securely resets the entire network. We consider a network such that any sensor is able to communicate with the sink to provide gathered data on demand. Each node has sufficient memory to accommodate the collected data between sink visits. The sensor storage is erased immediately following off-loading of data to the sink and there is no longer any further formal memory modification from the network respect. It should be noted that, in this thesis, we are not concerned with communication between the sink and sensors, nor are we concerned with energy consumption, though we try to reduce it by minimizing the number of transmissions.

3.2 Adversary Model

Our adversary model is also based on the assumptions described in [6] and [7]. We will take for consideration an ADV which is aware of the target data. It aims at corrupting sensed data between sink visits and so tries to prevent the original collected data from reaching the sink. The ADV is mobile. It knows the entire network strategy and is able to roam the network between sink visits, corrupting any sensor it wants. It does not modify the network topology, routing strategy or sensors' program because it wants to stay undetected and be invisible to the collector. It only "attacks" the sensors and modifies data content, and possibly all sensors in the entire network, when t_{sink} is large.

First, the ADV defines the target node and the target data. For the ADV, any data is a potential target. Let A be the node targeted by the ADV, and t_{target} be the time the target data is collected by node A. Then, at time t_{ADV}, which is defined as being equal to t_{target} plus Δt units of time ($t_{ADV} = t_{target} + \Delta t$), it starts corrupting the network by randomly visiting one node after another. The time it takes to corrupt one node is defined as δ units of time. We assume that the ADV: (1) is capable of corrupting maximum N nodes (network size), (2) is able to remember the nodes it has visited, and (3) visits each node only once. It has enough power to roam the network from the time t_{ADV} until the time the sink shows up. For each visited node, the ADV compromises the sensor by: (1) reading the sensor's storage, (2) monitoring the sensor's communications, and (3) replacing the original data with an arbitrary value.

In this model, we focus on scenarios where the ADV only modifies data content but does not modify source or packet IDs. Furthermore, the ADV does not corrupt a node while it is collecting or broadcasting data.

3.3 Defence Strategy

Since the ADV is focused on modifying data, encryption is a natural defence, as presented in Chapter 2. However, in this thesis, we propose a different approach which is based on the data forwarding rule that yields a random distribution of the data items to provide data survivability. Data survivability is defined as data survival at the time of sink arrival. In other words, when the sink arrives to collect the data, it is able to recognize the original and valid data. As this is our primary motivation, we have chosen random distribution to replicate data.

As an introduction to the approach, our basic ideas can be delineated as follows:

(a) Random Data Forwarding: Sensors collaborate to propagate the collected data within the network. Details will be presented in section 3.4.

(b) How the sink determines which version of data is valid: When the sink gathers all sensed data, it does not know whether the data is corrupted, it thus compares the various versions (i.e., the original one as opposed to the corrupted one). For a specific data item, if ADV has not corrupted enough nodes, then the majority of the collected versions are original and reliable. On the other hand, if the majority of the nodes are corrupted, then the data are no longer reliable.

(c) How our method outsmarts the ADV action: Analyses and results are demonstrated in section 3.5 and 3.6.

3.4 Random Data Distribution (RaDD)

Random Data Distribution (RaDD) can be loosely defined as a method to replicate sensed data within network. The network model consists of several data which are sensed by all sensors. We do not know which data is potentially targeted by ADV. Thus, RaDD algorithm is applied to any collected data item. Without loss of generality, we describe the storing and forwarding rules applied to one single data item which is called the target data (we use the words "data" and "target data" interchangeably). When collecting a data item, a node stores it and forwards it to its neighbouring nodes. When receiving one data item from a neighbour, a node can either store the data locally (with probability P) and/or forward it to its neighbours (with probability Q), with P and Q as pre-defined probability thresholds.

Each sensor is equipped with a Pseudo Random Number Generator (PRNG) which enables it to generate numerical values. Specifically, all sensors store P and Q at the time of deployment, and then use PRNG to generate numbers to be compared with P and Q. Depending on the values randomly generated, a decision will be made to (1) store data, (2) forward it, (3) store and forward it, or (4) discard it. It should be noted that sensor nodes make decision independently. Furthermore, for each node, the decision made for one data item is also independent with others. Table I describes these four decisions

Table I: Decision based on storing and forwarding rules

Decision		Store (Yes/No?)	
		Yes	**No**
Forward (Yes/No?)	**Yes**	Store and Forward	Forward only
	No	Store only	Not Store or Forward

The decision to store and forward data, though from a single node, comprises two independent or separate acts. If a node decides to forward the data to its neighbouring nodes, it may wait for a random time to broadcast (i.e., forward the data). The randomness inherent in this type of forwarding time is intended to keep data travelling between sink visits. In other words, if all nodes received data then forwarded it right away, in our network model it takes only a few seconds to complete, then the RaDD is done. In such a case, if an ADV visit a node after RaDD, it is certainly able to capture the target data successfully. For this reason, forwarding data based on a random time is preferable in exploiting randomness. Recall that target node A senses the target data at time $t = t_{target}$, commits the data to its storage, and broadcasts it to its neighbouring nodes. Tables III and IV describe a RaDD pseudo-code run by a sensor node when it receives a broadcasted data packet from its neighbour. Notation of variables used in all algorithms in this thesis is given in Table II.

Table II: Notation of variables used in algorithms

Symbol	*Description*
P	Storing probability threshold
Q	Forwarding probability threshold
t	Current time
i	Current node
$i\text{-}1$	Previous hop
j	Current received packet
$t_{rcv_{i,j}}$	Time node i receives packet j
$t_{str_{i,j}}$	Time node i stores packet j
$t_{fw_{i,j}}$	Time node i forwards packet j
t_{sink}	Time the sink arrives
t_1, t_2	Forwarding time range
$t_{rebrd_{i,j}}$	Time node i re-broadcasts packet j
$PktSet_i$	List of packets node i has seen (sent and received packets)
S	List of all sensor nodes in the network
C_{ADV}	List of nodes ADV has corrupted
t_{ADV_i}	Time node i is corrupted
$StrCor_{i,j}$	If $StrCor_{i,j}$ is True then packet j which is stored at node i is corrupted. $StrCor_{i,j}$ is False by default.
$BrdCor_{i,j}$	If $BrdCor_{i,j}$ is True then node i broadcasted a corrupted version of packet j. $BrdCor_{i,j}$ is False by default.
$ReBrd_{i,j}$	If $ReBrd_{i,j}$ is True then node i re-broadcasts packet j . $ReBrd_{i,j}$ is False by default.

Table III: Main function

MAIN FUNCTION: Random data distribution algorithm

```
If (j not in PktSet_i ) then {          /* if node i has not seen packet j before */
    PktSet_i += {j}   /*node i noted j as a seen packet*/
    Generate two random numbers X and Y between 0 and 1
        If ( X ≤ P ) then {  /* node i decides to store the received packet*/
            t_str_i,j = t    /*node i stores packet j at time t_str_i,j */
            If ( Y ≤ Q ) then { /* node i decides to forward the received packet*/
                Forward (i, j)   /* call Forward function*/ }
        } else { /* node i decides not to store the received packet*/
            If ( Y ≤ Q ) then {  Forward (i, j) /* decide to forward */ }
            else { Discard (i, j)  /*decide not to store or forward */  }
        }
}
```

Table IV: Forward function

FORWARD FUNCTION: Random data distribution algorithm

```
Forward (node i, packet j) {
    Generate a random number k in (t_1, t_2)
    If (t_rcv_i,j + k < t_sink) then { /* if the scheduled time is before the sink arrival*/
        t_fw_i,j = t_rcv_i,j + k
    /* node i forwards packet j at time t_fw_i,j equal to the received time plus a random time*/ }
}
```

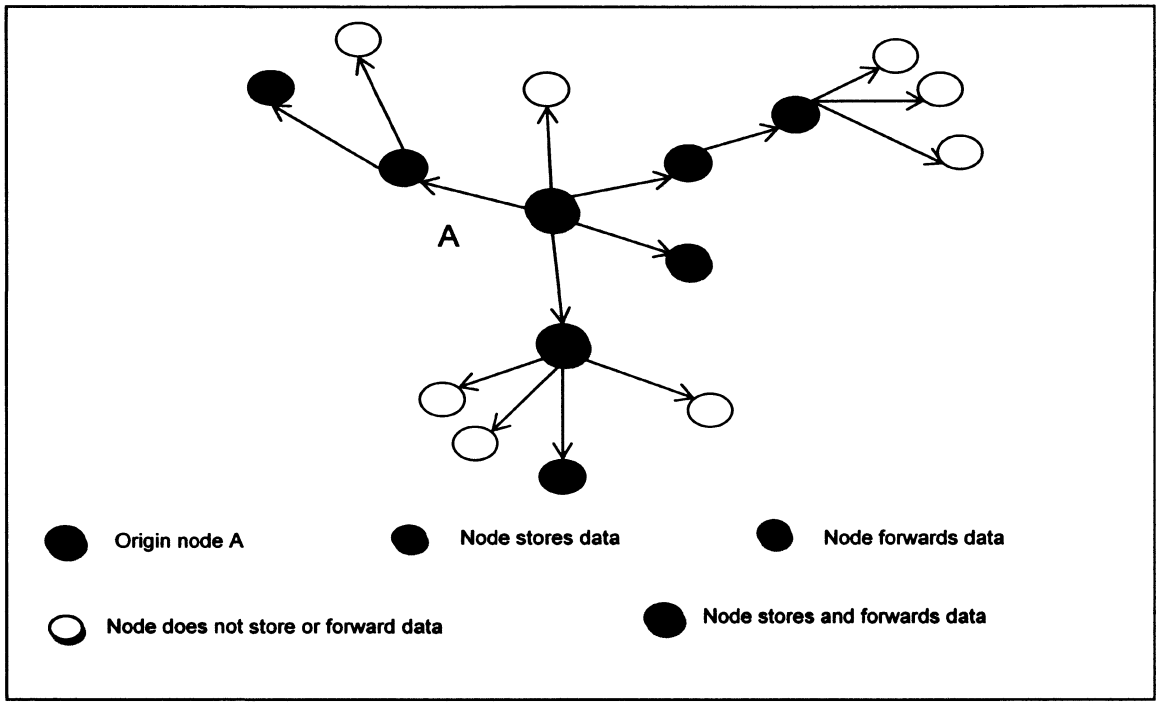

Figure 3: An Example of Random Data Distribution

Since the decision to store a data item, the decision to forward it, and the time to broadcast it are all randomly made, the data will be available at arbitrary nodes within the network. Figure 3 shows an example of Random distribution of a data collected by node A. The origin node collects a data item and broadcasts it. Purple circles represent for nodes that decide to forward node A's data. Blue circles represent for nodes that decide to store it. Blue circles with purple outlines represent for nodes that decide both store and forward it. White circles represent for nodes that decide to do nothing for this particular data of sensed by node A, however, these nodes will store and/or forward other data items in the network. Because of the random distribution of data, an encryption algorithm is not necessary but may be used in addition to our method to improve the performance. An ADV is unable to predict all the locations of the data, and can only randomly pick the nodes to corrupt in order to achieve its goal. Let S be the set of sensors in the network and

C_{ADV} be the set of nodes ADV has corrupted. The pseudo-code run by an ADV can be represented in Table V.

Table V: Adversary activity

MOBILE ADVERSARY ACITIVITY
$\boldsymbol{S = \{0, 1, 2, \ldots, N\}}$ /*set of all sensors (ID) in the networks*/
$\boldsymbol{t_{ADV} = t_{target} + \Delta t}$ /*ADV starts corrupting the network at time $\boldsymbol{t_{ADV}}$, $\boldsymbol{\Delta t}$ units of time after the target data is collected*/
$\boldsymbol{C_{ADV} = \{\}}$ /* Initialize the set of sensor ADV has corrupted*/
while $(\boldsymbol{t < t_{sink}})$ **do** { /* while the sink has not arrived*/
Pick one random node in $\boldsymbol{S \backslash \{C_{ADV}\}}$, assume node $\boldsymbol{i}$ is selected
/* ADV picks node *i* to corrupt such that ADV has not corrupted it before */
$\boldsymbol{t_{ADV_i} = t_{ADV}}$ /* ADV corrupts node *i* at time $\boldsymbol{t_{ADV_i}}$ */
$\boldsymbol{t_{ADV} += \delta}$ /*ADV finishes corrupting node *i* */
$\boldsymbol{C_{ADV} + \{i\}}$ /* add node *i* into set of corrupted nodes*/
} **end while**

Detecting corruption in a data version:

In our model, both the data and the ADV travel continuously around the network, but for completely different reasons. Whereas the data moves around to mislead the ADV, the ADV seeks out the data. Being constantly and randomly on the move, the data has a better chance of outsmarting the ADV than it would were stationary and confined to non-random, pre-determined distribution.

If the ADV corrupts a node before it receives the data, then the target data remains original and reliable. If the ADV corrupts a node after it receives the target data and

before the node forwards it, then the corrupted version of the data is broadcast and shared with the neighbouring nodes. Otherwise, if the ADV corrupts a node after it receives the target data and after the node forwards it the data, only the version stored at the corrupted node is damaged. Table VI provides a pseudo-code to evaluate whether a stored packet is corrupted or not. In this Basic model neither the sensor or the sink know whether the data is corrupted or not. The pseudo-code is for the model evaluation purpose only.

Table VI: Determine data corruption

DETERMINE DATA CORRUPTION: Random data distribution

If $(BrdCor_{i-1,j})$ **then** {/* if the received packet from a node broadcasting a corrupted version */

  If ($X \leq P$) **then** { /* if the current node decides to store the packet */

    $StrCor_{i,j} = True$ /* packet *j* stored at node *i* is corrupted */ }

  If ($Y \leq Q$) **then** { /* if the current node decides to forward the packet */

    $BrdCor_{i,j} = True$ /*node *i* broadcasts the corrupted version of pkt *j*/ }

} **else** **{**

  If $(X \leq P)$ && $(t_{str_{i,j}} < t_{ADV_i})$ **then** {

    /* if ADV corrupts node *i* after it stores packet *j* then it stores the corrupted version of *j* */

    $StrCor_{i,j} = True$ }

  If $(Y \leq Q)$ && $(t_{rcv_{i,j}} < t_{ADV_i} < t_{fw_{i,j}})$ **then** {

    /* if ADV corrupts node *i* after it receives packet *j* and before it re- broadcasts *j* then it broadcasts the corrupted version of *j*/

    $BrdCor_{i,j} = True$ }

  }

Outsmarting the ADV:

Fixing the values of the probabilities P and Q may increase the chances for the sink to collect mostly reliable data. If P is close to 1, many nodes are likely to store the data up to their buffer limit. The value of Q determines the number of broadcast messages and battery consumption. If Q is close to 1, most of the nodes broadcast, flooding the network. Therefore, P and Q should be set to appropriate values as a balance between data sharing requirements and minimizing network overhead.

One can notice the trivial case with $P = 0$: A node senses a data item and stores it locally. It may not forward it to its neighbouring nodes, none of which will store the data. ADV is likely to "catch" the target data as it is not shared within the network. Similarly, the case with $Q = 0$ (i.e., no sharing of data with neighbours) maximizes the chance of an ADV to corrupt the data.

Initial expectations of proposed method:

The main advantage of our proposed algorithm is that data are randomly distributed in the network to mislead the ADV, thereby increasing the chances for data survival. Disadvantages of this method involve an increased need for storage and communication overhead, which results in the higher network load. However, these drawbacks can be minimized by refining the appropriate P and Q.

3.5 RaDD Analysis

This section provides primitive analysis of Random data distribution algorithm. We summarize the notation which is used throughout the thesis in Table VII.

Table VII: Notation of symbols used in simulation

Symbol	*Description*
N	Network Size
N_{target}	Number of nodes storing the target data
M	Number of corrupted nodes in the entire network
M_{cor}	Number of corrupted nodes storing the target data
L	Number of nodes storing the healthy target data
$P_{H\,(\%)}$	Percentage of healthy nodes
B_M	Number of broadcast messages

From the ADV definition in the Adversary model (section 3.2), one can notice that the total number of nodes an ADV can corrupt between successive sink visits can be represented by the formula

$$M = \frac{t_{sink} - t_{ADV}}{\delta}. \tag{2}$$

Let N_{target} be the average number of nodes storing the target data, then it is linear with storing probability P and forwarding probability Q. This is because P determines the possibility to store a data packet and Q determines that number of nodes forwarding data which also results in possibility to store a packet.

An attempt was made to find the percentage of healthy data $P_H(\%)$ in the worst-case scenario, which occurs when an ADV corrupts all nodes after they receive the data, as follow:

The average number of corrupted nodes which stored the target data in the worst-case is

$$M_{cor} = \frac{M}{N} N_{target}, \quad (3)$$

which yields the number of healthy nodes that have the original target data as (also called healthy nodes or healthy data)

$$L = N_{target} - M_{cor} = (1 - \frac{M}{N}) N_{target}, \quad (4)$$

and the percentage of healthy nodes which have the target data as

$$P_H(\%) = \frac{L}{N_{target}} = \left(1 - \frac{M}{N}\right)(\%). \quad (5)$$

It can be seen from Equation 5, P_H is linear with the *M/N* ratio and is not affected by *P* or *Q*. With our defence strategy description in section 3.4, RaDD is only effective when P_H is greater than 50%. This happens when the ratio of *M* to *N* is less than 50% or, in other words, when the number of corrupted nodes is less than half of all sensors in the networks.

3.6 Simulation Results

In this thesis, we use the simulation results running from the Network Simulator NS2.34 to evaluate our algorithms in terms of data survivability. Specifically, the evaluation of the performance is based on

(1) the percentage of nodes with the original data at the sink arrival *(P_H),*

(2) the total number of broadcast messages (*B_M*), and

(3) the number of nodes that store the target data *(N_{target}).*

In this section, we will investigate RaDD results.

3.6.1 Simulation settings

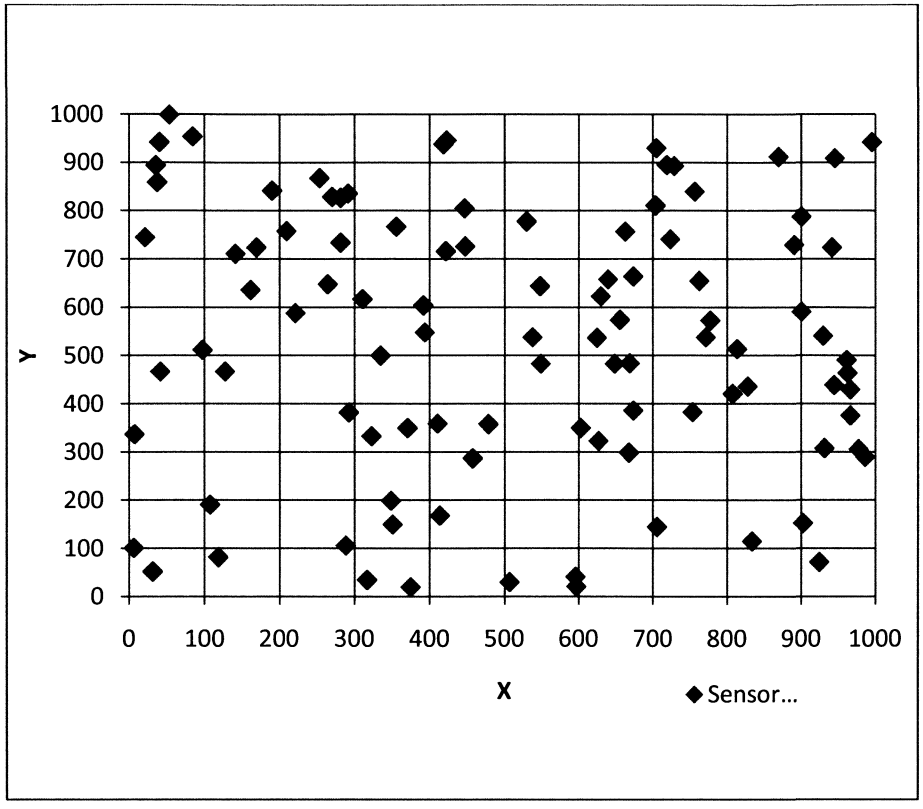

Figure 4: An example of network deployment

First, we create a network that plots random coordinates on an X-Y plane of 1000m×1000m, as depicted in Figure 4. The network size N is equal to 100. We assign each sensor with a unique ID i (e.g., i=0, 1, 2,…99). Each sensor has a transmission range of 250m. One data item is treated as a packet which has its own packet ID. Target node A has coordination (X,Y) equal to (291,836). The input parameters for simulation are summarized in Table VIII. We assume ADV starts to corrupt the network 0.1 minutes after the target node collects the data so that ADV will not corrupt the node while it sensing or broadcasting. We chose the value of δ such that if an ADV visits the network early, aim at targeting the data collected at time t=0, it would be able to corrupt all 100

nodes in the network. The values of t_1 and t_2 are arbitrarily chosen between 0 and t_{sink} such that the randomness of forwarding time is exploited and the majority of broadcasts are executed before the sink' arrival.

