CLASSIFICATION AND EXPERIMENTAL ANALYSIS OF NODE CLONE DETECTION TECHNIQUES IN WSN

Project report submitted in fulfillment of the requirement for the degree of

Bachelor of Technology

in

Computer Science and Engineering

By

Priya Saini (121309)

Under the supervision of

Dr. Yashwant Singh

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CERTIFICATE

Candidate's Declaration

I hereby declare that the work presented in this report entitled "**Classification and Experimental Analysis of Clone Node Detection Techniques in WSN**" in fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2015 to June 2016 under the supervision of **Dr. Yashwant Singh,** Assistant Professor(Senior Grade) in **Computer Science and Engineering** department.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

[illegible]

Priya Saini(121309)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

[illegible]

Supervisor Name: Dr. Yashwant Singh

Designation: Assistant Professor(Senior Grade)

Department name: Computer Science and Engineering

Dated: 10|06|2016

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. I am grateful to my project guide **Dr. Yashwant Singh** for the guidance, inspiration and constructive suggestions that helped me in the preparation of the project.

I also thank my colleagues who have helped me in successful completion of the project.

[illegible]

Date: 10/6/2016

Priya Saini (121309)

TABLE OF CONTENT

LIST OF ABBREVIATIONS

Abbreviation	Word
WSN	Wireless Sensor Network
sWSN	Static wireless sensor network
SMP	Stable Marriage Problem
RED	Randomized, Efficient and Distributed
XED	Extremely Efficient Detection
RAWL	Random Walk
TRAWL	Table assisted Random Walk
NDFD	Non-deterministic and Fully Distributed
MEMS	Micro-Electro Mechanical System
VLSI	Very Large Scale Integration
DoS	Denial of Service

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

ABSTRACT

Wireless Sensor Networks (WSNs) are specialized network of large number of sensor equipped nodes which are spatially deployed over the region of interest where the nodes may be stationary or mobile in nature. Wireless sensor networks are a group of small untethered sensor devices which are densely populated in a specific target area, where they collaborate in an ad-hoc manner to sense phenomena and report sensed data for various uses. Sensor nodes are vulnerable to a number of attacks and hence its security can be compromised by an attacker. The node replication attack or the clone node attack is a security threat where an attacker creates its sensor nodes and joins the network as if they are the legitimate nodes of the network. For this attack to happen, the attacker will physically capture one node from the network and extract all the secret information of the node such as node ID, Keys etc. Using the extracted information, the attacker creates many replicas/clones of the compromised node. These clones are then deployed into the WSN at suitable positions and start to attack the whole network internally which will hamper the network.

In this project, I have proposed extended SMP algorithm for clone node detection. This algorithm has already been used for clone code detection. This algorithm works only on static networks. Also, a comparative study of already existing techniques has been done.

CHAPTER- 1
INTRODUCTION

1.1 INTRODUCTION

The phenomenal advancement in technologies like micro-electromechanical systems (MEMS), very large scale integration (VLSI) and wireless communications which facilitates the growth of low cost, low power, multifunctional sensor nodes, which are small in size with limited processing and computing resources and they are inexpensive[1]. These features have made WSNs popular and these are being used in many real life problems.

1.1.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) is a collection of nodes organized into a cooperative network. Each node consists of processing capability (one or more microcontrollers, CPUs or DSP chips), may contain multiple types of memory (program, data and flash memories), have a RF transceiver (usually with a single omni-directional antenna), have a power source (e.g., batteries and solar cells), and accommodate various sensors and actuators [2]. These sensor nodes are spatially deployed over a region of interest and they collaborate in order to achieve a common goal. These systems can scale from tens to thousands nodes and seamlessly integrate with existing wired measurement and control systems.

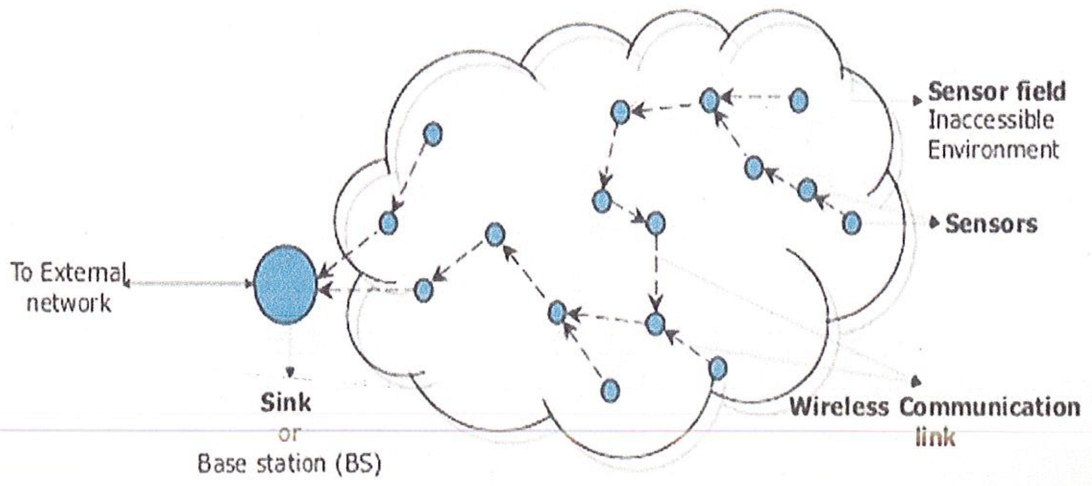


Figure 1: Wireless Sensor Network [1]