Table VIII: The values of input parameters used in simulation

Parameter	*Value*
N (nodes)	100
t_{sink} *(minutes)*	5.1
t_1 *(minutes)*	1
t_2 *(minutes)*	1.5
δ *(minutes)*	0.05
Δt *(minutes)*	0.1

3.6.2 Results

The simulation results are generated when implementing various case studies by varying the values for *P, Q* and *M*. For each scenario, the statistics are derived from 100 simulation runs.

a) Percentage of healthy data

Figure 5 depicts the percentage of healthy nodes *(P_H)* versus the number of corrupted nodes *(M)* in RaDD with various values of storing and forwarding probability *P* and *Q*. The simulation results exhibit a similar relationship between P_H and *M* as in Equation 5, although the simulations are a bit more optimistic, as can be seen in graphs from Figure 5. It is because Equation 5 represents the worst case scenario while the graphs do not; a node that is corrupted before it receives the target data still has the original version of data at the arrival time of the sink.

It is observable that increasing the number of corrupted nodes decreases the percentage of healthy nodes. Specifically, when the number of corrupted nodes is equal to or larger than 50, the percentage of healthy nodes drops below 50%. The increase of values P and/or Q does not improve P_H , as P_H is proportional to the ratio of M to N, as described in Equation 5. Hence, the basic RaDD cannot provide sufficient effectiveness in cases where an ADV corrupts more than 50% of the nodes in a network.

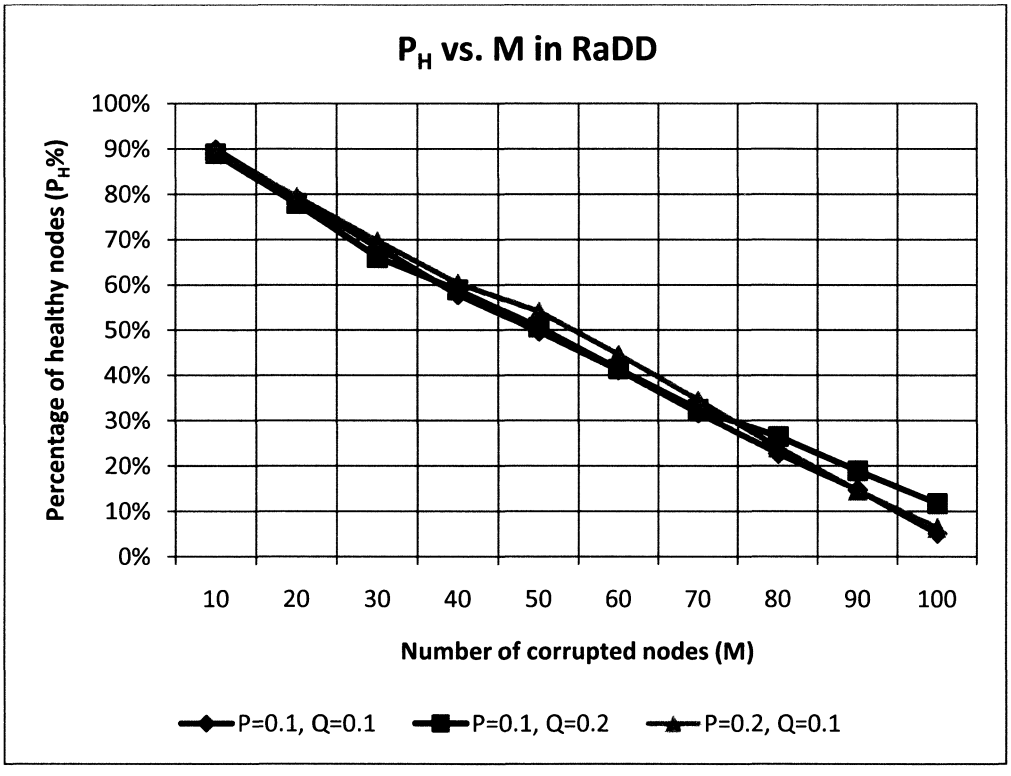

Figure 5: Healthy nodes vs. Corrupted nodes in RaDD

b) Broadcast messages

Figure 6 illustrates the number of broadcast messages B_M of the RaDD algorithm. It is apparent that the storing probability P does not affect the B_M (i.e. in the graphs, the blue line for P=0.1 coincides with the red line), whereas the number of broadcast

messages linearly increases along with the increase of the forwarding probability Q. In scenario $t_{target} = 0$, B_M is increased sharply from approximately 55 to 1488 messages when Q is varying from 0.1 to 1 (the blue line). It also indicates that B_M increases when target data is collected earlier, especially in the case of Q is close to 1. Because when target data is collected early, there is ample time for sensors to continue forwarding it within the network.

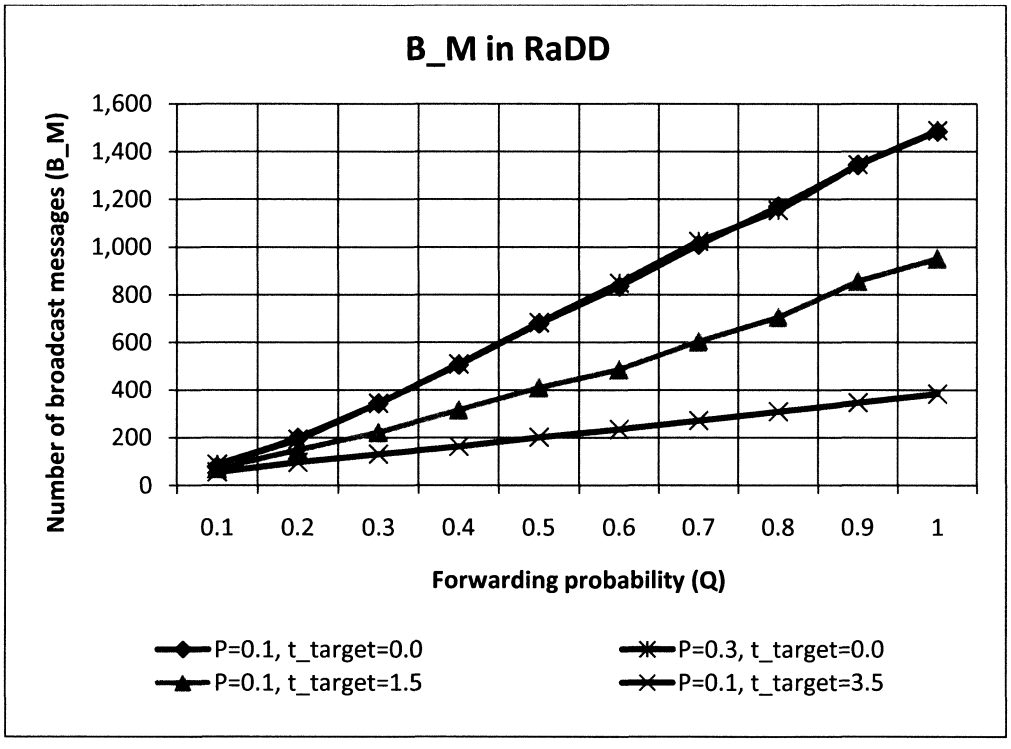

Figure 6: Broadcast messages in RaDD

c) Nodes storing the target data

The number of node storing target data (N_{target}) is determined by the P (storing probability) and Q (forwarding probability). When the storing probability is high and there are more nodes to store, the onus is on the performance of P. Alternatively, when

more nodes are broadcast and can receive data, resulting in a greater possibility for storage, then the focus is on Q. Figures 7 and 8 show N_{target} with various P and Q respectively, for scenario t_{target}=0 where there are the highest number of N_{target}.

In Figure 7, when Q=*0.1*, the number nodes storing target data is linearly increase from 4.2 (P=*0.1)* to 33.4 (P= *1,* where all nodes receiving the broadcast data will store it). When Q=*1*(all nodes broadcast), N_{target}=10.85 with P=*0.1*, N_{target}=97.93 with P=1. These simulation results make sense because when Q=1 (i.e., all nodes are broadcast), N_{target} is likely equal to storing probability P multiply by the network size N (i.e. $N_{target} \approx P \times N = P \times 100$, e.g. P=0.1 , N_{target} =10.85 ≈ 10% × 100 and P=1, N_{target}= 97.93 ≈ 100% × 100).

In Figure 8, N_{target} is increased dramatically when Q is varying from 0.1 to approximately 0.5. It then likely levels out because when Q is too large, the network produces redundant broadcasts.

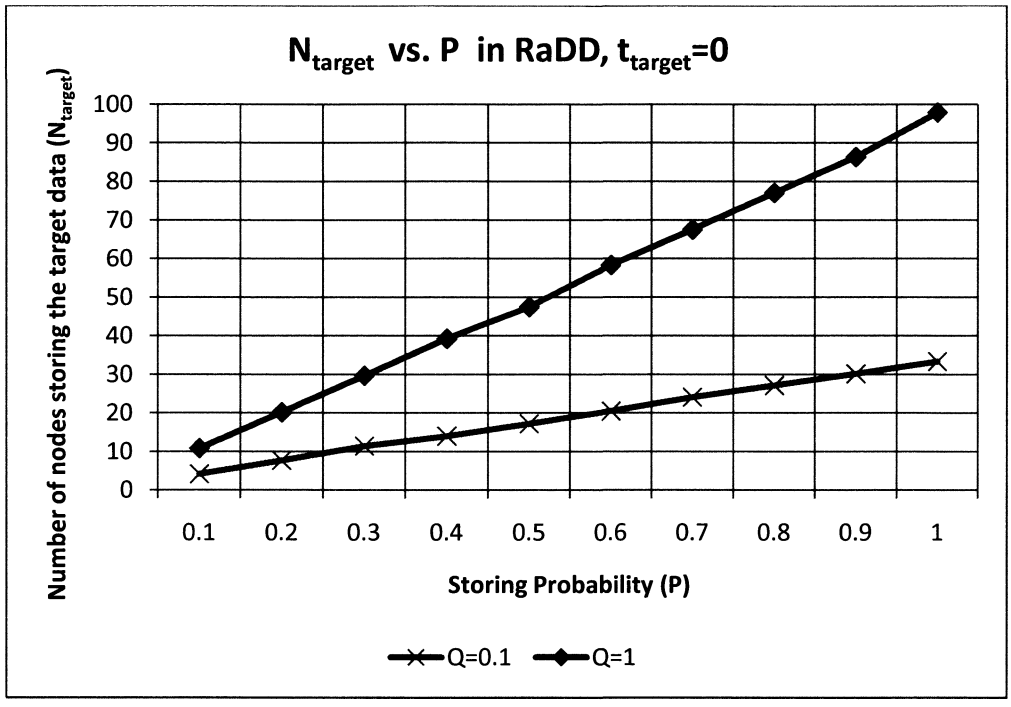

Figure 7: Number of nodes storing the target data versus *P* in RaDD

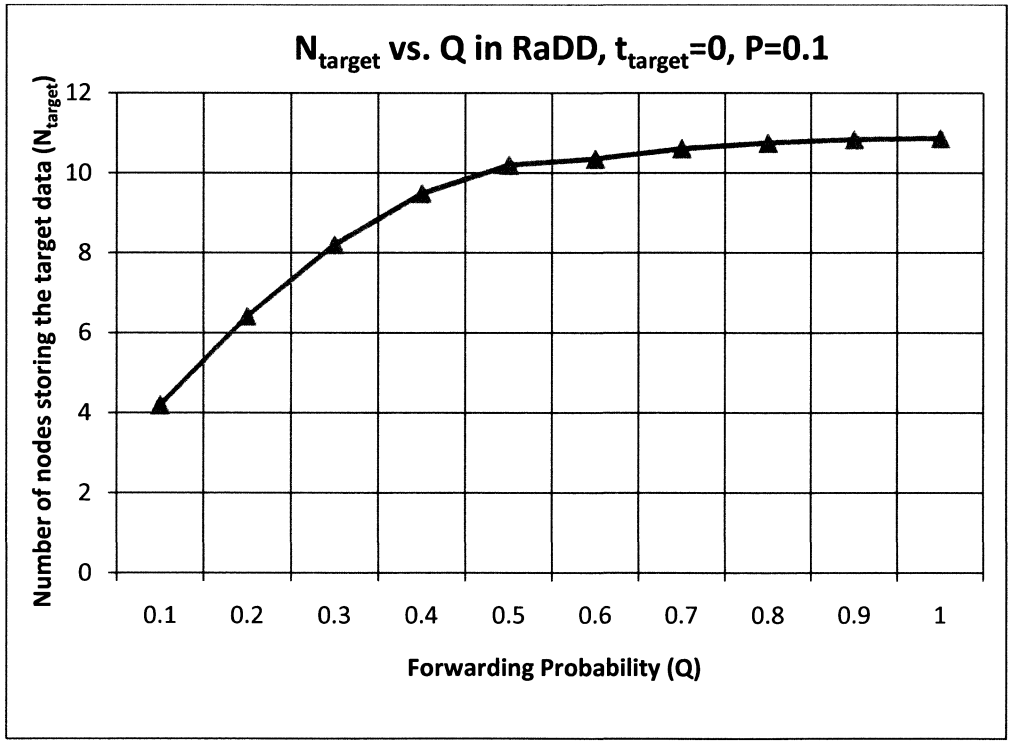

Figure 8: Number of nodes storing the target data versus *Q* in RaDD

d) Conclusion

The higher the values of P and Q, the higher the network load in terms of broadcast messages and storage. Therefore, for the most effective network, P and Q should be set to the lowest values to reduce network cost while still maintaining network reliability. From our simulation results, $P=0.1$ and $Q=0.1$ seems to be a good choice for RaDD. In this case, when the number of corrupted nodes is less than 50, the number of healthy nodes that have the original target data L varies from 2.01 to 2.42) and the percentage of healthy nodes is greater than 50% (from 57.84% to 89.91%) with the 95% confident interval fluctuating from 3% to 6% (Table IX and Figure 9). We conclude that an RaDD algorithm is effective and efficient ($P=0.1$, $Q=0.1$) if an ADV corrupts the network less than 50% of total nodes. Otherwise, a smarter strategy needs to be developed in order to defeat ADV. With this in mind, we define improved algorithms for RaDD with Damage Recovery and Damage Control to optimize healthy data in Chapter 4.

Table IX: Simulation results of RaDD with P=0.1 and Q=0.1

M	N_{target}	L	$P_{H(\%)}$
10	2.70	2.42	89.91%
20	2.70	2.13	79.11%
30	3.46	2.36	68.28%
40	3.46	2.01	57.84%
50	3.93	1.95	49.70%
60	3.97	1.63	41.12%
70	4.01	1.29	32.31%
80	4.17	0.94	22.69%
90	4.17	0.61	14.69%
100	4.20	0.38	7.59%

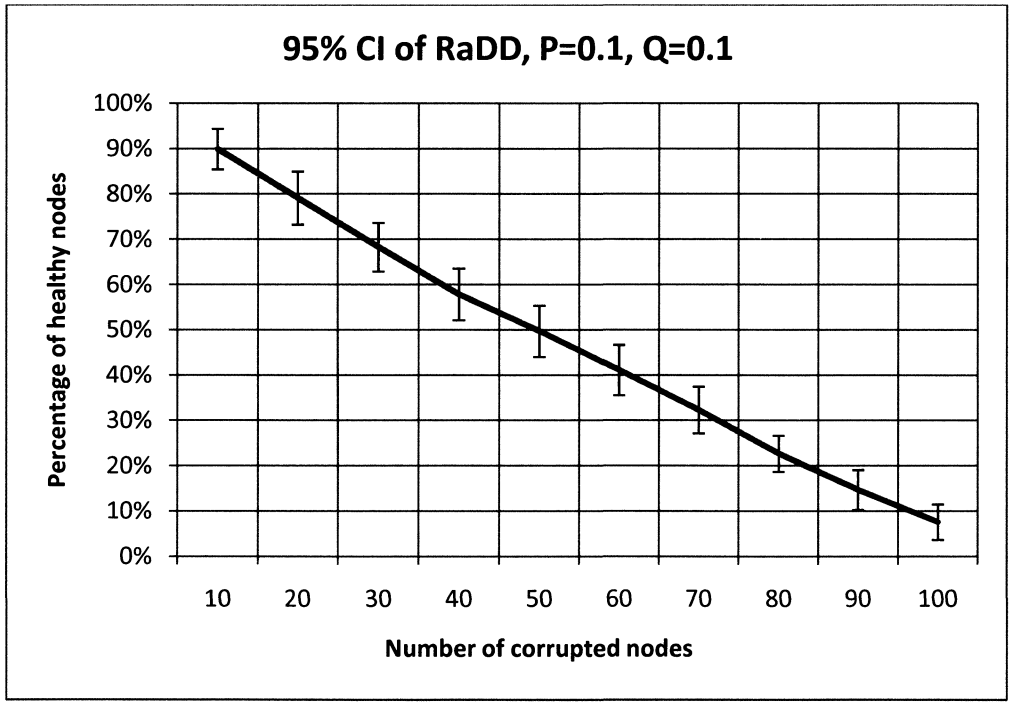

Figure 9: Confident Interval of RaDD with P=0.1 and Q=0.1

Chapter 4: Enhanced Models

As we saw in the previous chapter, Random Data Distribution (RaDD) is used to provide data survival. However, when the majority of nodes in the network are corrupted (≥ 50%), the percentage of healthy nodes may be less than 50%. When this occurs, the sink will be unable to determine whether the version of data it gathers is the original or the corrupted one. For this reason, we define the two strategies of Damage Recovery (DR) and Damage Control (DC) to enhance RaDD. Damage Recovery re-broadcasts data in a second phase, while Damage Control limits the propagation of corrupted data within the network.

This chapter will detail how enhanced strategies function. Specifically, it will show how they improve the basic RaDD by bringing the percentage of healthy nodes above 50%, even in cases where an ADV has corrupted every single node in the network.

4.1 RaDD with Damage Recovery (DR)

In Chapter 3, we assumed that only an external actor modifies a sensor's memory. The networks include sensor nodes and a sink which are internal actors. As such, they do not modify; thus, if memory is modified, it is an unwanted action from an external actor. Since ADV corrupts the network by altering data in the sensor's buffer, we assume that

the sensor is able to detect whether its stored data is corrupted. We summarize sensor behaviours based on buffer status as follows:

(1) data collection – the sensor itself writes sensed data into its buffer,

(2) data offloading – the sink securely resets the sensor's buffer,

(3) data corruption – the sensor's buffer is modified by an external entity.

It should be noted here that only the sensor itself know whether its stored data is corrupted or not based on sensor's behaviours. When a data item is sent out, the received node or the sink does not know whether this data is original or corrupted.

Damage Recovery (DR) is an additional step to RaDD that aims at increasing the number of healthy nodes by re-broadcasting the data in a second phase. Shortly before the sink's visit, all healthy nodes (or only some, if the node count is high) whose buffer has not been altered by ADV re-broadcast their stored data to their neighbours. When receiving the data a second time, a node that was corrupted can overwrite its buffer with the received version, to undo the previous ADV attack.

The network operation of RaDD with DR is separated into two phases. **Phase 1** is for the time period from zero to t_{DR}, which is defined equal to t_{sink} minus τ units of time. In this phase the network operation is only basic RaDD. **Phase 2** starts at time t_{DR} until t_{sink}, with the network operation being a combination of basic RaDD and Damage Recovery. In this phase, along with storing and forwarding rules from RaDD, all the healthy nodes of Phase 1 broadcast their original stored data for a second time for damage recovery at time t_{DR} plus a jitter. The purpose of a jitter is to avoid collision.

This is a one-time broadcast, and only nodes that have stored this broadcasted data (identified by a unique packet id) but was corrupted replace with the healthy one.

Note that in the second phase, ADV is still attacking the sensor nodes. But the short period of time between the beginning of the second phase and the sink's arrival limits the damage that could be caused, and can still significantly improve the percentage of healthy nodes. Diagram 1 shows the sequence of events

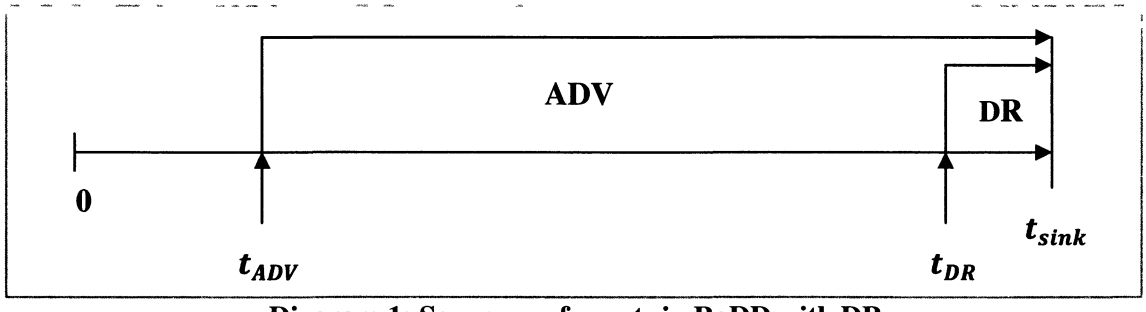

Diagram 1: Sequence of events in RaDD with DR

Tables X and XI describe the Damage Recovery pseudo-code which is run on top of RaDD. Notation was given in Table II. Note that pseudo-codes of Main function, Forward function, ADV activity and Determining a corrupted packet are the same as in the basis RaDD.

Table X: Re-broadcast function in Damage Recovery

RE-BROADCAST FUNCTION: Damage Recovery
$t_{DR} = t_{sink} - \tau$ **If** $(t == t_{DR})\&\&\ (!\ StrCor_{i,j})$ **then {** /* if node *i* stores a non-corrupted version of packet *j* at time t_{DR}*/ $t_{re\_brd_{i,j}} = t_{DR} + jitter$ $ReBrd_{i,j} = True$ /* node *i* re-broadcasts packet *j* at time t_{DR} plus a jitter (ms) to avoid collision*/ }

Table XI: Replace a corrupted packet in Damage Recovery

Replace a corrupted packet: Damage Recovery
while $(t_{DR} < t < t_{sink})$ **do {** /*DR phase duration is from t_{DR} **to** t_{sink} */ **If** $(j\ in\ PktSet_i)\&\&(ReBrd_{i-1,j})\&\&(StrCor_{i,j})$ **then {** /* if node *i* stores a corrupted version of packet *j* and receives a re-broadcast of *j*/ $t_{str_{i,j}} = t$ $StrCor_{i,j} = False$ /* node *i* replaces the corrupted packet with the healthy one*/ } **} end while**

An attempt was made to find the number of healthy nodes $L_{(phase1)}$ at the end of Phase 1 for the worst-case scenario in the following (Phase 1 is basic RaDD).

The number of corrupted nodes is

$$M_{(phase1)} = \frac{t_{DR} - t_{ADV}}{\delta}. \quad (6)$$

The number of corrupted nodes that stored the target data is

$$M_{cor(phase1)} = \frac{M_{phase1}}{N} N_{target(phase1)}, \quad (7)$$

and the number of healthy nodes that have the original target data is

$$\begin{aligned} L_{(phase1)} &= N_{target(phase1)} - M_{cor(phase1)} \\ &= \left(1 - \frac{M_{(phase1)}}{N}\right) N_{target(phase1)}. \end{aligned} \quad (8)$$

Assuming that all healthy nodes from Phase 1 will re-broadcast data for Damage Recovery, the number of nodes that will take part in re-broadcasting is $L_{(phase1)}$. From Equation 8, $L_{(phase1)}$ is proportional to $N_{target(phase1)}$ which is linear with P and Q as mentioned in Chapter 3. Increasing values of P and/or Q might increase the number of $L_{(phase1)}$ to participate in the DR phase in order to improve the healthy data.

4.2 RaDD with Damage Control (DC)

In this algorithm, as in section 4.1 regarding Damage Recovery, we make the same assumption that a sensor node is able to detect corruption (i.e., it can detect whether or not corruption is present in a packet of the node). The Damage Control (DC) rule is in addition to the basic RaDD rules, whereby if a node receives the data and decides to forward it, it will cancel the broadcast operation in order to limit the propagation of the corrupted version, if the node is corrupted before the actual broadcast operation time. Thus, if a message is broadcast, it would be a healthy version. The DC rule is applied to nodes:

(1) which have received data;

(2) which are scheduled to forward it, and;

(3) which are corrupted at time between the received and forwarded time

For instance, node i receives packet j at time $t_{rcv_{i,j}}$, schedules to forward it at time $t_{fw_{i,j}}$ and is corrupted at time t_{ADV_i}, with $t_{rcv_{i,j}} < t_{ADV_i} < t_{fw_{i,j}}$, then node i will cancel the forwarding operation. The sequence of events is shown in the diagram 2.

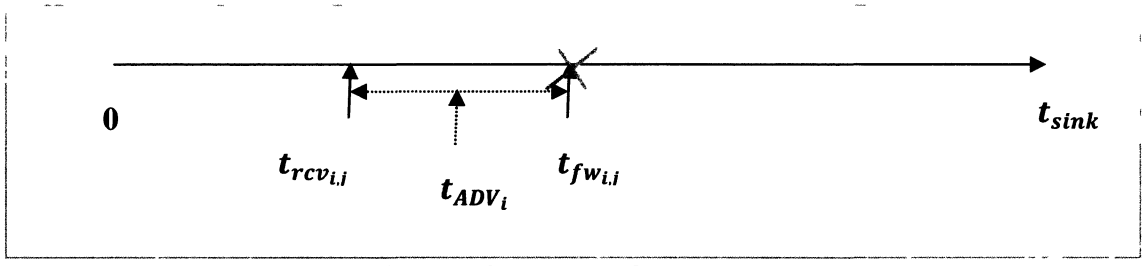

Diagram 2: Damage Control Rule

Additionally, nodes storing corrupted data will replace their stored data with the healthy version when they receive a broadcast message. The DC rule intuitively decreases both the number of broadcast messages and the number of nodes storing the target data while increasing the percentage of healthy nodes. The following tables describe pseudo-code of RaDD with Damage Control algorithm with notation given in Table II.

Table XII: Main function of Damage Control

MAIN FUNCTION: RaDD with Damage Control algorithm

```
If (j in PktSet_i ) then {    /* if node i has seen packet j before */
    If (StrCor_{i,j}) then {
        /* if node i stores a corrupted version of packet j then it replaces with the healthy one*/
            t_{str_{i,j}} = t
            StrCor_{i,j} = Fasle
    } else Discard (i,j)
} else { /* if this is a new packet*/
        PktSet_i += {j}   /* node i noted j as a seen packet */
        Generate two random numbers X and Y between 0 and 1
            If ( X ≤ P ) then {   /* node decides to store the received packet */
                t_{str_{i,j}} = t   /*node i stores packet j at time t_{str_{i,j}} */
                If ( Y ≤ Q ) then { /*node decides to forward*/
                        Forward (i,j)   /* calls Forward function */ }
            } else { /*node decides not to store*/
                    If ( Y ≤ Q ) then Forward (i,j) /*decide to forward*/
                    else Discard (i,j)   /*decide not to store or forward*/
                    }
        }
```

Table XIII: Forward function of RaDD with DC

FORWARD FUNCTION: RaDD with Damage Control algorithm
$\boldsymbol{Forward\ (node\ i, packet\ j)}$ {
Generate a random number k in (t_1, t_2)
If $(t_{rcv_{i,j}} + k < t_{sink})\&\&((t_{ADV_i} < t_{rcv_{i,j}})\|\|(t_{ADV_i} > t_{rcv_{i,j}} + k))$ **then {**
/*if packet j is not corrupted by the time of forwarding*/
$t_{fw_{i,j}} = t_{rcv_{i,j}} + k$
/* node i forwards packet j at time $t_{fw_{i,j}}$ equal to the received time plus a random time*/ }
else $\boldsymbol{Discard\ (i,j)}$
}

Table XIV: Determine data corruption in RaDD with DC

DETERMINE DATA CORRUPTION: RaDD with Damage Control
If $(X \leq P)$ && $(t_{str_{i,j}} < t_{ADV_i})$ **then {**
/* if ADV corrupts node i after it stores packet j then it stores the corrupted version */
$StrCor_{i,j} = True$
}

4.3 RaDD with DR and DC (Combined model)

A further possible algorithm to mitigate damage from corrupted nodes is the combined model. The combined model is the implementation of Damage Recovery running on top of RaDD with Damage Control. The operation of this algorithm is similar to that for RaDD with DR which consists of two phases. In Phase 1, the network operation is RaDD

with DC (from time zero to t_{DR}). Phase 2 is the Damage Recovery phase (from t_{DR} to t_{sink}), with the operation of DR on top of RaDD with DC.

The expectations from this combined DR and DC model are:

1) Compared to RaDD with only DC: It would increase the percentage of healthy data at a cost of more broadcast messages by adding the DR phase.

2) Compared to RaDD with only DR: It would limit the spread of corrupted data into the network. This action would save broadcast messages as well.

3) The number of nodes storing the target data N_{target} is the same as for RaDD with DC.

In general, this combined model is expected to increase the percentage of healthy data in order to defeat ADV. However, we will see how it works with the simulation results, which will be described in Section 4.4.

4.4 Simulation Results

In this section, we evaluate enhanced models using the same simulation settings as in Chapter 3.

4.4.1 RaDD with DR

We set τ equal to 0.2, and thus phase 2 starts at time $t_{DR} = 4.9$ mins (one can try to assign different value of τ such that ADV does not have much time to corrupt majority of the nodes). In the simulation results below, we use all nodes having healthy data at time t_{DR} to take part in re-broadcasting for damage recovery, thus the number of re-broadcasted nodes is $L_{(phase1)}$. The reason for this is that we try to use as small a value of

P and Q as possible (to save network load as detailed in Chapter 3). Moreover, the number of healthy nodes of Phase 1 is small ($L_{(phase1)}$) is only from 0.64 to 2.66 in case P=0.1, Q=0.1, as shown in Table XV). If we do not use all of them, there will be insufficient nodes to spread the healthy data for recovery in order to defeat the ADV.

Table XV: The average $L_{(phase1)}$ in case P=0.1, Q=0.1

t_{target}	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
$L_{(phase1)}$	0.64	1.04	1.47	1.67	1.98	2.09	2. 34	2.35	2.37	2.66

a) Percentage of healthy data

The effects of P and Q are taken into consideration as, contrary to RaDD, the changing of the values P and/or Q here affects the result of P_H.

Effect of parameter *P*

Figure 10 shows that as P increases, there is a substantial increase of P_H – namely, approximately from 10% to 61% in the case of Q=0.1, and ADV corrupts all nodes (the worst case). This dramatic increase is because the value of P affects the number of nodes storing data. Along with the increasing of P, the number of nodes storing the data $N_{target(phase1)}$ increases, thus it results in higher $L_{(phase1)}$ nodes to re-broadcast Equation 8), and the percentage of healthy nodes P_H is likewise increased.

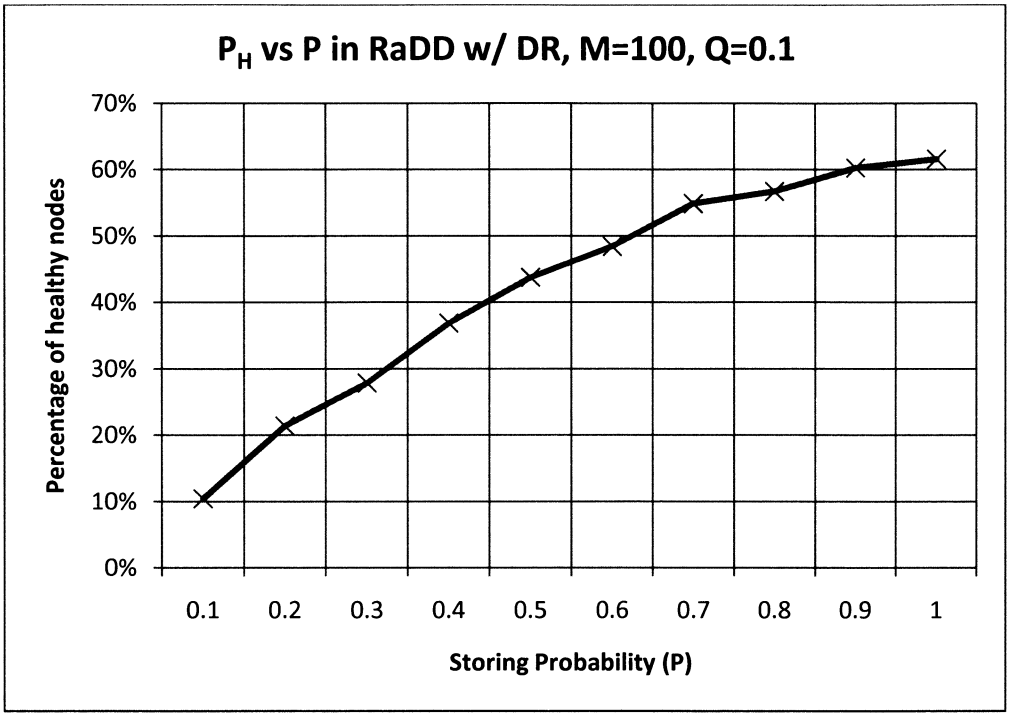

Figure 10: Healthy nodes vs. Storing Probability *P* in RaDD with DR

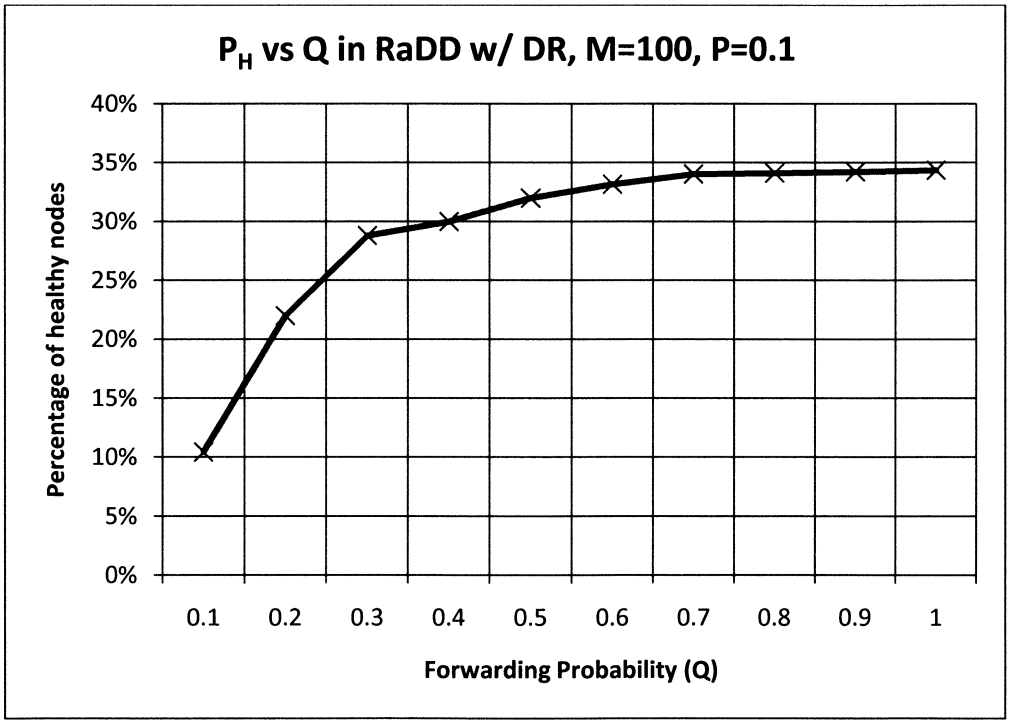

Figure 11: Healthy nodes vs. Forwarding Probability *Q* in RaDD with DR

Effect of parameter Q

The effect of parameter Q must also be taken into account, as the increasing of Q also increases $N_{target(phase1)}$. This is because it increases the number of broadcasted data and there are thus more chances for nodes to receive the target data in order to replace the corrupted version. However, when Q is close to 1 (i.e., when most of the nodes are broadcasting), it results in redundancy of broadcast messages. In other words, when Q is too large, too many nodes broadcast messages. For example, most nodes can receive data (that is, data can be spread network-wide) when only a certain portion of nodes broadcast. However, if most of nodes broadcast, redundancy ensues. Figure 11 shows that as the forwarding probability Q increases, the percentage of healthy nodes also increases when $Q<0.5$ and then nearly levels out. From the simulation results, our algorithms tend to work more efficiently with Q equal or less than 0.5.

Figure 12 illustrates the percentage of healthy versus the number of corrupted nodes with different values of P, Q and M. Compared to RaDD (Figure 5), the percentage of healthy data P_H in RaDD with DR increases significantly. When M is equal or less than 70 nodes, RaDD with DR can provide more than 50% of P_H with $P=0.1$, Q = 0.1 (the blue line). When M is larger than 70 nodes, the algorithm can also guarantee the majority of the healthy data by increasing the values of P and Q (P=0.2, Q=0.2 for the case M=80 or 90 (the green line), and P=0.4, Q=0.3 for the case M=100 (the purple line)). The superiority of this enhanced strategy is to introduce a network with reliability of healthy data despite the large number of compromised nodes. Specifically, we can get healthy data more than 50%, as the sink can determine the validity, in most of the cases and we

can thus outsmart the ADV. In particular, for the scenario with P=0.4 and Q=0.3, RaDD with DR is able to provide the collector more than 50% of healthy data even if the ADV has corrupted every single node (M=100). The 95% Confident Interval of the case P=0.4 and Q=0.3 is illustrated in Figure 13 with the maximum fluctuation of 6%. In other words, this strategy can defeat ADV in most of the cases with confidence.

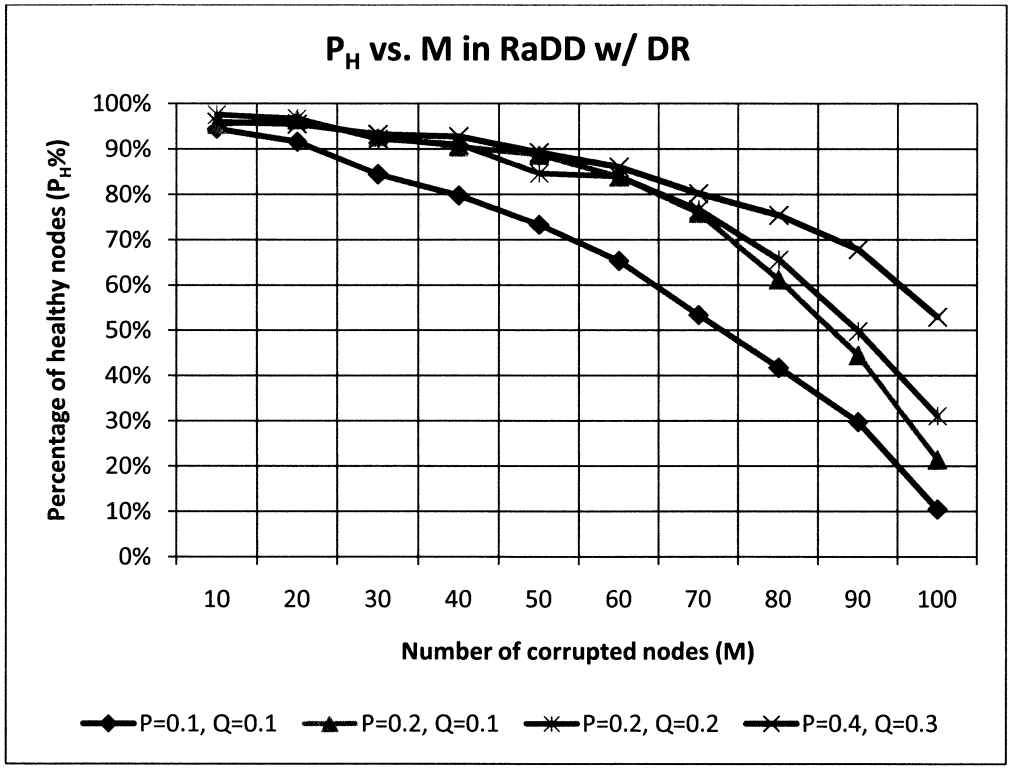

Figure 12: Healthy nodes vs. Corrupted nodes in RaDD with DR

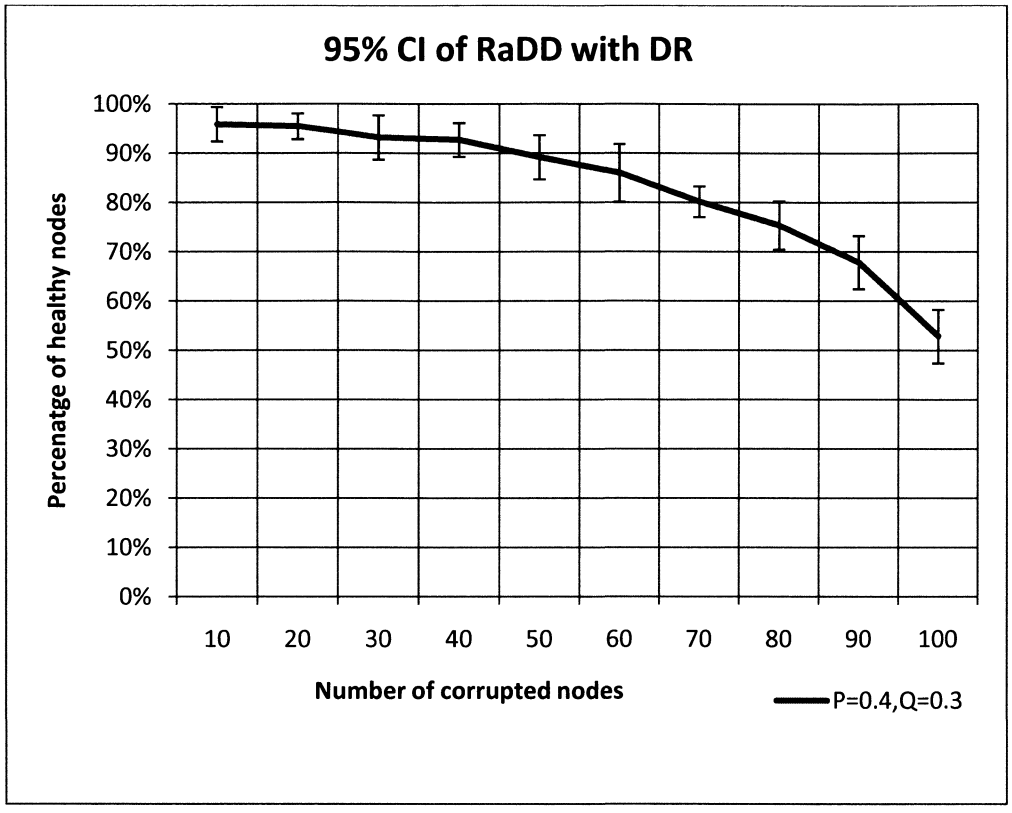

Figure 13: 95% Confident Interval of RaDD with Damage Recovery

b) Broadcast Messages

The total number of broadcast messages *B_M* of RaDD with DR is equal to those in RaDD algorithm plus the re-broadcasting messages of phase 2. As seen previously, *B_M* of RaDD, which is affected only by *Q,* was evaluated in Section 3.6, whereas *B_M* of DR is affected by $L_{(phase1)}$, and $L_{(phase1)}$ is affected by *P* and *Q* as mentioned in section 4.1. On the other hand, *B_M* is affected not only by *Q* but also by *P.* This explains why *B_M* is increased with *P* or *Q* increasing in Figures 14 and 15. However, when *Q* increases, *B_M* increases more than *P* does because *Q* affects *B_M* in both Phase 1 and 2 while *P* only affects *B_M* in Phase 2.