1.1.2 Applications of WSNs

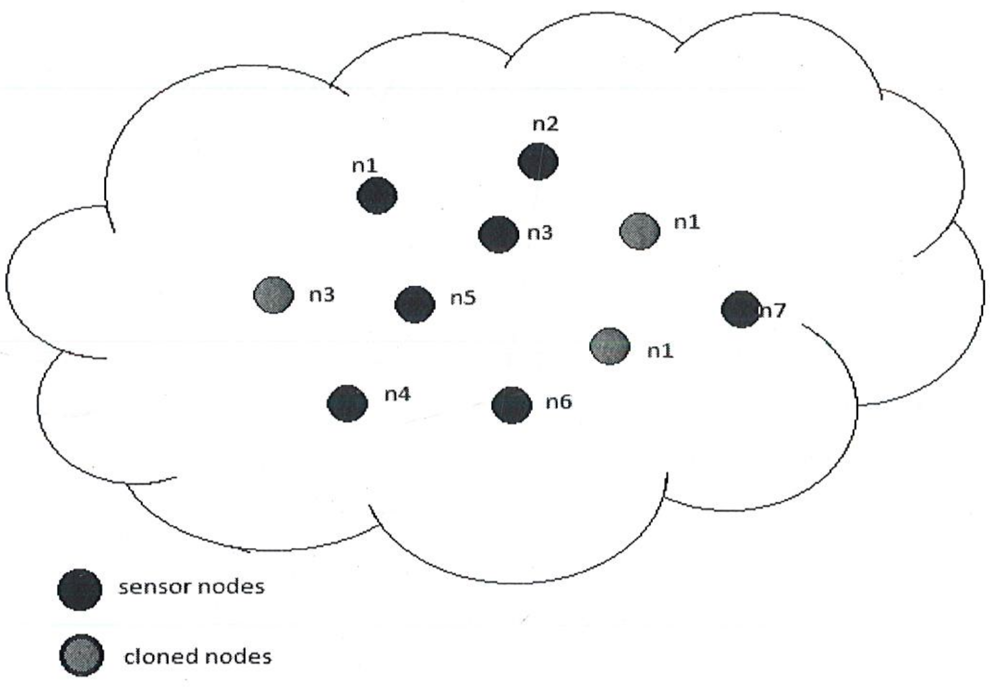
WSNs offer viable solutions to a number of real life problems. Using WSNs, one can easily monitor his assets and environment. These can be used for monitoring factory instrumentation, pollution levels, freeway traffic and structural integrity of building. WSNs can also be used for healthcare monitoring, climate sensing, landslide detection, water quality monitoring, control in office buildings, and home environmental sensing systems for temperature light, moisture, and motion [3]. WSNs are also being used in military areas as they are inexpensive, self organizing and don't need constant supervision. So they are often deployed in remote areas with harsh and hostile environment where it is humanly impossible to perform monitoring task personally.

1.1.3Security Issues in WSNs

Due to budget constraints, hence low-cost hardware, these sensor nodes are not tamper-resistant. Therefore, WSNs are facing a new set of security threats. Attacks on WSN on the basis of damage level or attackers' access level can be categorized as:

- **Active Attacks:** In an Active Attack, an attacker tries to modify or remove the information exchanged on the network, meaning interruption, modification or fabrication of information. Malicious acts are taken out against data confidentiality and data integrity. Examples of active attacks include jamming, impersonating, message modification, denial of service (DoS), black hole, sybil attack, flooding, clock skewing and message replay. [1,4]
- **Passive Attacks:** The Passive Attack is limited to listening and analyzing the exchanged traffic by unauthorized attackers. This type of attack is easier to realize and it is difficult to detect. Since, the attacker does not make any modification on exchanged information, the intention of the attacker is the knowledge of confidential information or the knowledge of the significant nodes in the network (cluster head node), by analyzing routing information, to prepare an active attack. Examples of passive attacks are eavesdropping, traffic analysis, and traffic monitoring.[4]

The Sensor nodes are often deployed in unattended environments where they are vulnerable to capture and compromise by an adversary. Once a sensor is compromised, the information inside is easily accessible. An adversary may replicate sensors using secret credentials (ID, cryptographic keys, etc) of compromised node and deploy at least one copy of functional node in the network to launch a variety of insider attacks. This attack process is broadly termed as a **Node Clone Attack**. Using these cloned nodes, the adversary can gain control over the network. Thus the security of the network is compromised. These cloned nodes can be used to launch a number of insider attacks. If these clones are left undetected, the network is unshielded to attackers and thus extremely vulnerable. [5]



Clone Node Attack in WSN

Figure 2

Therefore in this paper, use of a variant of Stable Marriage Problem (SMP) algorithm is proposed for detection of node clones in wireless sensor networks. Using this algorithm, pairs of nodes are formed. If any two nodes form a stable matching pair, then they are considered as clones of each other.

1.2 PROBLEM STATEMENT

Sensor nodes that are deployed in hostile environments are vulnerable to capture and compromise. An adversary may obtain private information from these sensors, clone and intelligently deploy them in the network to launch a variety of insider attacks. This attack process is broadly termed as a clone attack. Currently, the defenses against clone attacks are not only very few, but also suffer from selective interruption of detection and high overhead (computation and memory. The objective of our project is to classify the various clone detection approaches in wireless sensor network and also give the experimental analysis of these clone detection approaches in WSNs.

1.3 OBJECTIVES

Following are the objectives of our project-

- Study and comparative analysis of all node clone detection techniques
- Study of Stable Marriage Problem (SMP)
- Proposal and implementation of node clone detection using SMP algorithm

1.4 METHODOLOGY

Wireless Sensor Networks have a very wide domain of application which includes military services, health care, monitoring air pollution, factory instrumentation, etc. To work more safely and efficiently, sensor nodes are deployed in these harsh environments which is not feasible for humans. Working under these conditions, sensor nodes are generally prone to attacks by adversaries, which may try to obtain private information from these nodes. This information can be used for various kinds of attacks, out of which our main concern is *clone node attack*.