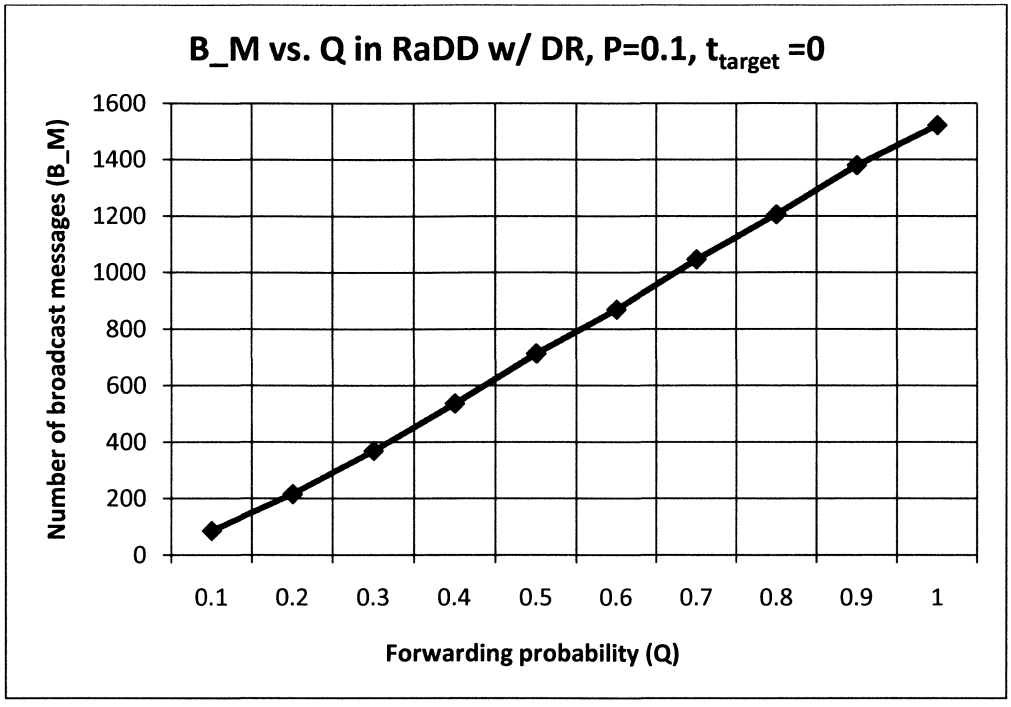

Figure 14: Broadcast messages versus *Q* in RaDD with Damage Recovery

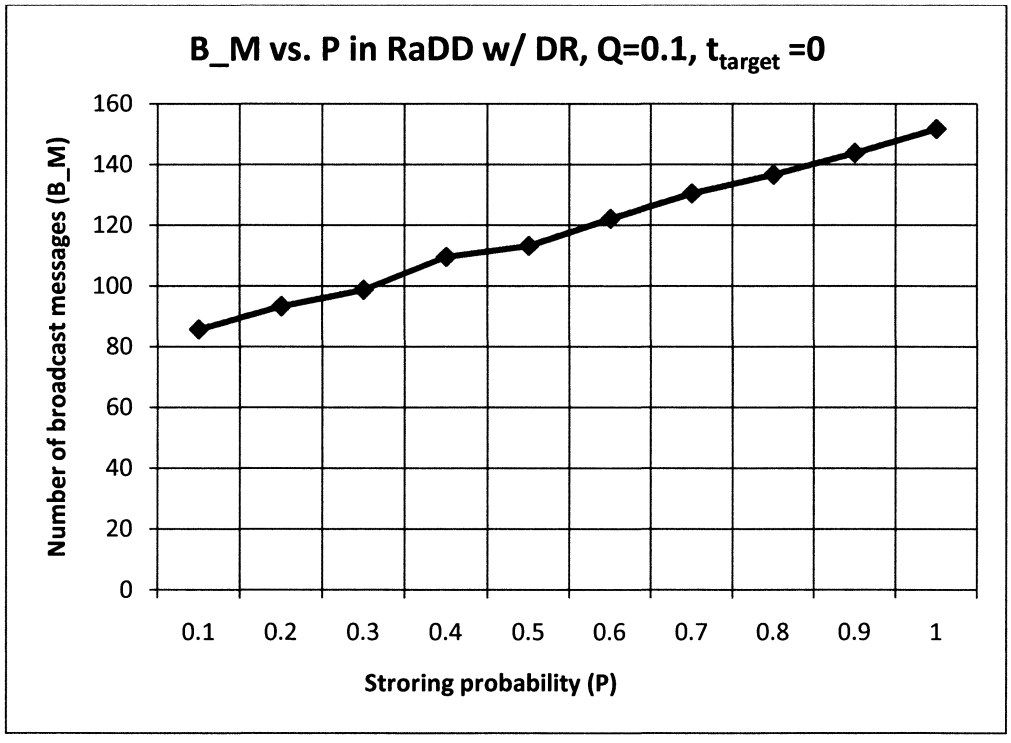

Figure 15: Broadcast messages versus *P* in RaDD with Damage Recovery

c) Number of nodes store target data

The number of nodes storing target data N_{target} of RaDD with DR is the same as in RaDD. It does not increase because only the nodes with damaged data replace the healthy version.

d) Improvement of RaDD with implementation of DR

Table XVI presents the percentage of healthy nodes and the total number of broadcast messages resulting from the implementation of Basic RaDD with and without DR. It shows that implementing DR enables healthy data to defeat the ADV with the cost of additional messages. Thus, it is a trade-off between the cost of broadcast messages and health of the data. Even if the ADV arrives early enough to corrupt most of the nodes (M>70), the implementation of DR can guarantee that the majority of the nodes (P_H) remain healthy at sink arrival time, whereas RaDD cannot do this. In a case where M=70, DR can defeat the ADV with a small values of P and Q equal only to 0.1 with the cost of about 33 broadcast messages (33≈100.4-66.47). In a case where M=80, P=0.2, Q=0.1, the percentage of healthy nodes is brought from 25.83% (without DR) to 61.26% (with DR), for an addition of around 45 broadcast messages (45≈117.32-72.14). And when ADV corrupts all the nodes in the network, DR can bring the percentage of healthy nodes higher than 50% (53.39%) for approximately cost of 100 broadcast messages (≈441-343) with *P=0.4, Q=0.3* compare to RaDD.

Table XVI: Improvement of RADD with Damage Recovery

Scenario	RaDD without DR		RaDD with DR	
	P_H(%)	B_M	P_H(%)	B_M
M=100, P=0.4, Q=0.3	14.87	343.27	52.90	441.30
M=90, P=0.3, Q=0.1	15.81	72.44	56.95	117.34
M=80, P=0.2, Q=0.1	25.83	72.14	61.26	117.32
M=70, P=0.1, Q=0.1	32.31	66.47	53.39	100.04

4.4.2 RaDD with DC

This section will present simulation results of RaDD with Damage Control, demonstrating the extent to which RaDD with DC shows a marked improvement over RaDD without DC.

a) Percentage of healthy data

Since P_H is proportional with M (the number of corrupted nodes) if there is no data replacement operation (basic RaDD). Thus the percentage of healthy data here is equal to those in RaDD plus the portion of nodes storing the target data which will replace corrupted version with a healthy one by the damage control process. P_H of RaDD is not affected by P or Q, while the percentage of data replacement is affected by Q. Because Q determines the number of broadcasts which affects the possibility for nodes storing the target data to receive the broadcasted healthy replacement in order to replace the stored corrupted one. Hence, P_H of RaDD with DC is determined by Q.

Figure 16 illustrates the percentage of healthy nodes versus forwarding probability Q. Along with the increase of Q, P_H increases dramatically (Q<0.5) and then roughly levels out. P_H increases because, with the larger values of Q, there are more nodes to broadcast.

Corrupted nodes have a greater chance to overwrite their data with the healthy version. However, when Q is greater than 0.5, P_H remains more or less constant as the network produces redundant broadcasting with large values of Q.

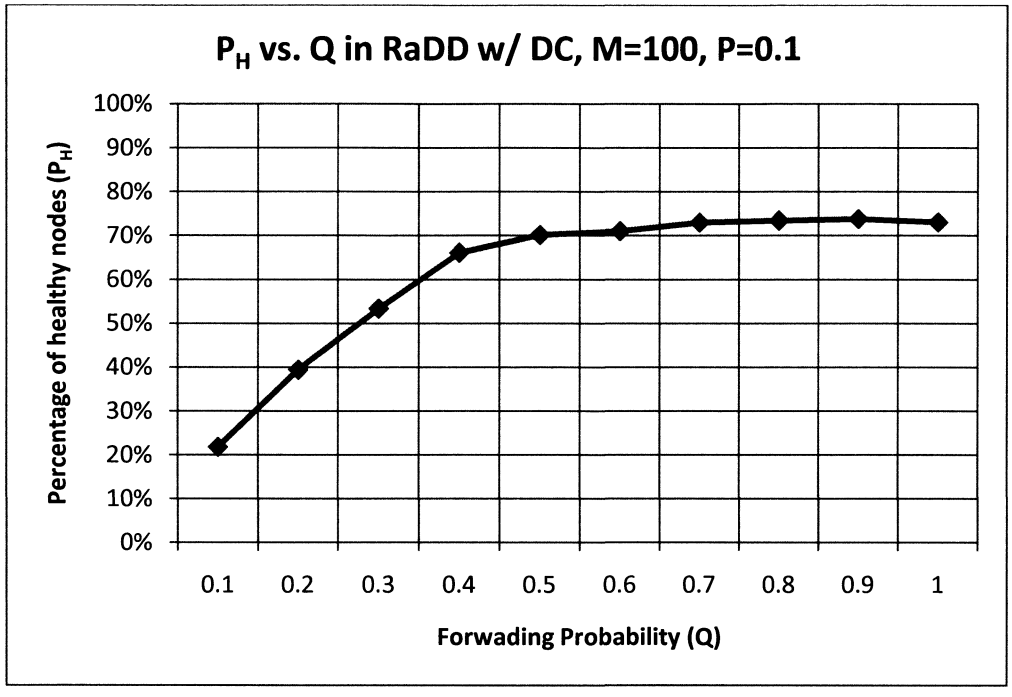

Figure 16: Percentage of healthy nodes vs. Q in RaDD with DC

Figure 17 exhibits the relationship between the percentage of healthy nodes versus corrupted ones with the implementation of RaDD with DC, from the simulation results. Since P does not affect P_H , we evaluate P_H with small value of P equal to 0.1 . When Q=0.1 (the blue line), the algorithm can provide the network with more than 50% of healthy nodes, with corrupted nodes M less than 70. When Q=*0.2* and Q=*0.3* (the red and green lines), the network can defeat ADV in most cases (P_H>50%). Moreover, even when ADV corrupts all nodes in the network (*M=100),* the algorithm can still defeat ADV with Q=0.3.

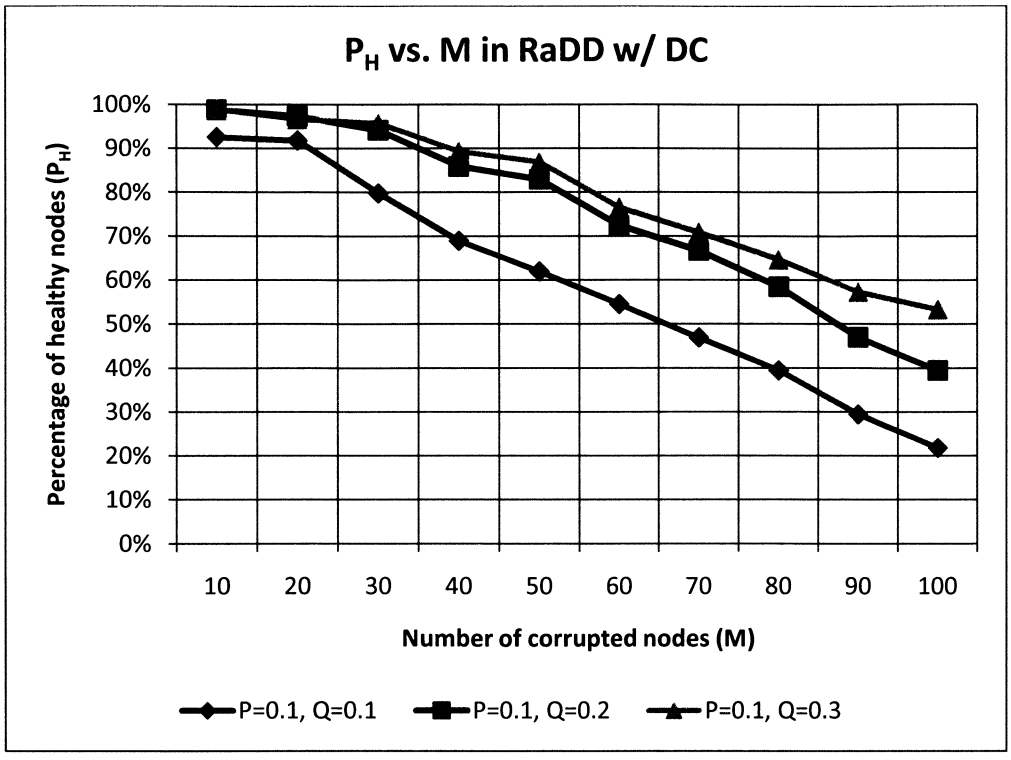

Figure 17: Percentage of healthy nodes vs. Corrupted nodes in RaDD w/ DC

b) Broadcast messages

Figure 18 demonstrates the number of broadcast messages in RaDD with the DC algorithm in the case the target data is collected early at t_{target}=0. Similar to basic RaDD, the number of broadcast messages increases linearly Q and is not affected by P. Because B_M is equal to those in RaDD minus the cancelled broadcasts by the damage control process. It is observable that B_M here is lower than the one in RaDD (Figure 18 vs. Figure 6).

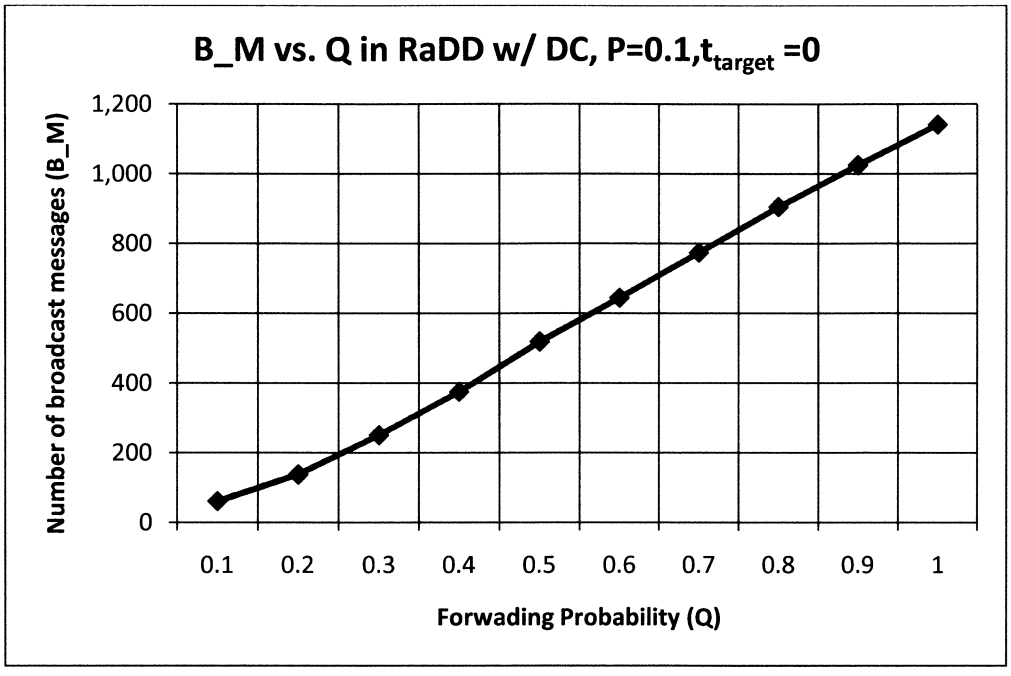

Figure 18: Broadcast messages in RaDD with DC

c) Number of nodes storing target data

The relationship between N_{target} and P or Q is similar to that for RaDD. Specifically, N_{target} increases linearly with P and also with Q when $Q < 0.6$ and roughly moderates when Q is greater than 0.6 (Figures 19 and 20). Technically, N_{target} here is less than N_{target} of RaDD by a portion of $N_{target(RaDD)}$ which cannot receive the data because of broadcast cancellation from Damage Control. This portion is determined by P and Q which affects the possibility to store and to forward data. However, when P and/or Q are small, along with the small value of $N_{target(RaDD)}$, then the improvement is slightly noticeable. We will see examples of this (small) difference in Table XVII.

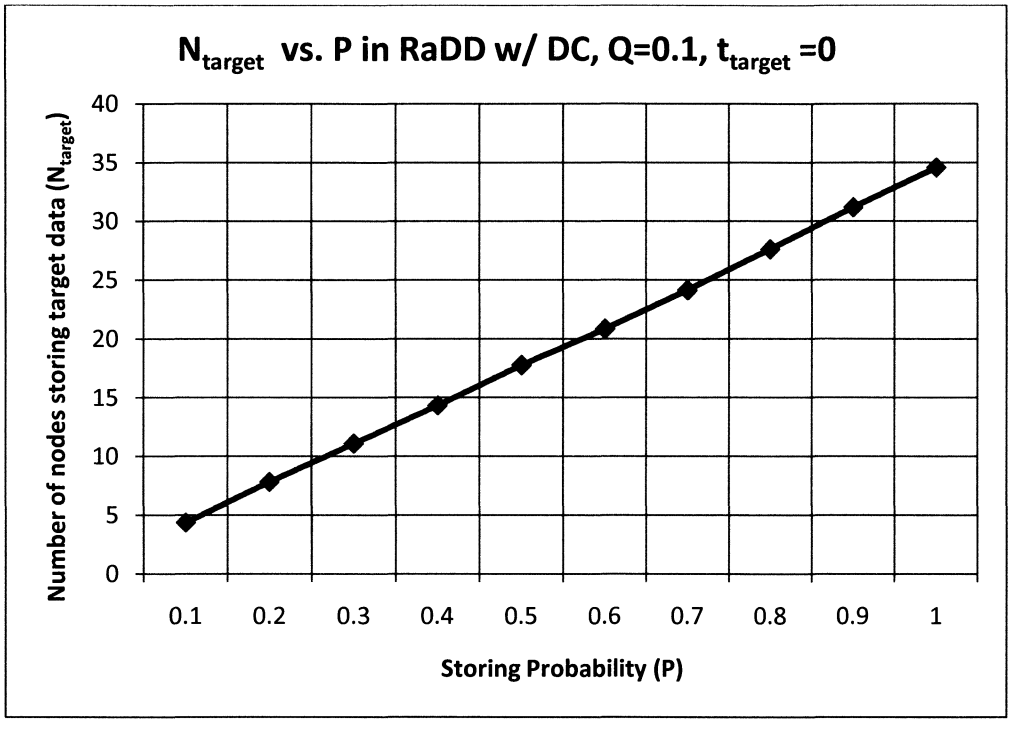

Figure 19: Number of nodes storing target data vs. *P* in RaDD with DC

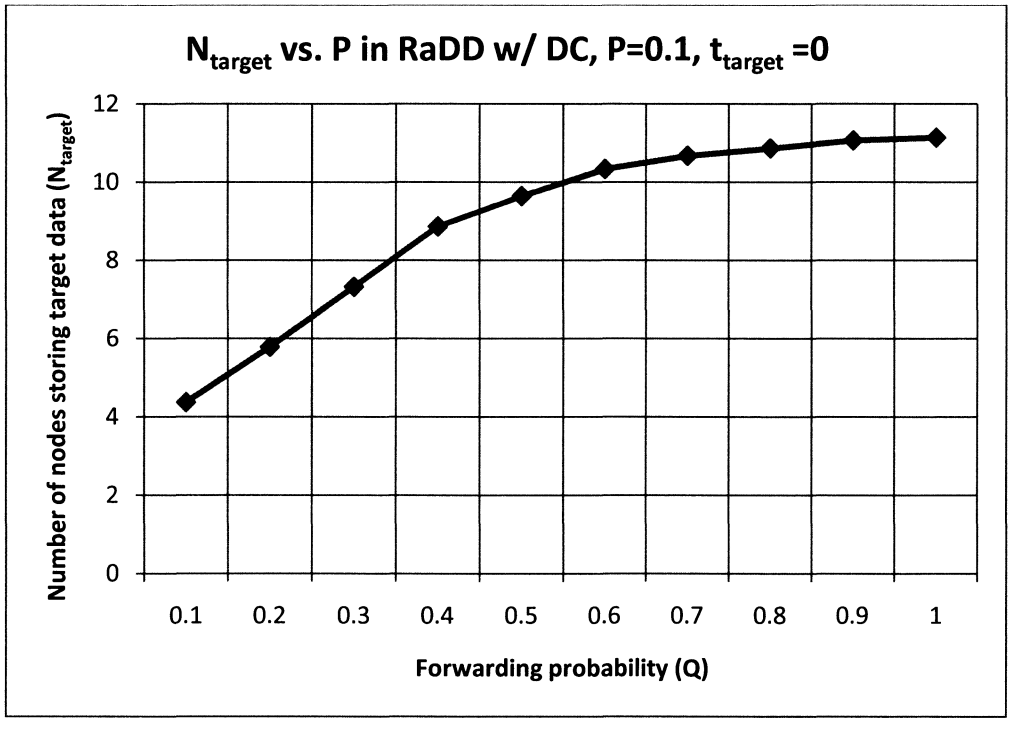

Figure 20: Number of nodes storing target data vs. *Q* in RaDD with DC

d) Improvement of RaDD with implementation of DC

RaDD with Damage Control shows a decided improvement over basic RaDD. Table XVII shows that DC-augmented RaDD can defeat ADV (P_H>50) in scenarios involving high numbers of corrupted nodes (M>50) when basic RaDD cannot. Furthermore, RaDD with DC reduces network costs by decreasing both the number of broadcast messages and the number of nodes storing data. This is advantageous, as DC brings improvements without inciting additional cost. For example, in scenario M=80, P=0.1, Q=0.2, DR can bring P_H of RaDD from 26.42% to 58.45% with about 53 fewer broadcast messages (~177.23-124.07) and a decrease from 6.11 to 5.59 in the average number of nodes storing messages.

Figure 21 shows a 95% Confident Interval in cases P=0.1 and Q=0.3, with a half-width interval fluctuating between 1.1% and 4.95%. This shows that networks can produce a majority of healthy nodes in most cases, even in instances involving large numbers of corrupted nodes, such as 90 or 100.

Table XVII: Improvement of RaDD with implementation of DC

Scenario	RaDD without DC			RaDD with DC		
	P_H (%)	B_M	N_{target}	P_H (%)	B_M	N_{target}
M=100, P=0.1, Q=0.3	12.38	343.27	8.21	53.41	251.03	7.45
M=90, P=0.1, Q=0.3	18.67	296.67	7.58	57.31	224.36	7.09
M=80, P=0.1, Q=0.2	26.42	177.23	6.11	58.45	124.07	5.59
M=70, P=0.1, Q=0.2	32.47	149.51	5.55	66.73	117.16	5.36
M=60, P=0.1, Q=0.1	41.12	69.62	3.97	54.54	56.17	3.87

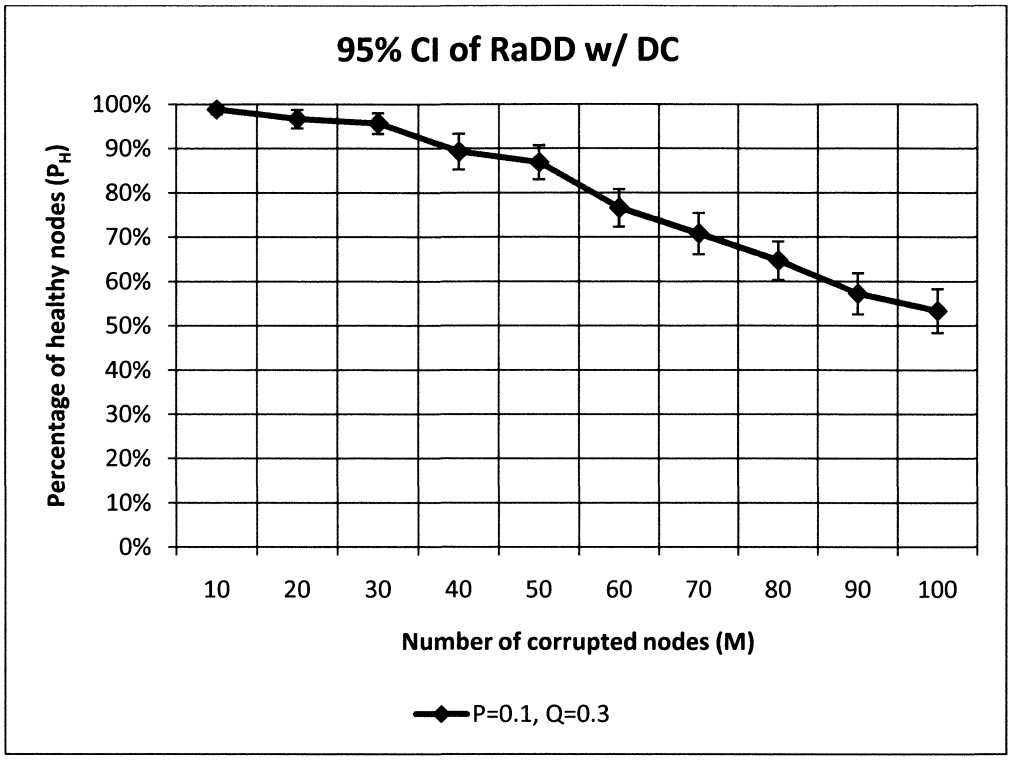

Figure 21: Confident Interval of RaDD with DC

4.4.3 Damage Recovery vs. Damage Control

This section compares the performance of Damage Recovery with Damage Control based on the simulation results.

Figures 22, 23 and 24 show the simulation results of RaDD with the implementation of Damage Recovery and RaDD with the implementation of Damage Control. When P=0.1 and Q=0.1, DR seems to perform slightly better than DC (Figure 22). Further, when P=0.2 and Q=0.1 (Figure 23), DR performs much better than DC. However, when P=0.1 and Q=0.3, DC is superior to DR (Figure 24).

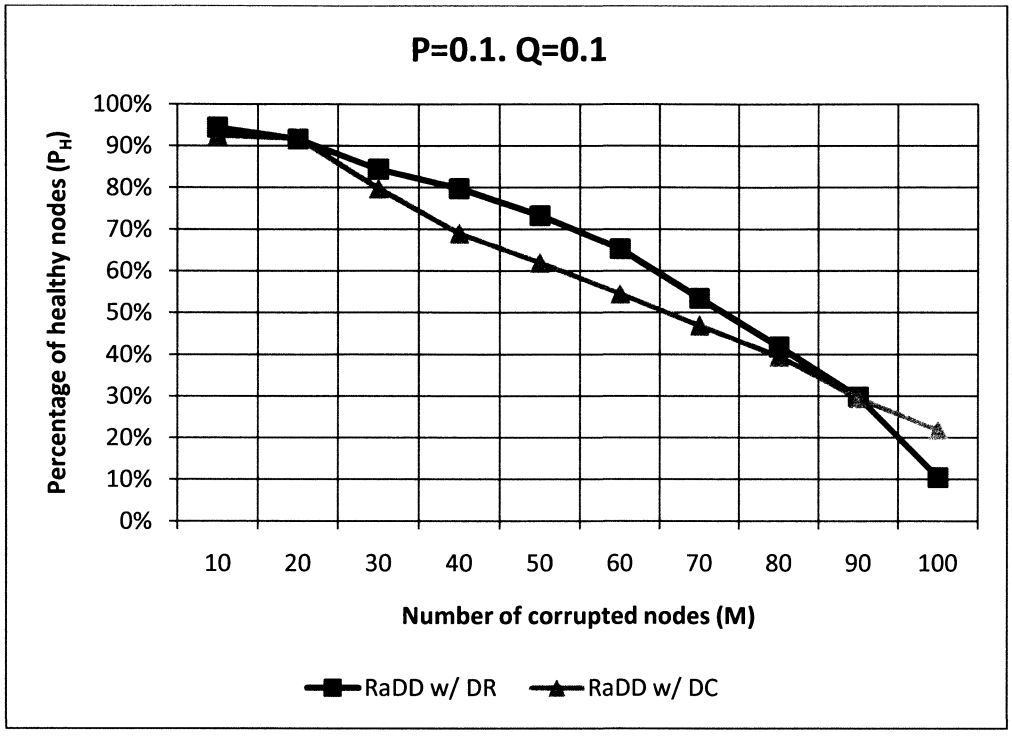

Figure 22: DR vs. DC, P=0.1, Q=0.1

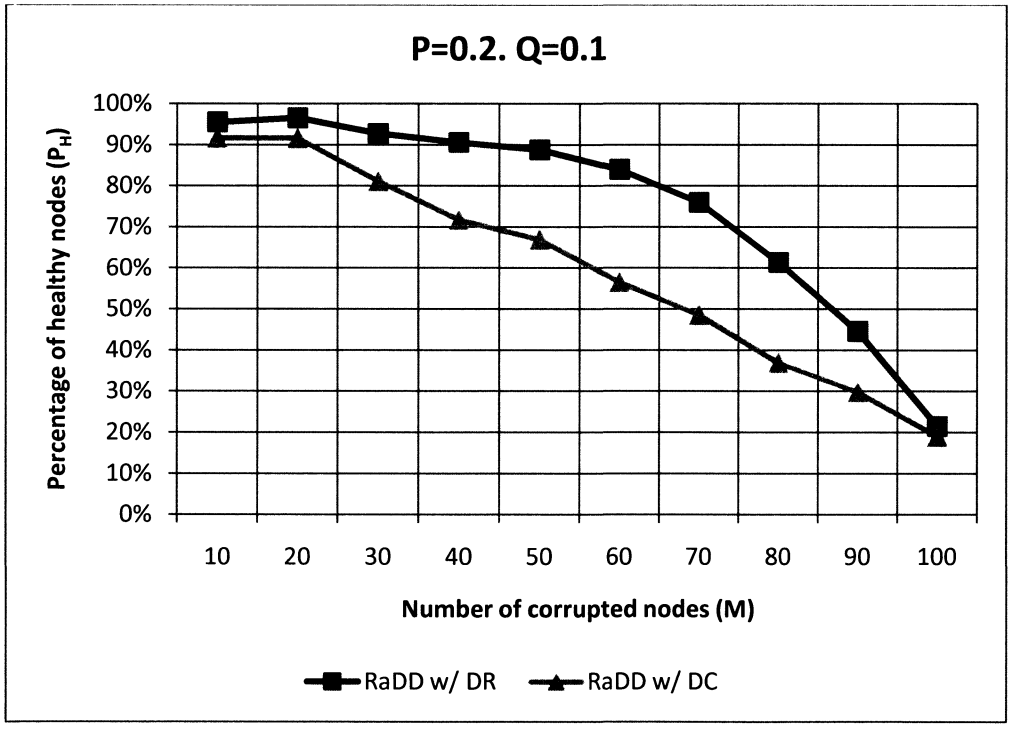

Figure 23: DR vs. DC, P=0.2, Q=0.1

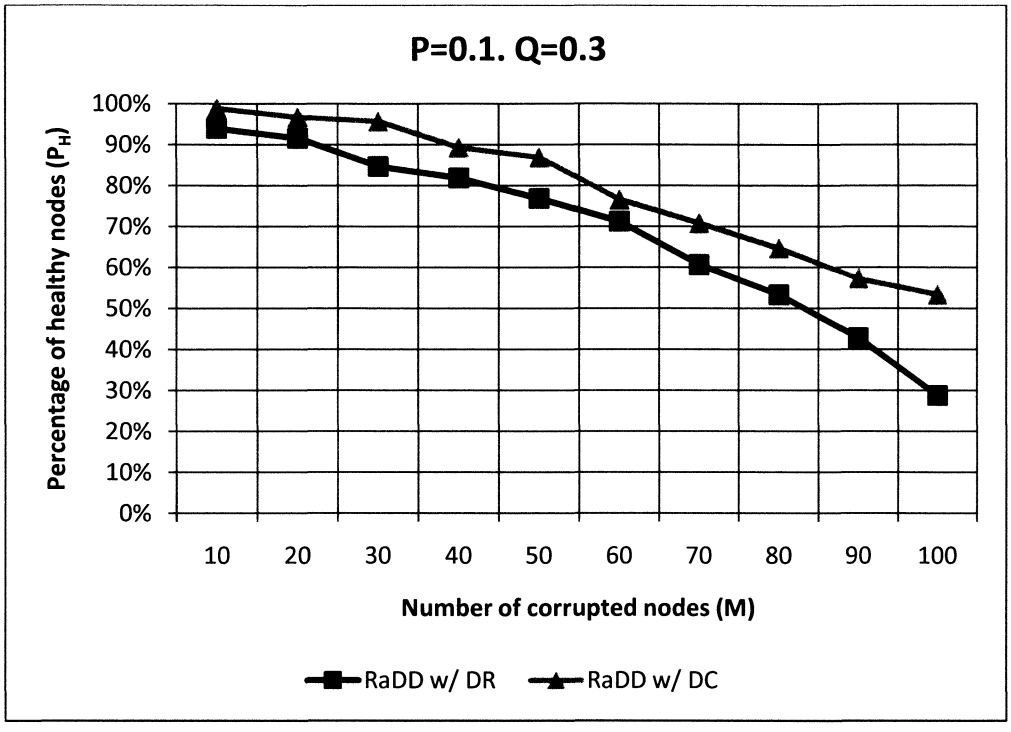

Figure 24: DR vs. DC, P=0.1, Q=0.3

Table XVIII presents the percentage of healthy nodes and the total number of broadcast messages resulting from the implementation of Basic RaDD with DR and DC. With the same number of corrupted nodes in the network, DR can beat ADV with the higher value of P whereas DC can do so with the higher value of Q. For instance, in the scenario of M=80, DR provides the network with 61.26% of healthy data with P=0.2 and Q=0.1, while DC provides it with 58.45% with P=0.1 and Q=0.2.This table also shows that, in all scenarios, at least one of the two schemes guarantees P_H that is at least 50%.

Table XVIII: Comparison between Damage Recovery and Damage Control

Scenario	RaDD with DR		RaDD with DC	
	P_H (%)	B_M	P_H (%)	B_M
M=100, P=0.1, Q=0.3	28.78	369.06	53.41	251.03
M=90, P=0.1, Q=0.3	42.79	334.12	57.31	224.36
M=90, P=0.3, Q=0.1	56.95	117.34	23.88	69.84
M=80, P=0.1, Q=0.2	48.78	213.26	58.45	124.07
M=80, P=0.2, Q=0.1	61.26	117.32	36.89	56.35
M=70, P=0.1, Q=0.1	53.39	100.04	46.91	59.59

In conclusion, DR works better with higher P, while DC works better with higher Q, The choice of the two parameters P (probability to keep a local copy of a data) and Q (probability to forward it to the neighbours) for either DR or DC can noticeably improve the final value of P_H.

4.4.4 Combined Model

a) Percentage of healthy data

Figure 25 shows P_H vs. M in the combined model. It is observable that it provides the network with a majority of healthy nodes in most cases. P_H increases with the increasing of P and/or Q. The combined model provides higher percentage of healthy nodes than single DR or DC, especially in cases of large corrupted nodes (Figure 25 vs. Figures 12 and 17).

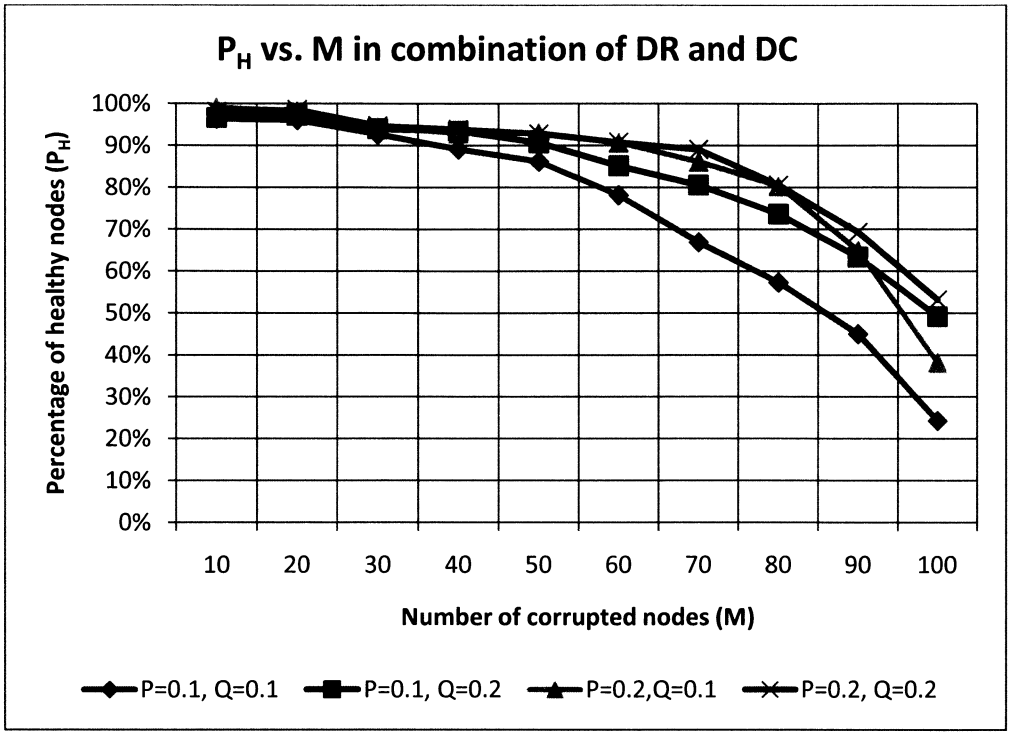

Figure 25: Percentage of healthy nodes in Combined model

b) Broadcast messages

The number of broadcast messages vs. *P* and *Q* is shown in Figures 26 and 27 for a situation where the target data is collected early at time t=0 (right after the sink leaves, at the outset of the network operation period). In this case, there is the highest number of broadcast messages because the later the data is collected, the lower the number of broadcast messages, as the time allotted to broadcast message is decreased.

B_M increases significantly with *Q* (Figure 26), as *B_M* here is the sum of *B_M* of RaDD with DC and *B_M* of DR phase. The *B_M* of each is dramatically increased with *Q*, as we explored above. For these reasons, the sum of indicators shows a dramatic increase as well.

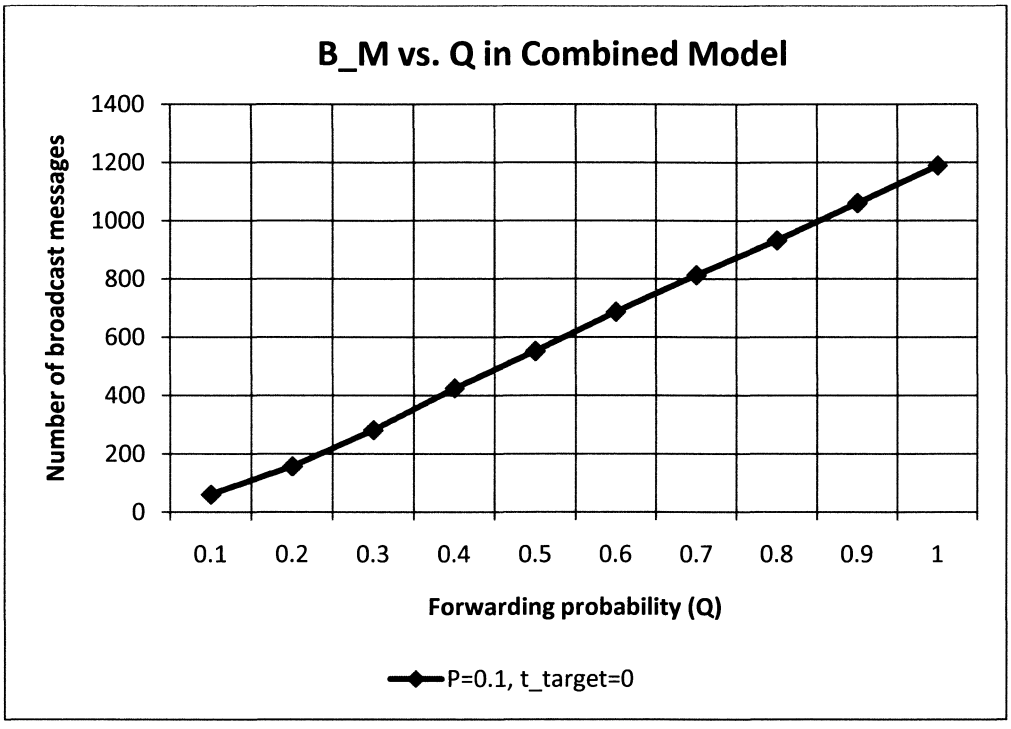

Figure 26: Broadcast messages vs. *P* in Combined Model

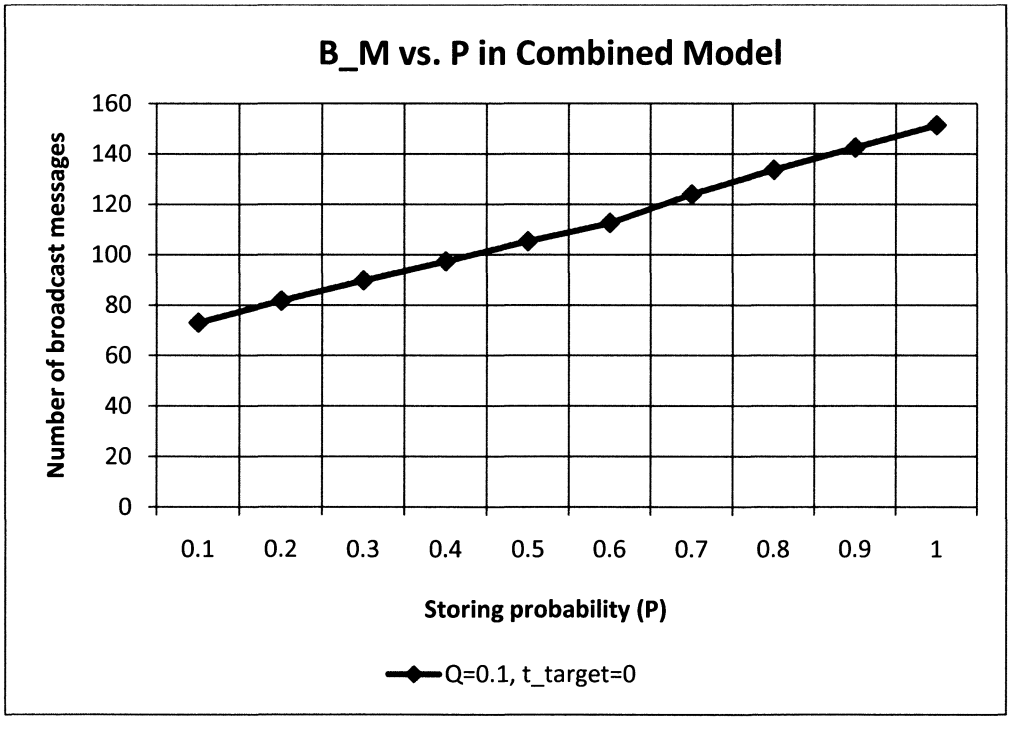

Figure 27: Broadcast messages vs. *P* in Combined Model

When P increases, B_M does so only slightly (Figure 27). This is because P values only affect B_M in the Damage Recovery operation. It does not affect B_M of RaDD with DC, as we have seen already in analyses in previous sections of this chapter.

c) Number of nodes storing the target data

The number of nodes storing data in the combined model is equal to those in RaDD in DC because it is the implementation of Damage Recovery on top of RaDD with Damage Control.