Earlier SMP was used for clone code detection, so working on the same grounds I propose this algorithm for clone node detection in wireless sensor networks. In our modified algorithm, I have tried to make stable pairs on the basis various parameters like node id, cryptographic key and performed various tests like energy test to verify our results. All the obtained stable pairs are treated as clone pairs which on successful detection, are removed from the network.

1.5 ORGANISATION

This paper is organized as follows: In Chapter 2, I have given an overview of the research papers I have read. The Chapter 3 briefs about the various system development phases of the project. The Chapter 4 contains the analysis of the proposed algorithm. And I conclude our discussion in Chapter 5, further discussing about the future scope of the project.

CHAPTER-2
LITERATURE SURVEY

To get familiar with the given topic, I studied various research papers that have been already proposed. I collected various papers related to the topic to gather all the information about the wireless sensor networks and the different attacks associated with it. Following is the abstract of few papers that I surveyed:

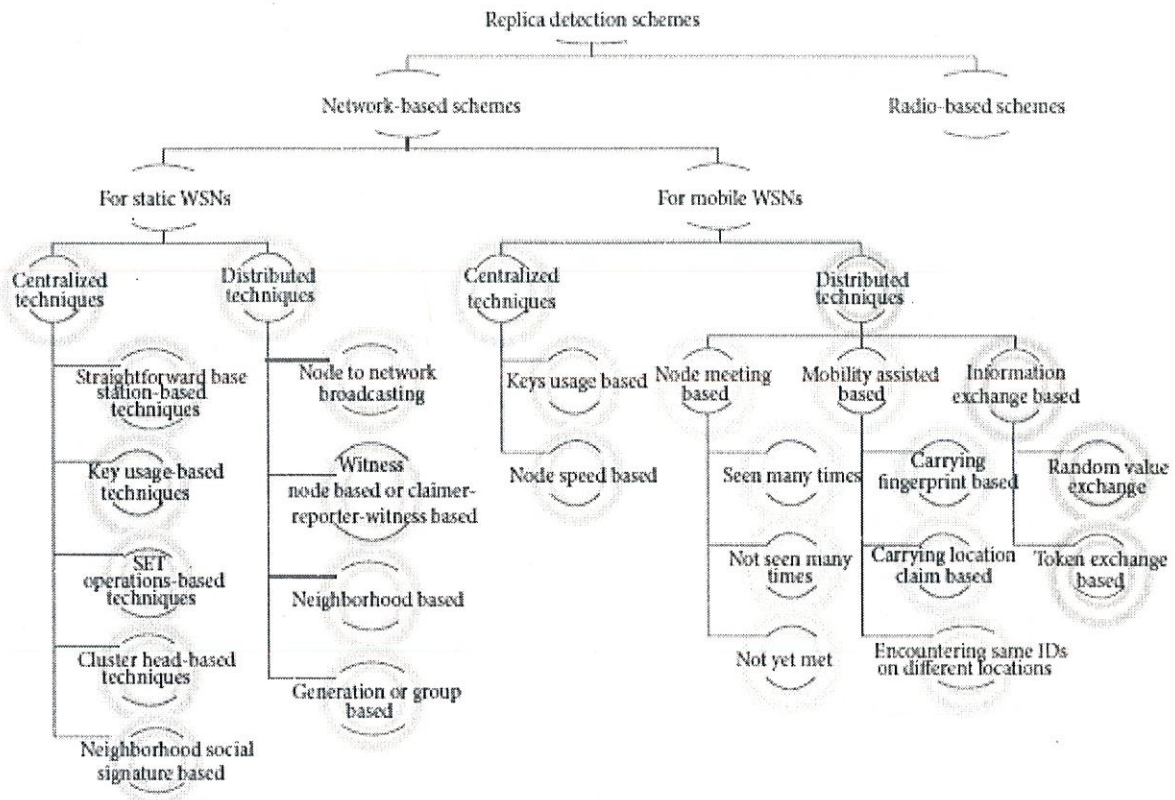
***2.1 Wazir Zada Khan, Mohammed Y. Aalsalem, Mohammed Nauful Bin Mohammed Saad, and Yaang Xing*, Detection and Mitigation of Node Replication Attacks in Wireless Sensor Networks: A Survey, 22 March 2013, 22 pages**

The paper introduces us to wireless sensor networks and various attacks related to it. It tells that wireless sensors networks are a collection of a number of tiny, low cost and resource constrained sensor nodes which are commonly not tamper proof. As a result, WSN are prone to a wide variety of physical attacks. In this paper, the main focus is on node replication attack, where the adversary creates its own low cost sensor nodes called clones and misinforms the network to acknowledge them as legitimate nodes. In this attack, an adversary physically captures one node and after collecting all secret credentials like node id, cryptographic key etc. an adversary replicates the sensor nodes and deploys one or more nodes of compromised node into the network at strategic locations.

The paper further categorizes the various types of attacks into two broad categories i.e. mainly for static and mobile WSNs. The prime difference between static and mobile WSNs is that mobile nodes are able to reposition and organize themselves in the network, and after initial deployment, the nodes spread out to gather information. Mobile nodes can communicate with one another when they are within the range of each other, and only then they can exchange their information gathered by them. Another important difference is that in static WSNs fixed routing or flooding is used for data distribution, while in mobile WSNs dynamic routing is used. As static and mobile WSNs differ in their characteristics hence replication detection schemes for stationary and mobile WSNs will be substantially different.

In a static or stationary WSN, a sensor node has a unique deployment position, and thus if one logical node ID is found to be associated with two or more physical locations, node replication is detected. But this is inapplicable to mobile WSNs where sensor nodes keep roaming in the deployment field. So, replication detection in such mobile WSN involves different scenarios and techniques.