4.4.5 Comparisons between RaDD with and without DR and/or DC

In this section, we will compare the performance of the four algorithms discussed above: 1) RaDD, 2) RaDD with DR, 3) RaDD with DC, and 4) RaDD combined with both DR and DC. In addition, we will provide recommendation for the best algorithms with the values of P and Q according as the number of corrupted nodes M in the network.

Figures 28, 29 and 30 demonstrate the simulation results from the implementation of RaDD with and without DR and/or DC for different values of P and Q. It shows that implementing of DR and/or DC with RaDD significantly increases the percentage of healthy nodes. Thus the choice of algorithm along with appropriate values of P and Q can defeat ADV in most cases. It is also noticeable that DR and DC together as a combined model provides higher percentage of healthy data than the other three models.

However, a good algorithm is determined by not only the high healthy data but also the low cost of broadcast messages and storage. Table XIX summarizes the performance the four algorithms with the various values of the number of corrupted nodes M from the

simulation results and the choice of algorithms is suggested. When M is less than 50, the basic RaDD is the simplest algorithm that can successfully defeat ADV with P=0.1 and Q=0.1, while RaDD with DC is also effective, with a similar cost of B_M and N_{target}. As M increases, RaDD no longer provides sufficient reliability for the network and therefore enhanced strategies with more sensor processing are required to provide a majority of healthy nodes. When M=50 or 60, any enhanced algorithm can provide the network with more than 50% of healthy nodes, with P=0.1 and Q=0.1. For this case, DC is the best algorithm with the lowest B_M and N_{target}. When M=70, RaDD with DR and combined DR and DC can defeat ADV with P=0.1, Q=0.1, while RaDD or DC cannot. When M=80, 90 or 100, the combined DR and DC algorithm emerges as the dominant method to defeat ADV.

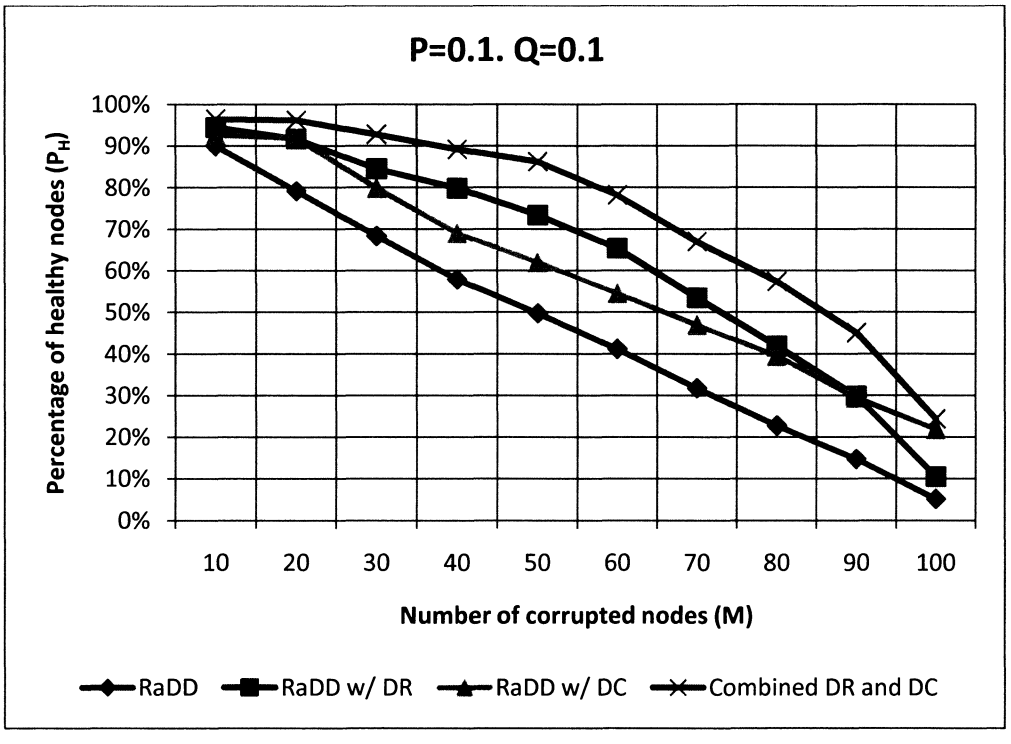

Figure 28: A comparison of four models in case P=0.1, Q=0.1

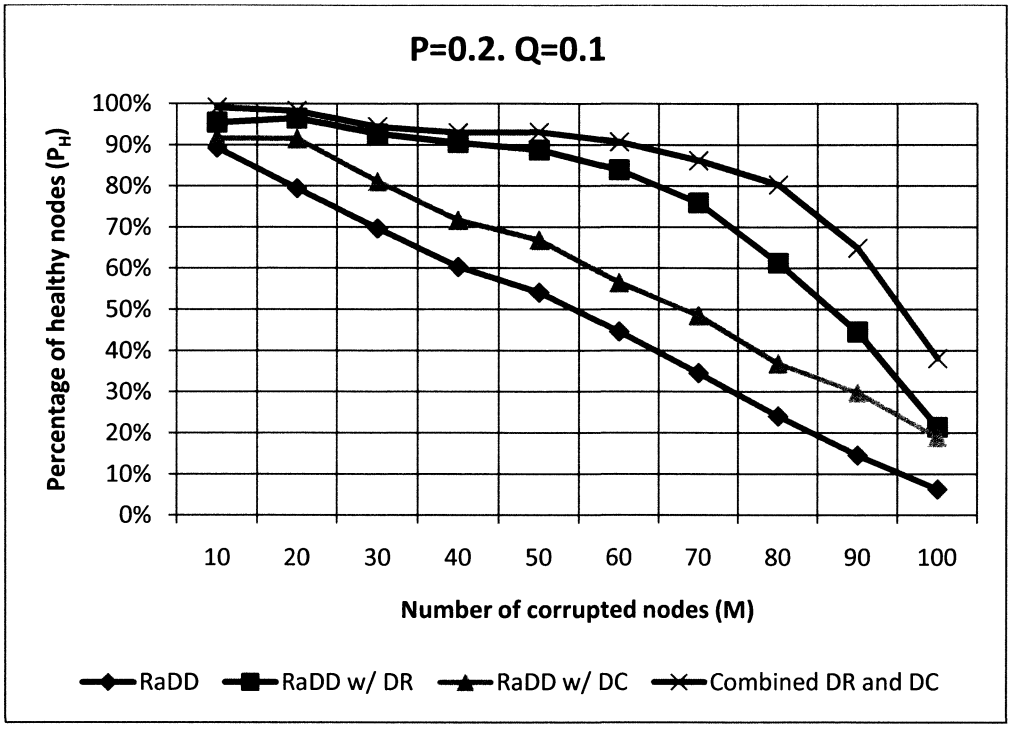

Figure 29: A comparison of four models in case P=0.2, Q=0.1

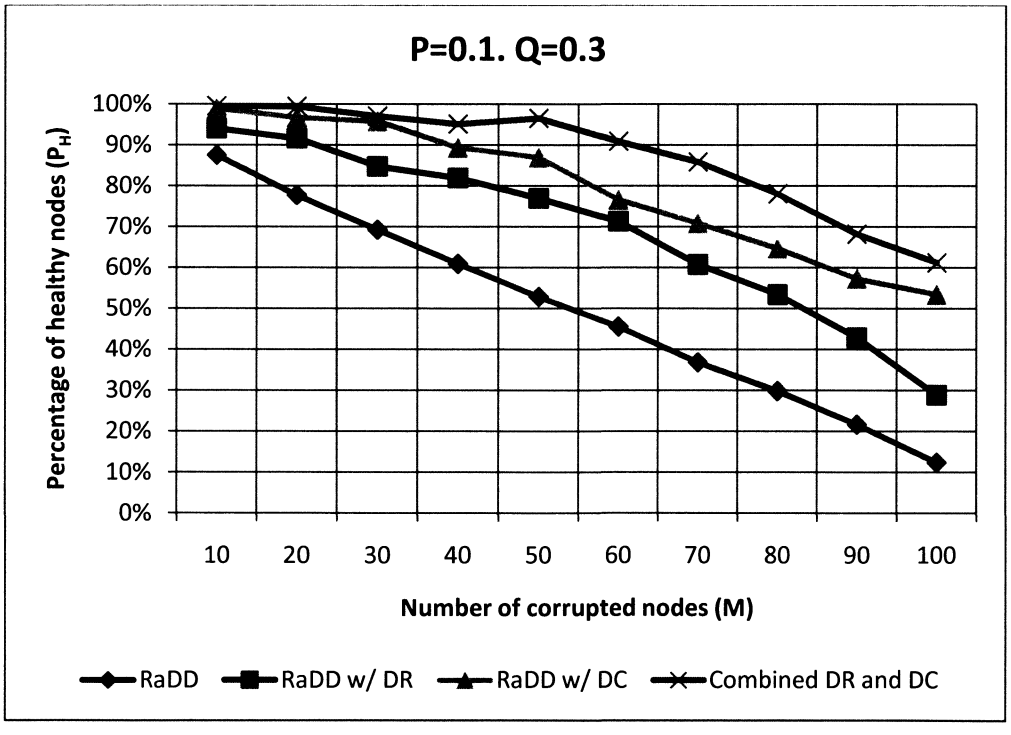

Figure 30: A comparison of four models in case P=0.1, Q=0.3

Table XIX: Summary of algorithms

Scenario	*Algorithm*	P_H *(%)*	*B_M*	N_{target}	*Best algorithms*
M<50, P=0.1, Q=0.1	RaDD	(57, 90)	(19, 55)	(2.7, 3.5)	RaDD, RaDD w/ DC
	RaDD w/ DR	(79, 95)	(85, 104)	(2.7, 3.5)	
	RaDD w/ DC	(68, 92)	(19, 48)	(2.7, 3.5)	
	RaDD w/ DR and DC	(90, 96)	(85, 103)	(2.7, 3.5)	
M=50, P=0.1, Q=0.1	RaDD	49.70	68.11	3.93	RaDD w/ DC
	RaDD w/ DR	73.28	113.96	3.93	
	RaDD w/ DC	61.98	55.71	3.87	
	RaDD w/ DR and DC	86.2	100.59	3.83	
M=60, P=0.1, Q=0.1	RaDD	41.12	69.62	3.97	RaDD w/ DC, RaDD w/ DR and DC
	RaDD w/ DR	65.3	109.83	3.97	
	RaDD w/ DC	54.54	56.17	3.87	
	RaDD w/ DR and DC	78.15	93.95	3.85	
M=70, P=0.1, Q=0.1	RaDD	32.31	66.47	4.01	RaDD w/ DR, RaDD w/ DR and DC
	RaDD w/ DR	53.39	100.04	4.01	
	RaDD w/ DC	46.91	59.59	4.05	
	RaDD w/ DR and DC	66.94	87.83	4.05	
M=80, P=0.1, Q=0.1	RaDD	22.69	76.94	4.17	RaDD w/ DR and DC
	RaDD w/ DR	41.78	101.06	4.17	
	RaDD w/ DC	39.47	60.46	4.05	
	RaDD w/ DR and DC	57.32	80.24	4.05	
M=90, P=0.1, Q=0.2	RaDD	18.90	179.76	6.12	RaDD w/ DR and DC
	RaDD w/ DR	37.46	207.85	6.12	
	RaDD w/ DC	46.94	128.01	5.61	
	RaDD w/ DR and DC	93.34	165.49	5.61	
M=90, P=0.2, Q=0.1	RaDD	14.56	75	7.41	RaDD w/ DR and DC
	RaDD w/ DR	44.52	118.28	7.41	
	RaDD w/ DC	29.78	56.69	6.72	
	RaDD w/ DR and DC	64.92	115.35	6.72	
M=100, P=0.1, Q=0.3	RaDD	12.38	343.27	8.21	RaDD w/ DC, RaDD w/ DR and DC
	RaDD w/ DR	28.78	369.06	8.21	
	RaDD w/ DC	53.41	251.03	7.45	
	RaDD w/ DR and DC	61.15	273.91	7.45	
M=100, P=0.2, Q=0.2	RaDD	10.89	199.5	11.91	RaDD w/ DR and DC
	RaDD w/ DR	31.16	232.45	11.91	
	RaDD w/ DC	38.96	129.6	10.26	
	RaDD w/ DR and DC	53.16	189.19	10.26	

Chapter 5: Extended Models

This chapter describes extended models in which we will explore how our algorithms can work with different situations. First we will examine how RaDD can function over long period of activity. Then we will consider a RaDD algorithm against an Eraser ADV. Finally, a dynamic RaDD will be probed in order to reduce the network cost in terms of broadcasts and storage.

5.1 RaDD over long period of activity

In reality, there are situations where sensors' activities are left unattended for a longer time than 5.1 minutes, such as earthquake monitoring or historical data collection. The following section will examine RaDD with and without Damage Recovery (DR), or Damage Control (DC) in such situations. The purpose here, as in previous Chapters, is to evaluate the algorithms in terms of data survival probability, based on simulation results. In other words, we intend to investigate and measure how our proposed algorithms function over longer time periods.

5.1.1 System Model

The system model here is similar to the one described in Section 3.1 and 3.2, however there are some noteworthy changes in network activity that result in consequent changes in ADV strategy as follows.

For the network, the time between sink visits is t_{sink}=30min. (compared to 5.1 min. in Chapter 3 and 4). Since the network operates over an extended period of time between sink visits, it is not practical for an ADV to remain continuously in the network. This is mainly because it does not have sufficient energy to be constantly active. Furthermore, the stored data in sensors' storage might be out of date at a certain time, and sensors would replace them with the recent collected one. Thus we assume ADV visits the network periodically, every five minutes. Each time visit, it stays there for T' units of time and corrupts up to M' nodes such that it would corrupt at most N nodes (network size) between sink visits.

As with previous Chapters, in this section, we examine one data which also called the target data. The target data is collected at time t_{target}=5. Meanwhile, an ADV starts to corrupt the network for the first time at $t_{ADV(1)}$ which is equal to t_{target} plus Δt units of time ($t_{ADV(1)} = t_{target}+\Delta t$, with $\Delta t = 0.1$). It then repeats the operations every five minutes. Thus, with our setting, ADV visits the network five times at $t_{ADV(1)} = 5.1$, $t_{ADV(2)} = 10.1$, $t_{ADV(3)} = 15.1$, $t_{ADV(4)} = 20.1$, and $t_{ADV(5)} = 25.1$. Upon each visit, it is able to corrupt up to M' = 20 nodes continuously such that it can corrupt maximum 100 nodes between sink visits. Recall that δ is the time for an ADV to corrupt a node. It

should be noted here that each time an ADV visits the network, it stays there for a duration of $T'=M'\times \delta$.

Changes to the values of input parameters used in simulation are given in Table XX. Note that in this model (t_{sink} = 30 min.), we set t_1 and t_2 to 6 and 9 which are roughly mapping values of t_1=1, and t_2=1.5 in the case t_{sink}= 5.1min.

Table XX: Changing the values of input parameters

Parameter	***Values in Basic RaDD***	***Values in RADD over longer period***
N (nodes)	100	100
t_{sink}*(minutes)*	5.1	30
t_1, t_2 *(minutes)*	1, 1.5	6, 9
t_{target} *(minutes)*	Dynamic (0, 0.5, 1.0,..., 5.0)	5
Δt *(minutes)*	0.1	0.1
δ *(minutes)*	0.05	0.05
t_{ADV} *(minutes)*	$t_{target} + \Delta t$	5.1, 10.1, 15.1, 20.1, 25.1
τ *(minutes)*	0.2	1.2
t_{DR} *(minutes)*	4.9	28.8

Table XXI provides pseudo-code for ADV activity. The other pseudo-codes of RaDD algorithms here are the same as in Chapters 3 and 4.

Table XXI: Mobile Adversary Activity in Extended Model

MOBILE ADVERSARY ACTIVITY
$S = \{0, 1, 2, \ldots, N\}$ /*set of all sensors (ID) in the networks*/
$t_{ADV} = t_{target} + \Delta t$
Iter =1 /* number of times ADV has visited the network*/
$t_{ADV(Iter)} = t_{ADV}$
/*ADV starts corrupting the network Δt units of time after the target data is collected*/
$C_{ADV} = \{\}$ /* Initialize the set of sensor ADV has corrupted*/
while $(t < t_{sink})$ **do** { /* while the sink has not arrived*/
Counter=0 /* a count of corrupted nodes at each visit*/
while (Counter < M'**) do** { /* ADV corrupts M' nodes at each visit*/
Pick one random node in $S \backslash \{C_{ADV}\}$**, assume node** i **is selected**
/* ADV picks node *i* to corrupt such that ADV has not corrupted it before */
$t_{ADV_i} = t_{ADV(Iter)}$ /* ADV corrupts node *i* at time t_{ADV_i} */
$t_{ADV(Iter)} \mathrel{+}= \delta$ /*ADV finishes corrupting node *i* */
$C_{ADV} + \{i\}$ /* add node *i* into set of corrupted nodes*/
Counter ++
} end while
Iter++
$t_{ADV(Iter)} = t_{ADV} + 5 * (\mathbf{Iter} - 1)$ /* set time for next visit of ADV*/
} end while

5.1.2 Simulation Results

Basic RaDD

The simulation results show that basic RaDD over a long period is similar to one with a short period of activity (Figure 31 vs. Figure 5). It is effective as an algorithm only when M is less than 50. In such cases, P and Q can be set to be equal to 0.1 in order to defeat an ADV. Noted that t_{target} here is constant, t_{target} = 5, B_M and N_{target} is similar to the case where t_{target} is around 0.85 in Section 3.6.

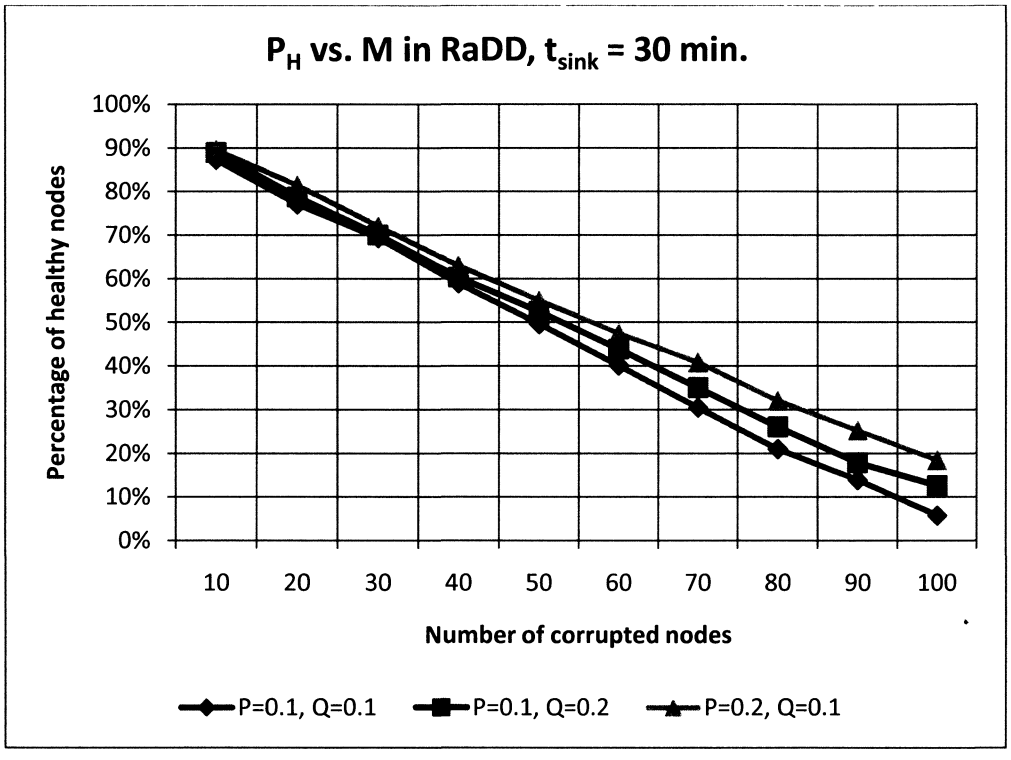

Figure 31: Healthy nodes vs. Corrupted nodes in RaDD, t_{sink}= 30 min.

RaDD with Damage Recovery

The Damage Recovery (DR) phase starts 1.2 minutes prior to the arrival of the sink. Here, the simulation result is the same as for that of the shorter operation (Figure 32 vs.

Figure 12). In most cases, DR can defeat an ADV, with P=0.4 and Q=0.3 (the green line).

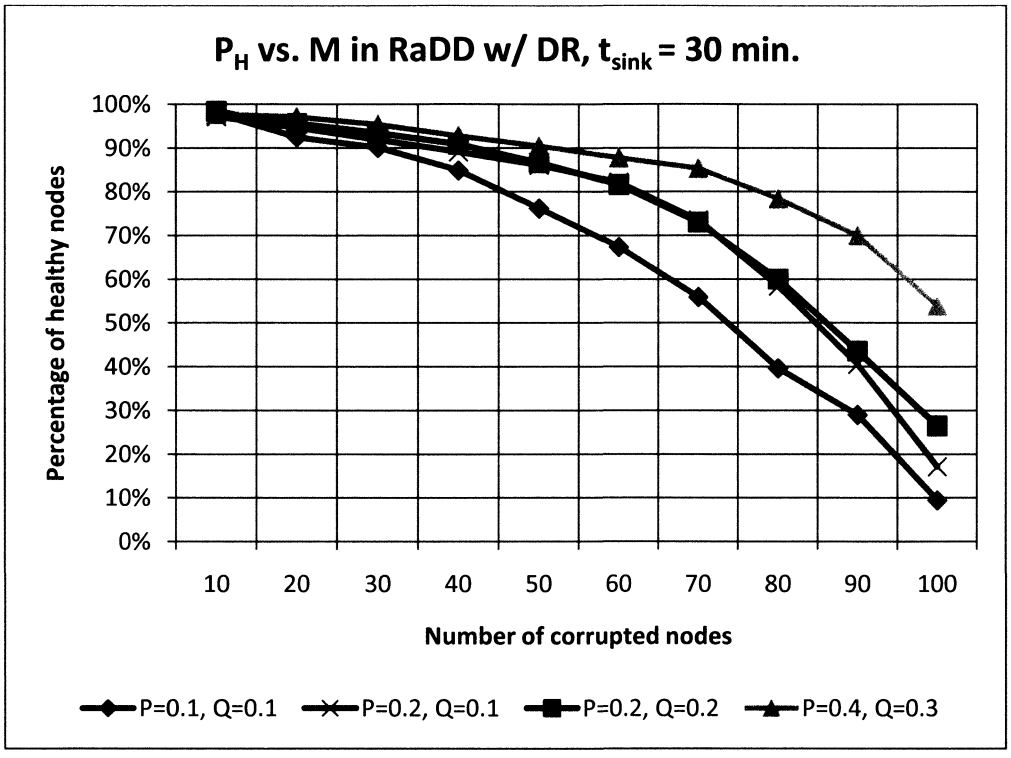

Figure 32: Healthy nodes vs. Corrupted nodes in RaDD w/ DR, t_{sink} = 30 min.

RaDD with Damage Control

Simulation results indicate that, when M<50, DC gives results that are similar to cases involving shorter time spans (Figure 33 vs. Figure 17). However, when $M \geq 50$, DC offers better results. In particular, with *P=0.1* and *Q=0.2*, a network can produce healthy nodes more than 50% even ADV 100 nodes (the red line in Figure 33). This can be compared to Section 4.4, where Q requires higher values (Q=0.3) to provide similar results (the green line in Figure 17). Figure 34 shows 95% confident interval with half width fluctuated approximately from 1% to7% in the case P=0.1 and Q=0.2.

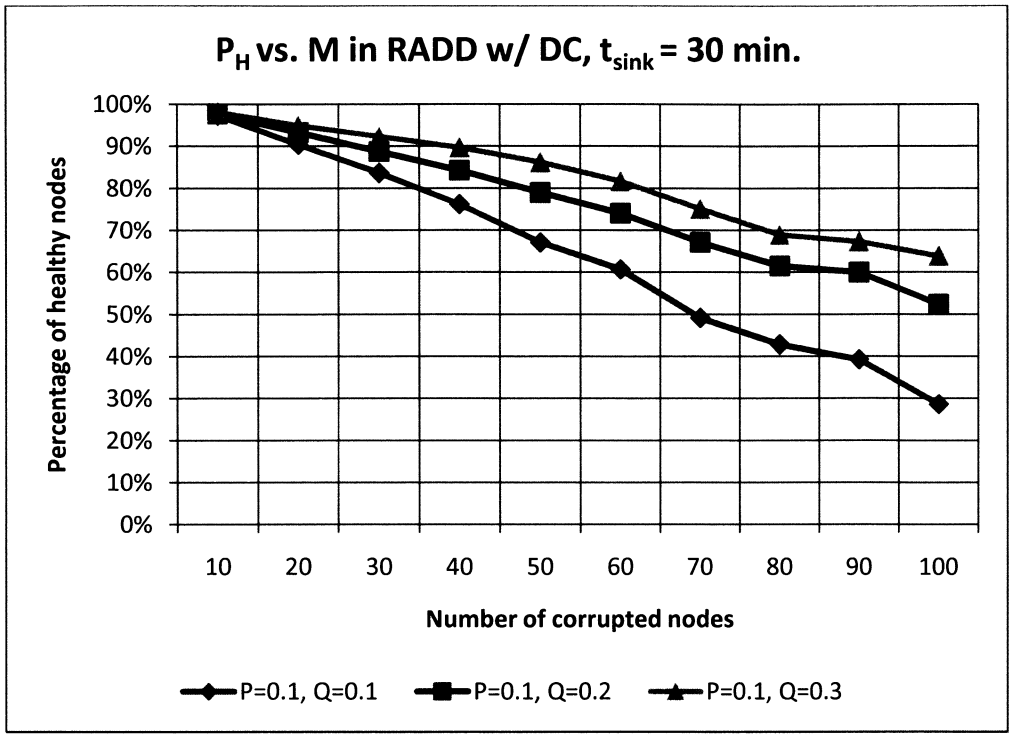

Figure 33: Healthy nodes vs. Corrupted nodes in RaDD w/ DC, t_{sink}= 30min.

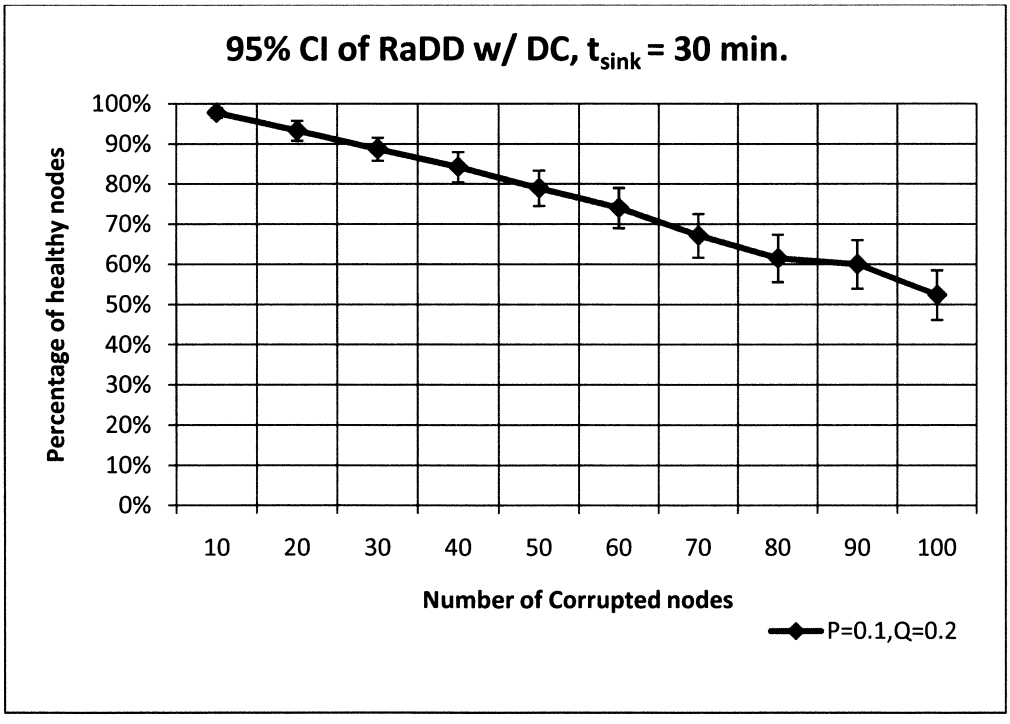

Figure 34: 95% Confident Interval in RaDD w/ DC, t_{sink}= 30min.

In summary, it can be observed from the preceding that our proposed algorithms work well in case network scenarios involving longer operating times. We believe our algorithms with very large t_{sink}, such as one day or longer, and where the maximum corrupted nodes is *N*, would be particularly effective. Based on these encouraging findings, our future endeavours will focus on a continuation of these investigations.

5.2 RaDD against Eraser ADV

Eraser ADV is another type of ADV who does not target any particular data. It aims to delete as much data as possible between sink visits. In the other words, its main goal is denial-of-service [6]. Those deleted data might contain critical information. Thus, data redundancy a natural choice to obtain data survivability to the collector.

In this section, we will examine how our basis RaDD algorithm can produce data replication (or redundancy) in order to defeat such an Eraser ADV.

5.2.1 System model

The network model is the same as the Basic model (section 3.1). There are some changes in ADV activity in order to achieve its goal as an eraser as follow:

ADV starts to roam the network at time t_{ADV} which is randomly chosen between t=0 and t_{sink}, and migrates from node to node continuously until the sink arrives. For each visited node, it corrupts the sensor by deleting all collected data storing in the sensor's buffer. We assume that ADV only erases collected data, it thus does not interfere with any other sensor setup. The number of corrupted nodes *M* is calculated by the same Equation 2 as in Basic model.

5.2.2 RaDD against Eraser ADV

Recall that RaDD is a random distribution method which replicates sensed data within the network to provide data survivability. Data survivability here is defined as the existence of at least one data replication at the time of sink arrival. As with previous models, all collected packets are treated equally in the network, thus we examine the chance to survive of one data packet which is called the target data. The target data is sensed by node A at time t_{target}. Node A stores the collected data into its memory and broadcast it to its neighbouring nodes. The following tables describe pseudo code of RaDD against Eraser ADV. Notation was given in Table II. It should be noted here that the algorithm is almost the same as the basic RaDD except for the deletion part. Because when a packet has been stored and/or scheduled to be forwarded and an ADV visits the node, then the packet is permanently deleted and the broadcast operation cannot be executed.

Table XXII: Main function of RaDD against Eraser ADV

MAIN FUNCTION

If (node *i* recieves a new packet j) **then** {

Generate two random numbers *X* and *Y* between 0 and 1

If ($X \leq P$) **then** { /* node decides to store the received packet*/

$t_{str_{i,j}} = t$ /*node *i* stores packet *j* at time $t_{str_{i,j}}$ */

If ($Y \leq Q$) **then** { /* node decides to forward the received packet*/

$\boldsymbol{Forward\ (i,j)}$ /* call *Forward* function*/ }

} **else** { /* node decides not to store the received packet*/

If ($Y \leq Q$) **then** { /* decide to forward the received packet*/

$\boldsymbol{Forward\ (i,j)}$ /* call *Forward* function */ }

else { $\boldsymbol{Discard\ (i,j)}$ } /* decide not to store or forward */ }

}

}

Table XXIII: Forward function of RaDD against Eraser ADV

FORWARD FUNCTION

$\boldsymbol{Forward\ (node\ i, packet\ j)}$ {

Generate a random number k in (t_1, t_2)

If $(t_{rcv_{i,j}} + k < t_{sink})\&\&((t_{ADV_i} < t_{rcv_{i,j}})||(t_{ADV_i} > t_{rcv_{i,j}} + k))$ **then** {

/*if packet *j* has not been deleted by the time of forwarding, node *i* forwards pkt *j* at time $t_{fw_{i,j}}$ equal to received time plus a random time */

$t_{fw_{i,j}} = t_{rcv_{i,j}} + k$ }

else $\boldsymbol{Do\ nothing}$ /* packet was deleted*/

}

Table XXIV: Eraser ADV activity

ERASER ADVERSARY ACITIVITY
$S = \{0, 1, 2, \ldots, N\}$ /*set of all sensors (ID) in the networks*/
$t_{ADV} = rand(0, t_{sink})$ /*ADV starts corrupting the network at time t_{ADV}*/
$C_{ADV} = \{\}$ /* Initialize the set of sensor ADV has corrupted*/
while $(t < t_{sink})$ **do** { /* while the sink has not arrived*/
Pick one random node in $S \backslash \{C_{ADV}\}$, assume node i is selected
/* ADV picks node *i* to corrupt such that ADV has not visited it before */
$t_{ADV_i} = t_{ADV}$ /* ADV corrupts node *i* at time t_{ADV_i} */
DELETE(node *i*, data) /* deletes all stored data*/
$t_{ADV} += \delta$ /*ADV finishes corrupting node *i* */
$C_{ADV} + \{i\}$ /* add node *i* into set of visited nodes*/
} **end while**

5.2.3 Simulation results

Simulation setup is the same as the one in the basic model. Simulations were run for the worst case where the target data is sensed at time t=0, and ADV arrives as early as it is able to corrupt every single node in the network ($t_{ADV} = 0.1$).

Figures 35 and 36 show the number of replications versus storing probability P and forwarding Q in the case an Eraser ADV has visited every single node in the network. N_{repli} and L_{repli} represent for the number of data replications by Random distribution process and the number of survived replications at the time sink' arrival, respectively. It is shown that N_{repli} and L_{repli} increase linearly with P and also increase linearly with Q when it is approximately less than 0.5 (behaviour of RaDD). When Q is roughly greater

than 0.5, N_{repli} and L_{repli} are nearly steady with increasing of Q. This is because the network produces redundant broadcasts as we already mentioned in previous Chapters. It is also shows that there survives at least one replication when P=0.2 or Q=0.2 (the purple lines in both graphs).

In conclusion, the basic RaDD, which produces data replications and randomly distributes them within the network, can guarantee data survival against an Eraser ADV even when it arrives early enough to corrupt all nodes.

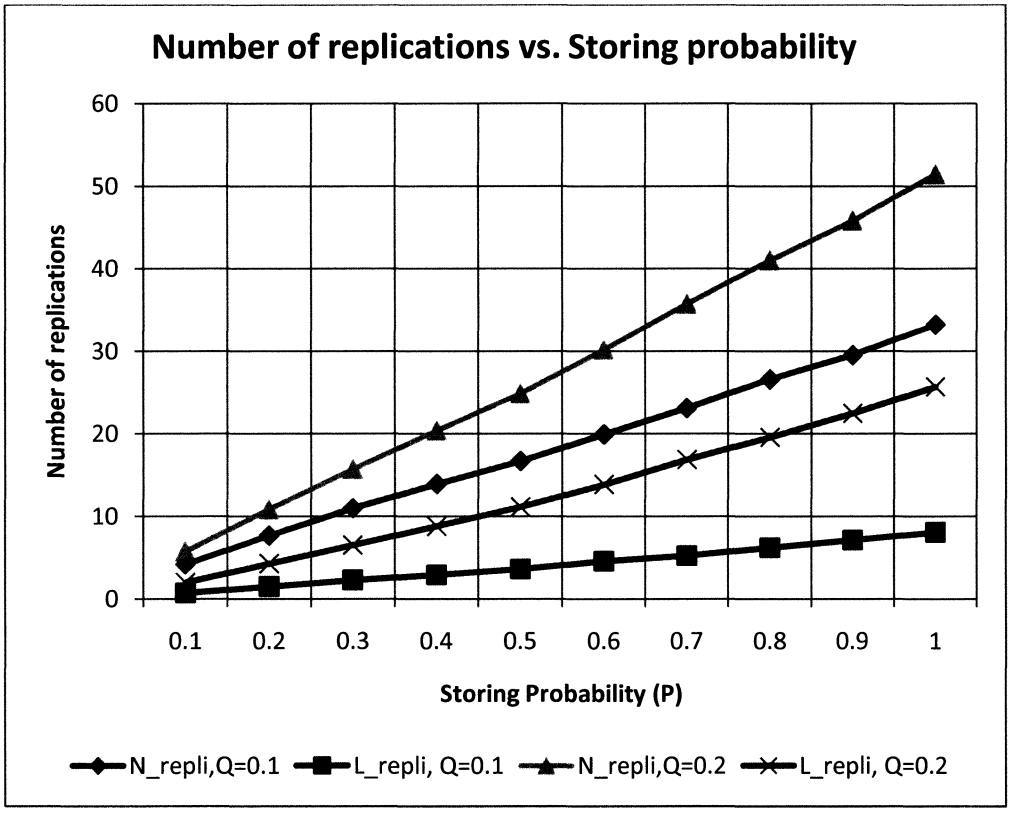

Figure 35: Number of replications versus storing probability *P*

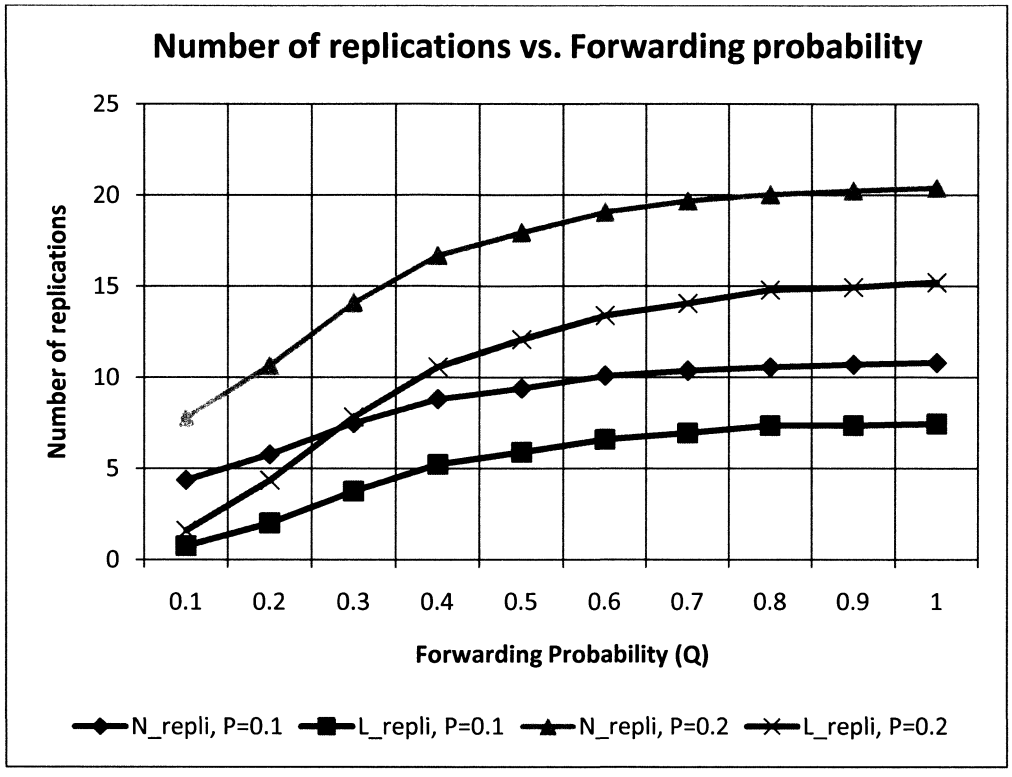

Figure 36: Number replications versus forwarding probability *Q*

5.3 Dynamic RaDD

Dynamic RaDD is based on basic RaDD but with changing values for *P* and *Q* dynamically. Our motivation for furthering our investigations into this field is to reduce storage and broadcast messages when an ADV has not presented in the network. The idea of our algorithm is to define initial small values of *P* and *Q* at the time of deployment of sensors *(P1, Q1).* In this way, data spreading in the network can be reduced, which saves both sensor storage and broadcast messages.