For mobile WSNs, both centralized and distributed techniques have been proposed in the literature. In the case of stationary WSNs, centralized techniques are further categorized into five types, namely, straightforward base station-based technique, key usage-based technique, SET operations techniques, cluster head-based techniques and neighborhood social signature-based techniques. The distributed techniques for stationary WSNs are further divided into four types naming Node to Network Broadcasting, claimer-reporter-witness-based techniques, neighbor-based and generation- or group-based techniques. On the other hand, mobile centralized detection techniques are further divided into two types including key usage-based and node speed-based techniques. The mobile distributed detection techniques are divided into three main types, namely, node meeting-based, mobility-assisted-based, and information exchange- based techniques. This inclusive categorization can be summarized with Figure



Replica detection schemes
Network-based schemes
Radio-based schemes
For static WSNs
For mobile WSNs
Centralized techniques
Distributed techniques
Centralized techniques
Distributed techniques
Straightforward base station-based techniques
Key usage-based techniques
SET operations-based techniques
Cluster head-based techniques
Neighborhood social signature based
Node to network broadcasting
Witness node based or claimer-reporter-witness based
Neighborhood based
Generation or group based
Keys usage based
Node speed based
Node meeting based
Mobility assisted based
Information exchange based
Seen many times
Not seen many times
Not yet met
Carrying fingerprint based
Carrying location claim based
Encountering same IDs on different locations
Random value exchange
Token exchange based

Figure 3: Taxonomy of replica detection schemes[3]

2.2 *Sagar Chandra has Jagtap, Geetika Narang*, Survey on Distributed Detection of Clone Attacks in Wireless Sensor Networks, 25 May 2014, 6 pages

This paper also discusses about the desired properties of the wireless sensor networks and the effects of node clone attack on the network. Previous works against node clone attacks suffer from either a high communication or storage overhead or poor detection accuracy. In this paper, for the detection of nodes replication attacks, there is analysis of the desirable properties of a distributed mechanism. After that paper shows the solutions for the problem, later for detection of node replication attacks, paper proposes a self-healing RED protocol i.e. Randomized, Efficient, and Distributed protocol, which meets the requirements. Finally it shows that the protocol is highly efficient in the memory, communication and computation. Several algorithms are developed to detect the node-clone attacks, in static WSNs and mobile WSNs. Each one has its advantages and disadvantages. This paper surveys and standards based communication costs these algorithms and compares their performance as memory.

This paper also reviews different methods of detection of clone attacks and further discusses about their advantages and disadvantages. Based on the figure above, various techniques have been discussed and compared. The comparative analysis of the various techniques has been summarized in the table.

Method	Method Used	Advantages	Disadvantages
Centralized Algorithms for Static WSNs Preliminary Approach	1. Random Key Pre distribution 2. SET detection: 3. SPRT:	True zone-trust report	computation and storage overheads required for biased-SPRT
Centralized algorithms for mobile WSNs	1. SPRT[15]	Used random walk method	Computation and storage overheads
Distributed Algorithms for static WSNs Using Fingerprint	1. Node-to-network delivery[14] 2. Detection using Group Deployment[17] 3. Randomized Multicast[4] 4. Line Selected Multicast[5][12] 5. RED[3] 6. Agent Based Detection[2]	More security Failure of BS does not affect system Less storage requirement High detection speed	Higher communication overload in the network.
Distributed algorithms for mobile WSNs	XED[18],SDD[19],EDD[20],SEDD, UTSLE, MTLSD	Communication cost is constant. Location information is not required to detect clone node.	Vulnerable to the smart attackers High communication cost and less detection ratio.

Table 1: Comparative Study of techniques [8]

The paper further proposes a new Random Efficient Distributed (red) protocol proposal, and proves that the protocols meet the requirements above all the existing detection techniques. RED makes it difficult for the enemy to attack the network and offer further analytical results. Finally, the simulation shows that the RED memory as highly skilled; Need communication and computing and distributed to other protocols show better than attack detection probability.

2.3 Mohamed-Lamine Messai, Classification of Attacks in Wireless Sensor Networks, 24 April 2014, 5 pages

In this paper, the author has discussed about wireless sensor networks but the main focus is on security issues with WSN. Recently, there was a huge interest to propose security solutions in WSNs because of their applications in both civilian and military domains. Adversaries can launch different types of attacks, and cryptography is used to countering these attacks. In this paper, challenges of security, and classification of the different possible attacks in WSNs have been presented. The author has also discussed the attacker model of adversary.

The attacker model can be described as: The goal of an attacker (adversary) is to illegally obtain keys stored in nodes by vulnerabilities exploitation. In a realistic attacker model, the attacker is able to supervise a fixed percentage of communication channels after deployment. The author quotes- "The hostile surveillance is not ubiquitous during the deployment phase of the network and only fraction of the established link keys can be obtained by the attacker."

Layer based classification of attacks on wireless sensor networks is given below-

a) **Physical Layer:** This layer deals with the specification of frequency bands. The attacks associated with this layer are few but difficult to prevent. Some attacks on this layer are given below:
   - **Jamming:** Jamming is the type of attack which interferes with the radio frequencies used by network nodes. In this, an attacker sequentially transmits

over the wireless network refusing the underlying MAC protocol. Jamming can interrupt the network impressive if a single frequency is used throughout the network. In addition jamming can cause excessive energy consumption at a node by injecting impertinent packets. The receiver's nodes will as well consume energy by getting those packets.

- **Tampering:** Given physical access to a node, an attacker can extract sensitive information such as cryptographic keys or other data on the node. The node may also be altered or replaced to create a compromised node which the attacker controls. One defense to this attack involves tamper-proofing the node's physical package but due to budget constraints it is avoided.

- **Node Outage Attack:** The node outage attack is stopping the functionality of WSN components and the attacks apply physically or logically in network. The effects of this attack are stopping the node services such as reading, gathering and launching the functions. Availability and authenticities are main threats for this attack in network.

b) **MAC Layer:** This layer manages the access to the radio channel (MAC layer), and control errors. Attacks on this layer are:

- **Collision:** In this attack, message is transmitted by two nodes on a same frequency simultaneously. If attacker succeeds in doing so; then, at the receiving end; the message packets will be discarded due to checksum mismatch. The retransmission of packets could cause exhaustion of necessary resources i.e. energy of the sensor nodes.

- **Exhaustion Attack:** The adversary can only induce collision in one byte of a transmission to disturb the entire data packet, obligating the victim node to retransmit the data packet and cause a death of this node by consuming its energy. Such continuous collisions lead to exhaustion of its resources.

c) **Network Layer:** WSNs use communication multi-hops for routing the packets towards the destination. The attacks in this layer are: black hole, selective forwarding, Sybil attack, HELLO flood attack, wormhole, and Identity replication attacks. Some of these attacks are explained below:

- **Wormhole Attack:** In this attack, a malicious attacker receives packets from one location of network, forwards them through the tunnel (wormhole link) and releases them into another location. The wormhole link can be established by a variety of means, e.g., by using an Ethernet cable, a long-range wireless transmission, or an optical link. Once the wormhole link is established, the adversary captures wireless transmissions on one end, sends them through the wormhole link and replays them at the other end.

d) **Transport and Application Layer Attacks:** WSNs are vulnerable against different transport and application layers attacks. Attacks on these layers are as follows:

- **De-synchronization Attack:** In this attack, established connections between legitimate nodes are disrupted by re-synchronizing their transmission. It alters the sequence numbers of packets and disrupts the communication protocol.

- **Data Aggregation Distortion:** It is an attack against data integrity. It is disrupting data aggregation, modifying collected data and distorting the final aggregation results.

- **Hello Flood Attack:** Many routing protocols use "HELLO" packet to discover neighboring nodes and thus to establish a topology of the network. The simplest attack for an attacker is sending such messages to flood the network and to prevent other messages from being exchanged.

- **Skew Clock Attack:** This attack is an attack on authenticity, integrity and availability. It involves disseminating false timing information to desynchronize the nodes.

***2.4 Moirangthem Marjit Singh, Ankita Singh and Jyotsna Kumar Mandal,* Towards Techniques of Detecting Node Replication Attack in Static Wireless Sensor Networks, Volume 4, Number 2(2014), 12 pages**

The paper reports the techniques to detect node replication also called clone node attacks in the Static Wireless Sensor Networks (WSNs) and critically reviews them. The focus of the paper is to highlight and discuss various techniques to deal with node replication attack in static WSNs. These techniques are given below-

a) Node-To-Node Broadcasting: In this method, each node stores the location information for its neighbors and broadcasts it to the entire network. It incurs a storage cost of O(d). If it receives a conflicting claim, it revokes procedures against the offending node. This method achieves 100% detection of all clones under the assumption that the broadcast is received by all the nodes. [12]

b) Deterministic Multicast: This method improves the communication problem suffered by the previous method. When location claim is broadcast by a node, its neighbors forward that claim only to a limited subset of deterministically chosen nodes called "witnesses", chosen as a function of the node's ID. If a clone is present, then conflicting locations are received by witnesses for the same node ID. There is a problem with this method that if an adversary knows the node's ID and function that determines the witnesses, then he can easily attack without detection. [13, 14]

c) Randomized Multicast: This method suggests a novel protocol that randomly selects witness nodes for a given sensor node's location claim within its communication range so that adversary cannot predict the witness' locations and forwards the location claim with a probability to the nodes closest to chosen locations by using geographic routing. In a network of n nodes, if each location produces $\sqrt{n}$ witnesses, then the birthday paradox predicts at least one collision with high probability, i.e., at least one witness will receive a pair of conflicting location claims, hence detecting the clone attack. This method of clone detection has a high communication cost. [13, 14]

d) Line Selected Multicast: This method uses the routing topology of the network to detect replication. Every node that forwards the claim also saves the claim, i.e. the intermediate nodes within the multicast path are also the witness nodes. Therefore, they can also revoke the replicated nodes in case of any conflict of location claims. This scheme gives a higher detection rate and lesser communication as compared to the randomized multicast scheme but it requires a relatively high cost of storage. [1, 15]

e) Randomized, Efficient and Distributed (RED): This method chooses witnesses pseudo-randomly based on a network wide seed. A random value, rand, is broadcast among all the nodes in the network. This can be done using centralized mechanism. Each node digitally signs and broadcast its claim ID and geographic location to its neighbors. Each of the neighbors selects a witness node using the node ID and random value to forward the claim. Since the random value changes every time, so it is impossible for the attacker to determine the witness node in progress. Location claims with the same node ID will be forwarded to the same witness nodes in each detection phase. Therefore, clone nodes are identified in each detection phase. [1, 16] However this method doesn't specify how to generate seed or random value. Also, it does not take into consideration the existence of compromised nodes. Therefore, we cannot use it to create a global leader of a sensor network composed of a large number of nodes with some of them compromised. [15]

f) Random-Walk Based Detection: The random walk strategy outperforms previous strategies because it distributes a core step, the witness selection, to every passed node of random walks, and then the adversary cannot easily find out the critical witness nodes. The two protocols based on random-walk are Random Walk (RAWL) and Table-assisted Random Walk (TRAWL). The Random Walk (RAWL) starts several random walks randomly in the network for each node, and then passed nodes are selected as the witness nodes. If any witness node receives different location claims for same node ID, clone is detected. The TRAWL is similar to RAWL, only a trace table is added at each node. In this instead of storing location claim definitely, it is stored independently with probability c2 √n log n, where c 2 is a constant. These methods have comparably high communication overhead. [3, 17]

g) Extremely Efficient Detection (XED): This scheme is for mobile WSN. Every sensor node is having a random number generator. When a node encounters another node, they exchange the random numbers and record them for further verification. When they meet again, they check the previously exchanged random numbers. If the numbers match, they update the random numbers; otherwise they determine their counterparts as clones. This approach requires only constant communication cost per detection and also the location information of sensor nodes is not required but it leaves the network vulnerable to smart attackers. [1, 8]