If a sensor detects that there is the presence of an ADV in the network, it broadcasts a warning message to its neighbour indicating that there are attackers potentially nearby. Its neighbouring nodes continue spreading the warning message within the network. The

warning message is a small notice so it does not overwhelm the network. When a node receives the warning message, it increases both its storing and forwarding probabilities to *P2* and *Q2,* respectively. It should be noted here that *P1, Q1, P2* and *Q2* are determined prior to the time of deployment.

Simulation

We use the same simulation settings in Chapter 3 except for implementing of dynamic *P* and *Q*. Tables XXV and XXVI show the simulation results of Dynamic and Static RaDD respectively. We can see that N_{target} of dynamic RaDD decreases compare to static RaDD, thus there is a saving in storage. Likewise, *B_M* of dynamic RaDD also decreases, thus there is a saving in broadcast messages as well. However, $P_H(\%)$ of Dynamic RaDD is not higher than the static one. This is mainly because our model consists of 100 nodes, which results in a small N_{target} with small initial *P1* and *Q1*. Hence, there is an insufficient spread of data into the network. When an ADV arrives, it may corrupt almost all of the original data. Furthermore, as forwarding data time is scheduled randomly, some nodes would forward data after an ADV arrives.

However, Dynamic RaDD might be more effective with larger network such as 1000 nodes or more. In such network, it would result higher N_{target} even with small *P* and *Q*. Our future work will include this enlarged network for further investigation of dynamic scheme.

Table XXV: Dynamic RaDD

	N_{target}	P_H *of RaDD*	P_H *of RaDD w/ DR*	*B_M of RaDD*	*B_M of RaDD w/ DR*	t_{ADV}
P1=0.1						
Q1=0.05	4.84	21.35%	48.52%	51.36	77.49	1.1
P2=0.2	4.01	29.84%	58.35%	44.68	78.82	1.6
Q2=0.1	3.97	35.94%	67.12%	44.49	80.86	2.1
t_{target}=*0*	3.86	46.93%	79.78%	43.44	86.57	2.6
	3.87	57.14%	84.46%	43.65	97.75	3.1

Table XXVI: Static RaDD

	N_{target}	P_H *of RaDD*	P_H *of RaDD w/ DR*	*B_M of RaDD*	*B_M of RaDD w/ DR*	t_{ADV}
P=0.2	8.04	21.6%	65.0%	80.87	125.22	1.1
Q=0.1	8.04	30.3%	78.4%	80.87	140.98	1.6
t_{target}=*0*	8.06	39.5%	85.5%	81.11	154.12	2.1
	8.06	49.5%	87.7%	81.11	172.61	2.6
	8.08	59.2%	91.3%	81.28	188.70	3.1

Chapter 6: Conclusion and Future Work

In this thesis, we have proposed simple algorithms to protect the data collected in an unattended wireless sensor network; our solutions are not based on cryptographic strategies but rather on random distribution of the data. In our basic algorithm (called RaDD), sensors collaborate to propagate the collected data within the network. Each sensor node makes its own random decision to store a packet locally with probability P, and/or to forward a packet with probability Q. As a result of RaDD process, the data are located at various locations, the adversary is unlikely to corrupt all the nodes that already have a local copy of the data, before the sink arrival. Moreover, by assuming that a node can detect whether its memory content is altered by an external entity, we can enhance RaDD with Damage Recovery (DR) and/or Damage Control (DC). DC re-broadcasts data in a second phase and enables corrupted nodes to recover adversarial attacks. DC limits the propagation of the corrupted data. Enhanced RaDD with DR and/or DC brings significant improvements of the data survival rate, especially when the number of corrupted nodes is high. DR works better with higher storing probability P, whereas DC works better with higher forwarding probability Q. Simulation results showed that the proposed algorithms can guarantee the network survival (with the appropriate choice of

the two parameters P and Q) , even when the adversary corrupted a large number of nodes.

Future work

Our focus during this research was on data survivability in a homogeneous UWSN which is an emerging research area and such there are many issues subjected to be investigated. The possible directions needed to be further explored in this domain would include more realistic models by taking into consideration energy consumption and network failures. We would examine different types of ADV in term of its strategy and behaviour to see how our algorithms can deal with different types of attacks. We will consider a larger UWSN, a heterogeneous UWSN and a UWSN with mobile sensors. We also try to employ cryptography along with Random distribution to improve network performance and to provide data confidentiality.

Bibliography

[1] J. Stankovic, "Wireless Sensor Networks," Chapter in Handbook of Real-Time and Embedded Systems, CRC Press, Charlottesville, Virginia 22904, 2006

[2] http://en.wikipedia.org/wiki/Wireless_sensor_network. Last accessed: July 2010

[3] http://sprout.ics.uci.edu/projects/uwsnwebpage/security_uwsn.html. Last accessed : July 2010

[4] R. Di Pietro, L. Mancini, C. Soriente, A. Spognardi, and G. Tsudik, "Catch Me If You Can: Data survival in unattended sensor networks," in the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom'08), pages 185–194, 2008.

[5] R. Di Pietro, L. Mancini, C. Soriente, A. Spognardi, and G. Tsudik, "Playing hide-and-seek with a focused mobile adversary," Cryptology ePrint Archive, Report 2008/293, 2008.

[6] R. Di Pietro, L.V. Mancini, C. Soriente, A. Spognardi, and G. Tsudik, "Data security in Unattended Wireless Sensor Networks," IEEE Transactions on Computers, Special Issue on Autonomic Network Computing, 2009.

[7] R. Di Pietro, C. Soriente, A. Spognardi, and G. Tsudik, "Collaborative authentication in unattended Wireles Sensor Networks," in the Conference of ACM on Wireless Network Security (ACM WiSec'09), 2009.

[8] R. Di Pietro, G. Oligeri, C. Soriente, and G. Tsudik, "Intrusion-resilience in mobile unattended Wireles Sensor Networks," In Infocomm, 2010.

[9] W. Ren, Y. Ren, and H. Zhang, "HybridS: A Scheme for Secure Distributed Data Storage in Wireless Sensor Networks," Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference, pages 318 – 323, Dec. 2008.

[10] W. Ren, J. Zhao, and Y. Ren, "MSS: A Multi-Level Data Placement Scheme for Data Survival in Wireless Sensor Networks," Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference, Page(s):1 – 4, Sept. 2009.

[11] Y. Wang, G. Attebury, and B. Ramamurthy, "A survey of security issues in wireless sensor networks," *IEEE Commun. Surveys Tutorials*, vol. 8, pages 2–23, 2006.

[12] G. Padmavathi and D. Shanmugapriya, "Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks, " (IJCSIS) International Journal of Computer Science and Information Security, Vol. 4, No. 1 & 2, 2009

[13] Z. Ruan, X. Sun, W. Liang, D. Sun and Z. Xia, "CADS: Co-operative Anti-fraud Data Storage Scheme for Unattended Wireless Sensor Networks," Inform. Technol. J., 9: 1361-1368, 2010.

[14] R. Di Pietro, L.V. Mancini, C. Soriente, A. Spognardi, and G. Tsudik, "Maximizing data survival in unattended wireless sensor networks against a focused mobile adversary," Special Issue on Privacy and Security in Wireless Sensor and Ad Hoc Networks, 2009.

[15] T. Vo and J.Talim, "Random Distribution for Data Survival in Unattended Wireless Sensor Networks," sensorcomm, 2010 Fourth International Conference on Sensor Technologies and Applications, pages 468-471, 2010.

[16] D. Ma, C. Soriente, and G. Tsudik, "New adversary and new threats: Security in unattended sensor networks," IEEE Network, March 2009.

[17] D. Ma and G. Tsudik, "Security and Privacy in Emerging Wireless Networks," IEEE Wireless Communications, Special Issue on Security and Privacy in Emerging Wireless Networks, 2010

[18] http://en.wikipedia.org/wiki/Cryptography. Last accessed: August, 2010.

[19] D. Ma and G. Tsudik, "DISH: Distributed self-healing in unattended wireless sensor networks," In 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'08), pages 47–62, 2008.

[20] R. Di Pietro, D. Ma, C. Soriente, and G. Tsudik, " POSH: Proactive co-operative self-healing in unattended sensor networks," in 27th IEEE International Symposium on Reliable Distributed Systems (SRDS'08), pages 185–194, 2008.

[21] N. Subramanian, C. Yang, and W. Zhan, "Securing distributed data storage and retrieval in sensor networks," Pervasive and Mobile Computing Journal (Special Issue for PerCom 2007), 3(6):659–676, 2007.

[22] A. Yavuz and P. Ning, "Hash-based sequential aggregate and forward secure signature for unattended wireless sensor networks," In the Sixth Annual

International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2009), pages 1-10, 2009.

[23] D. Ma and G. Tsudik, "Forward-secure sequentical aggregate authentication," In Proceedings of IEEE Symposium on Security and Privacy 2007, May 2007.

[24] M. Shao, S. Zhu, W. Zhang, and G. Cao, "pDCS: Security and privacy support for data-centric sensor networks," In INFOCOMM'07, pages 1298–1306, 2007.

[25] C. Yu, C. Chen, C. Lu, S. Kuo, and H. Chao, "Acquiring Authentic Data in Unattended Wireless Sensor Networks," Sensors 2010, doi:10.3390/s100402770, volume 10, number 4, pages 2770-2792, 2010.

[26] A. Sharmir, "How to share a secret," Communications of the ACM, vol. 22, no. 11, pages 612–613, November 1979.

[27] http://en.wikipedia.org/wiki/Secret_sharing. Last accessed: September, 2010

[28] W. Ren, J. Zhao, and Y. Ren, "Network coding based dependable and efficient data survival in unattended wireless sensor networks," Journal of Communications, 4(11):894–901, Dec 2009.

[29] W. Ren, Y. Ren, and H. Zhang, "*H*2S: A Secure and Efficient Data Aggregative Retrieval Scheme in Unattended Wireless Sensor Networks," the Fifth International Conference on Information Assurance and Security, 2009.

[30] S. Ganeriwal and M. Srivastava, "Reputation-based framework for high integrity sensor networks," In Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks, Washington DC, USA, 2004.

[31] P. Cholewinski, Z. Benenson, and F. Freiling, "Simple Evasive Data Storage in Sensor Networks," Proc. Int'l Conf. Parallel and Distributed Computing Systems (PDCS '05), pages 779-784, 2005.

[32] T. Hara, "Effective Replica Allocation in Ad Hoc Networks for Improving Data Accessibility," Proc. IEEE INFOCOM 2001, pages 1568-1576, 2001.

[33] T. Hara, "Replica Allocation Methods in Ad Hoc Networks with Data Update," ACM-Kluwer J. Mobile Networks and Applications, pages 343-354, 2003.

[34] T. Hara and S. Madria, "Dynamic Data Replication Schemes for Mobile Ad-Hoc Network Based on Aperiodic Updates," Proc. Int'l Conf. Database Systems for Advanced Applications (DASFAA '04), pages 869-881, 2004.

[35] T. Hara, N. Murakami, and S. Nishio, "Replica Allocation for Correlated Data Items in Ad-Hoc Sensor Networks," ACM SIGMOD Record, pages. 38-43, 2004.

[36] T. Hara and S. Madria, "Consistency Management Among Replicas in Peer-to-Peer Mobile Ad Hoc Networks," Proc. Int'l Symp. Reliable Distributed Systems (SRDS '05), pages 3-12, 2005.

[37] T. Hara and S.K. Madria, "Data Replication for Improving Data Accessibility in Ad Hoc Networks," IEEE Trans. Mobile Computing, vol. 5, no. 11, pages 1515-1532, Nov. 2006.

[38] S. Khan, A. Maciejewski, H. Siegel, and I. Ahmad, "A game theoretical data replication technique for mobile ad hoc networks, " Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium, pages 1-12, 2008.

[39] V. Gianuzzi, "Data Replication Effectiveness in Mobile Ad-Hoc Networks," Proc. ACM Workshop Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN '04), pages 17-22, 2004.

[40] A. Detti, L. Bracciale, and F. Fedi, " Robust Data Replication Algorithm for MANETs with Obstacles and Node Failures, " Communications (ICC), 2010 IEEE International Conference, pages 1 - 6, 2010

[41] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth Codes: Maximizing Sensor Network Data Persistence," SIGCOMM Computer Comm. Rev., vol. 36, no. 4, pages 255-266, 2006.

[42] A.P. Speer and I. Chen, "Effect of Redundancy on Mean Time to Failure of Wireless Sensor Networks," Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06), 2006.

[43] S. Ganeriwal, S. Capkun, C. Han, and M. Srivastava, "Secure time synchronization service for sensor networks," In WiSe, 2005.

[44] B. Williams and T. Camp, "Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks," Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'02), pages 194-205, 2002.

[45] A. Perrig, J.A.Stankovic, and D. Wagner, "Security in wireless sensor networks," Communication of ACM, vol. 47, no. 6, pages 53–57, 2004.

[46] X. Chen, K. Makki, K. Yen, and N. Pissinou, "Sensor Network Security: A Survey," IEEE communications surveys & tutorials, vol. 11, no. 2, second quarter 2009

[47] Q. Zhang, T. Yu, and P. Ning, "A framework for identifying compromised nodes in wireless sensor networks," in ACM Transactions in Information and Systems Security (TISSEC), 2008.

[48] J. C. McEachen and J. Casias, "Performance of a wireless unattended sensor network in a freshwater environment," in HICSS '08: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, 2008.

[49] C. Hartung, J. Balasalle, and R. Han, "Node compromise in sensor networks: The need for secure systems," University of Colorado at Boulder, Technical Report TR-CU-CS-990-05, 2005.

[50] H. Song, L. Xie, S. Zhu, and G. Cao, "Sensor node compromise detection: The location perspective," in *Proc. International Conf.* Wireless Commun. Mobile Computing, pages 242–247, 2007.

[51] G. Gaubatz, J.-P. Kaps, E. Ozturk, and B. Sunar, "State of the art in public-key cryptography for wireless sensor networks," in *Proc. 3rd* IEEE International Conf. Pervasive Computing Commun. Workshops *(PERCOMW)*, pages 146–150, 2005.

[52] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "A survey on sensor networks, " IEEE Communications Magazine, August 2002.

[53] http://www.isi.edu/nsnam/ns. Last accessed: December 2009.

[54] I. Downard, "Simulating Sensor Networks in NS-2, " NRL/FR/5522--04-10073, Naval Research Laboratory, Washington, D.C., May 2004.

[55] M. Jevtic, N. Zogovic, and G. Dimic, "Evaluation of Wireless Sensor Network Simulators," 17th Telecommunications forum TELFOR 2009 Serbia, Belgrade, November 24-26, 2009

[56] T.O. Walker, M. Tummala, and J. McEachen, "Energy Efficiency of Centralized and Distributed Computation in Unattended Multi-Hop Wireless Sensor Networks for Battlefield Monitoring," System Sciences (HICSS), 43rd Hawaii International Conference on pages, 1-9, March 2010.

[57] http://www.vmware.com/pdf/ws71_manual.pdf. Last accessed: October 2009.

[58] http://www.vmware.com/pdf/GuestOS_guide.pdf. Last accessed: October 2009.

[59] http://nsnam.isi.edu/nsnam/index.php/Downloading_and_installing_ns-2. Last accessed: October 2009.

[60] http://vanitdiscovery.blogspot.com/2009/05/ns-2-installation-on-ubuntu-904.html. Last accessed: October 2009.

Appendix

NS-2 Installation on Ubuntu 9.04 includes four sections:

1. Introduction
2. Installing VMWare Workstation 7.0.1 on a Windows Host
3. Installing Ubuntu 9.04
4. Installing NS-2 on Ubuntu based System

1. Introduction:

NS-2 (Network Simulator) is designed to run from on most UNIX/LINUX based operating systems. NS-2 possible to run NS-2 on Windows machines using Cygwin. NS-2 can also use a virtual Linux machine and run that under Windows. VMware Workstation that allows me to run Linux systems like Ubuntu 9.04, which I use to run my scenario for my thesis.

2. The procedure for installing VMWare Workstation 7.01 on a Windows Host

Installation Steps:

1. From the Start menu, choose Run and specify the path to either the CD/DVD drive or the downloaded installer files:

- If you are installing from the installation media, enter D:**setup.exe,** where D: is the drive letter for your CD/DVD drive.

- If you are installing from a downloaded file, browsed the directory where you saved the downloaded installer file, and run the installer.

On Windows 7, when the User Account Control dialog box prompts you for permission to run the installer, click **Continue.**

2. When the wizard opens and finishes computing space requirements, click **Next.**

3. On the Setup Type page, select Typical.

4. Follow the wizard prompts to complete the installation.

Some installations might require that you reboot the computer.

3. The procedure for installing Ubuntu 9.04 in VMWare Workstation 7.0.1

Installation Steps:

1. Insert the Ubuntu 9.04 CD in the CD/DVD drive.
2. Power on the virtual machine

After the Ubuntu 9.04 installer copies the files it needs to the virtual disk, it ejects the installation CD and displays a message indicating that the computer will restart.

3. Follow the prompts to complete the installation.

4. The procedure for installing NS-2 in Ubuntu 9.04

Installation Steps:

1. Download a copy of ns-allinone-2.34.tar.gz (http://ftp.isi.edu/nsnam/ns/ns-build.html#allinone)
2. Place it in somewhere, e.g. /home/programmer, and then extract it.
   $ tar –xzf ns-allinone-2.34.tar.gz
3. Download and install some packages from repository
   $ sudo apt-get install build-essential autoconf automake libxmu-dev
4. Install the ns-2
   $ cd ns-allinone-2.34

   $ ./install

After a long wait and a whole lot of text, the installation finishes up with the text like the following:

Nam has been installed successfully.
Ns-allinone package has been installed successfully.
Here are the installation places:
tcl8.4.11: /home/myy/Programmer/ns-allinone-2.34/{bin,include,lib}
tk8.4.11: /home/myy/Programmer/ns-allinone-2.34/{bin,include,lib}
otcl: /home/myy/Programmer/ns-allinone-2.34/otcl-1.13
tclcl: /home/myy/Programmer/ns-allinone-2.34/tclcl-1.18
ns: /home/myy/Programmer/ns-allinone-2.34/ns-2.34/ns
nam: /home//myy/Programmer/ns-allinone-2.34/nam-1.14/nam
xgraph: /home/myy/Programmer/ns-allinone-2.34/xgraph-12.1

gt-itm: /home/myy/Programmer/ns-allinone-2.34/itm, edriver, sgb2alt, sgb2ns, sgb2comns, sgb2hier

5. Edit some paths
$ gedit ~/.bashrc

Put the following lines below on that file. Off courses, you might change /home/myy/Programmer for it depends on where you extract ns-allinone-2.34.tar.

 # LD_LIBRARY_PATH
 # OTCL_LIB=/home/myy/Programmer/ns-allinone-2.34/otcl-1.13
 # NS2_LIB=/home/myy/Programmer/ns-allinone-2.34/lib
OTCL_LIB=/home/myy/Programmer/ns-allinone-2.34/otcl-1.13
NS2_LIB=/home/myy/Programmer/ns-allinone-2.34/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB
:$X11_LIB:$USR_LOCAL_LIB
 # TCL_LIBRARY
TCL_LIB=/home/myy/Programmer/ns-allinone-2.34/tcl8.4.18/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

 # PATH
 # XGRAPH=/home/myy/Programmer/ns-allinone-2.34/bin:/home/myy/Programmer/ns-allinone-2.34/tcl8.4.18/unix:/home/myy/Programmer/ns-allinone-2.33/tk8.4.18/
 # unix:/home/myy/Programmer/ns-allinone-2.33/xgraph-12.1/
 # XGRAPH=/home/myy/Programmer/ns-allinone-2.34/bin:/home/myy/Programmer/ns-allinone-2.34/tcl8.4.18/unix:/home/myy/Programmer/ns-allinone-2.33/tk8.4.18/unix
XGRAPH=/home/myy/Programmer/ns-allinone-2.34/bin:/home/myy/Programmer/ns-allinone-2.34/tcl8.4.18/unix:/home/myy/Programmer/ns-allinone-2.34/tk8.4.18/unix

 # NS=/home/programmer/ns-allinone-2.33/ns-2.33/

NS=/home/myy/Programmer/ns-allinone-2.34/ns-2.34/ns

 # NAM=/home/programmer/ns-allinone-2.33/nam-1.13/

NAM=/home/myy/Programmer/ns-allinone-2.34/nam-1.14/nam

export PATH=$XGRAPH:$NS:$NAM:$PATH

6. Validate it (this step is taking very long time)
$ cd ns-2.34

$ ./validate

7. Create a symlink, so that ns-2 can be called from anywhere
$ sudo ln -s /home/myy/Programmer/ns-allinone-2.34/ns-2.34/ns /usr/bin/ns

8. Let it take effect immediately

$ source ~/.bashrc

9. Try to run it

$ ns

10. If the installation success, you will see % at the command prompt. Type following command to exit

% exit

The instruction was summarized from the following websites (last accessed: October 2009):

http://www.vmware.com/pdf/ws71_manual.pdf

http://www.vmware.com/pdf/GuestOS_guide.pdf

http://nsnam.isi.edu/nsnam/index.php/Downloading_and_installing_ns-2

http://vanitdiscovery.blogspot.com/2009/05/ns-2-installation-on-ubuntu-904.html