2.5 *Dan Gusfield, Robert W. Irving,* The Stable Marriage Problem – Structure and Algorithms.

This paper proposes the Stable Marriage Problem, well known problem for the determination of stable matching from the instance of equally sized sets of elements, given a preference order for each element. It was defined by David Gale and Lloyd Shapley in 1962.

The SMP of size *n* involves two disjoint sets, each of size *n*, namely the men and the women. A preference list is associated with each of the element of a set over the elements of the other set. This means each person has *a strictly ordered preference list* containing all the elements/members of the opposite sex. The corresponding output has to be stable, which implies that the married couple (or the matched pair) is satisfied and does not find a better choice of finding the other partner. Therefore the basic criterion of SMP is *stability*.

If there exists a man and a woman who prefer each other over their current matched pair, in the matching *M*, they are called as *rogue couple/blocking pair* and are said to *block* the matching *M.*

To prove this theorem, a basic version of the algorithm is discussed. This algorithm always finds a stable matching which turns out to be uniquely favorable to men and the women depending on the respective roles of the two sexes in the algorithm that is this algorithm is *asymmetrical* in the roles of the men and the women. There are two possible orientations, depending on who makes the proposals, namely the *man-oriented algorithm* and the *woman-oriented algorithm.*

Considering *man-oriented algorithm*, the idea is to iterate through all free men while there is any free man available. At any point of the algorithm's execution, each person is either engaged or free; each man may alternate between being free or engaged, but once a woman is engaged, she is never set free, although the identity of her fiancé may change. Every free man goes to all the women in his preference list one by one according to the order. For every woman he goes, he checks if the woman is free, if yes, they both get engaged. If the woman is not free, the woman chooses either to say no or dumps her current fiancé and chooses him.

For better understanding, general approach to the Gale-Shapely (man-oriented) algorithm can be summarized as-

- **Day 0:** each man ranks the women
- **Day 1: (in the morning)** each man proposes his top choice
  (in the evening) each woman rejects all but her top choice
- **Day n+1: (in the morning)** each rejected man proposes his top remaining choices
  (in the evening) each woman rejects all but her top choice

- When each man is engaged the algorithm terminates.

Considering above approach to the algorithm, the man-oriented version of the extended Gale-Shapely algorithm is described as-

```
assign each person to be free
while some man $m$ is free do
begin
    $w$=first woman on $m$'s list to whom $m$ has not yet proposed
    if $w$ is free then
        assign $w$ and $m$ as engaged to each other
    else
        if $w$ prefers $m$ to her fiancé $m'$ then
            assign $w$ and $m$ as engaged and set $m'$ to be free
        else
            $w$ rejects $m$ and $m$ remains free
end
output the stable matching consisting of $n$ engaged pairs
```

Figure 4: SMP algorithm [19]

The man-oriented and woman-oriented algorithms return man-optimal and woman-optimal stable matching respectively. The man optimal stable matching consist every man with his best possible choice of woman. However, while each man has his best fiancé, each woman might get the worst fiancé in any stable matching. Similarly woman-oriented algorithm has the same problem.

Further the paper proposes various theorems and examples to proof the problem stated.

2.6 *Hosam AlHakai, Feng Chen, Helge Janicke* , An Extended Stable Marriage Problem Algorithms For Clone Detection, Volume 5, Number 4, July 2014, 20 pages

This paper presents a novel approach to detecting cloned software by using a bijective matching technique. The proposed approach focuses on increasing the range of similarity measures and thus enhancing the precision of the detection. This is achieved by extending a well-known stable-marriage problem (SMP) and demonstrating how matches between code fragments of different files can be expressed. The proposed approach has improved clone detection in terms of scalability and accuracy.

SMP has solved several similar optimization issues in different fields such as matching jobs to the most suitable jobseekers. Since the original SMP algorithm allows only the candidates of the first set (Men) to propose to their first choices, the extended SMP by allows the candidates of the second set (Women) to make their own choices. The extended approach considers a dual multi allocation technique that allows the candidates of both first and second set to enter the competition and propose again for a certain times to their preferences. This adaption has enhanced the precision of the matching process. In the main SMP algorithm the assigned candidates are not necessarily similar to each other. However, in clone detection the concept of similarity is essential.

To apply the SMP algorithm in clone detection, it needs first to build the preference lists of both code fragments. This can be achieved using predefined metrics to specify the most similar related participants (code clone). Each code portion needs to strictly order the code fragments based on the similarity and vice versa. The traditional SMP algorithm performs a single assignment (one-to-one) for the involved candidates, which does not help especially in the case of allocating more than one code portion (method etc.) to the related code fragments of other source file. Multi Dual Allocation algorithm fulfils this requirement which widely needed in such fields. The preference list is built using metrics with parameters like Lines of code, number of variables, number of functions etc.

The extended SMP algorithm on clone detection can be applied with two main phases.

- **Phase1**, building the preference list of each code fragment of the first source file from the second source file's code portions, recording the most desired block and so on, repeating this process from second to first source files.

- **Phase2**, applying the adapted SMP algorithm based on the given metrics values. The main features of SMP-based approach for clone code detection competitively outweigh some already existing approaches in several facets such as increasing the spot of the detection process trying to detect every possible clone smell. However, this approach lacks two related features respectively time complexity and speed as the used algorithm is executed in a quadratic time, which impacts the overall scalability. The clone granularity has been set as a method-based blocks in which every method acts as a candidate (possible software clone).

CHAPTER-3
SYSTEM DEVELOPMENT

3.1 REQUIREMENTS

3.1.1 Functional Requirements

a) Deployment of nodes

- The system shall randomly deploy nodes in the given region.
- The system shall randomly generate a cryptographic key for the node.

b) Clustering

- The system shall divide the given region into clusters of sensor nodes
- The clustering shall be done in grid format.

c) Deploy Clones

- The system shall deploy clone nodes in the given network.

d) Clone Detection

- The system shall detect clone node in the given network.
- The system shall remove the clones from the network.

3.1.2 Non-Functional Requirements

a) Security

- The system must guarantee the security of the network information.

b) Concurrency

- The system should be able to handle multiple operations concurrently.

c) Availability

- The system must be available 24/7 for the network.

d) Compatibility

- The system must be easy to use and compatible with user system.

3.1.3 System Requirements

I have implemented our project on IDE Eclipse Kepler. Eclipse runs on today's most popular operating systems, including Windows XP, Linux, and Mac OS X. It requires Java to run, so if you don't already have Java installed on your machine, you must first install a recent version. Mac OS X has Java preinstalled. See the following table for the minimum and recommended system requirements.

Requirement	**Minimum**	**Recommended**
Java version	1.4.0	5.0 or greater
Memory	512 MB	1 GB or more
Free Disk Space	300 MB	1 GB or more
Processor Speed	800 MHz	1.5 GHz or faster

Table 2: System Requirements

3.2 PROCESS MODEL

For this project, I would suggest Spiral Model of software development process models which is an evolutionary software process model that couples the iterative nature of prototyping with controlled and systematic aspects of the waterfall model. A spiral model is divided into a number of framework activities also called task regions. A spiral model contains four task regions:

- **Determine Objectives**: This phase involves a good communication with the customer to well define the objectives or requirements for the software.
- **Risk Analysis**: This phase involves analyzing each aspect of software development in order to find every possible area that may involve risk while developing the software and thus designing modules accordingly.

- **Development:** This phase involves developing various diagrams and developing software according to the specification and thus testing the modules for their correct functioning.
- **Planning:** This phase involves planning for the next increment so that the increment may fit into already developed prototype and thus function according to the need.

3.3 DESIGN

3.3.1 Clone Node Detection using SMP

In WSN, each node can be uniquely identified using its ID, location, energy and cryptographic information. When a node is captured by an adversary, he can easily extract all this information and create any number of replicas using it. So, clones will have same parameters as above, may be with slight difference. Just like preference list of men and women was created, as described above, preference list for sensor nodes on the basis of similarity in the parameter values is created. Every node will have a preference list that includes every other sensor node in the network. Every node will have a preference list that includes every other sensor node in the network. So, nodes that are candidates for clones will have higher preference and genuine nodes will have least preference. After preference list is ready for every node, SMP algorithm can be applied to the network. Here, the separate sets of men m and women w are replaced by a single set of nodes n. The pairing will be done as many-to-one because single node can have multiple copies, so multiple nodes can map to a single node. A matching M in this case can be defined as: any two nodes N1 and N2 that are mutually acceptable to each other and will have almost identical parameters. A pair will form a blocking pair with respect to M if N1 is unmatched and finds N2 acceptable or N1 prefers N2 to the node it is assigned to. Therefore, if (N1, N2) forms a blocking pair in M, then N1 and N2 are likely to break their previously assigned arrangement. Thus, our goal is to find stable pairs and detect the clones present in the network. If no stable pair is formed that means our network does not have any clones in it.

Modified SMP Algorithm for node clone detection:

Following details are used for preparing the preference list-

Abbreviation used	Description
SID	Sensor node Id
LCN	Location of sensor node
ENG	Energy level of sensor node
CRP	Cryptographic key

Table 3: Parameters for preference list

Step1: Entire network is divided into clusters.

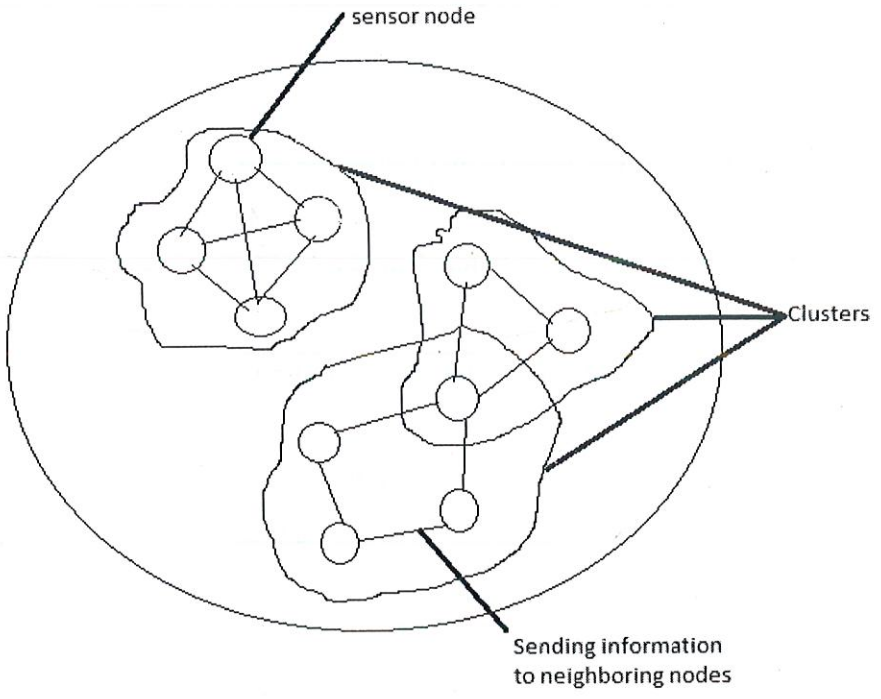


Figure 5: Wireless Sensor Network divided into clusters

Step2: In a cluster, every node will broadcast its information.

Step3: Every node will prepare a preference list using the information received from the broadcast. All those neighboring nodes whose parameter values are close to its values (or same) will be placed at higher priority. There can be multiple nodes with same priority.

Step4: SMP algorithm as given below is run within every cluster:

```
while preference list of some node N is not completely processed
do
        n=node with highest priority in N's preference list that it has not yet approached
        if n is free
        then
                assign n and N to each other as a matching
        else
                if n prefers N to its existing pair n'
                then
                        assign n and N to each other as a matching
                        set n' free
                else if n's current matching node(s) and N have same priority
                then
                        assign n and its matching node(s) to N
                end if
        end if
end while
```

At the end of this algorithm the stable matching pair we get will be clones within every cluster. Since clones tend to lie close to each other therefore they can be easily detected within a cluster. Once identified, these clone nodes can be removed from the network. The unstable pairs, i.e., pairs with non-similar parameter values or those that are paired with their last preference are considered genuine nodes.

Step5: Eliminate the clones.

3.3.2 Flow Chart

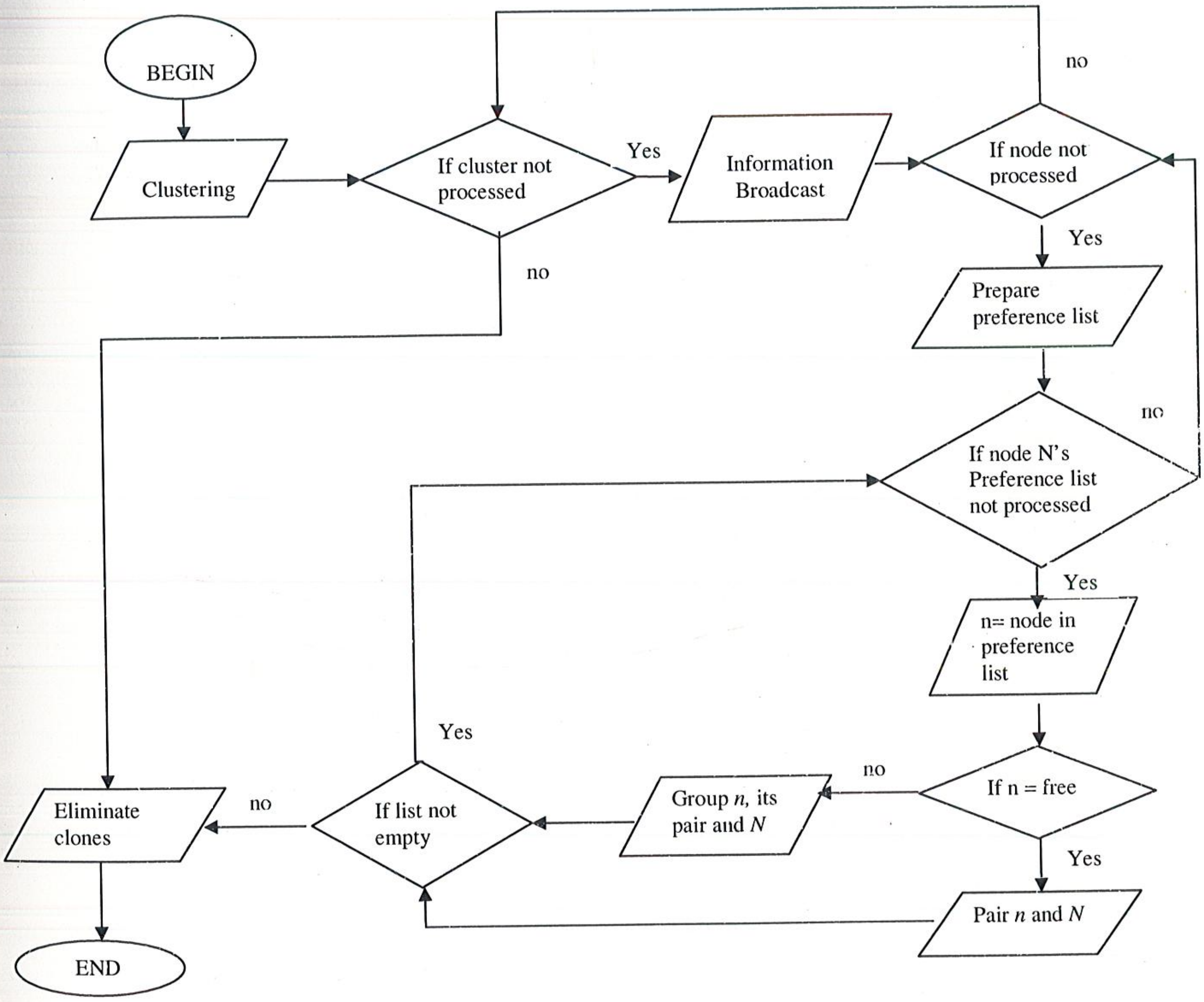


Figure 6: Flow Chart

3.3.3 Class Diagram

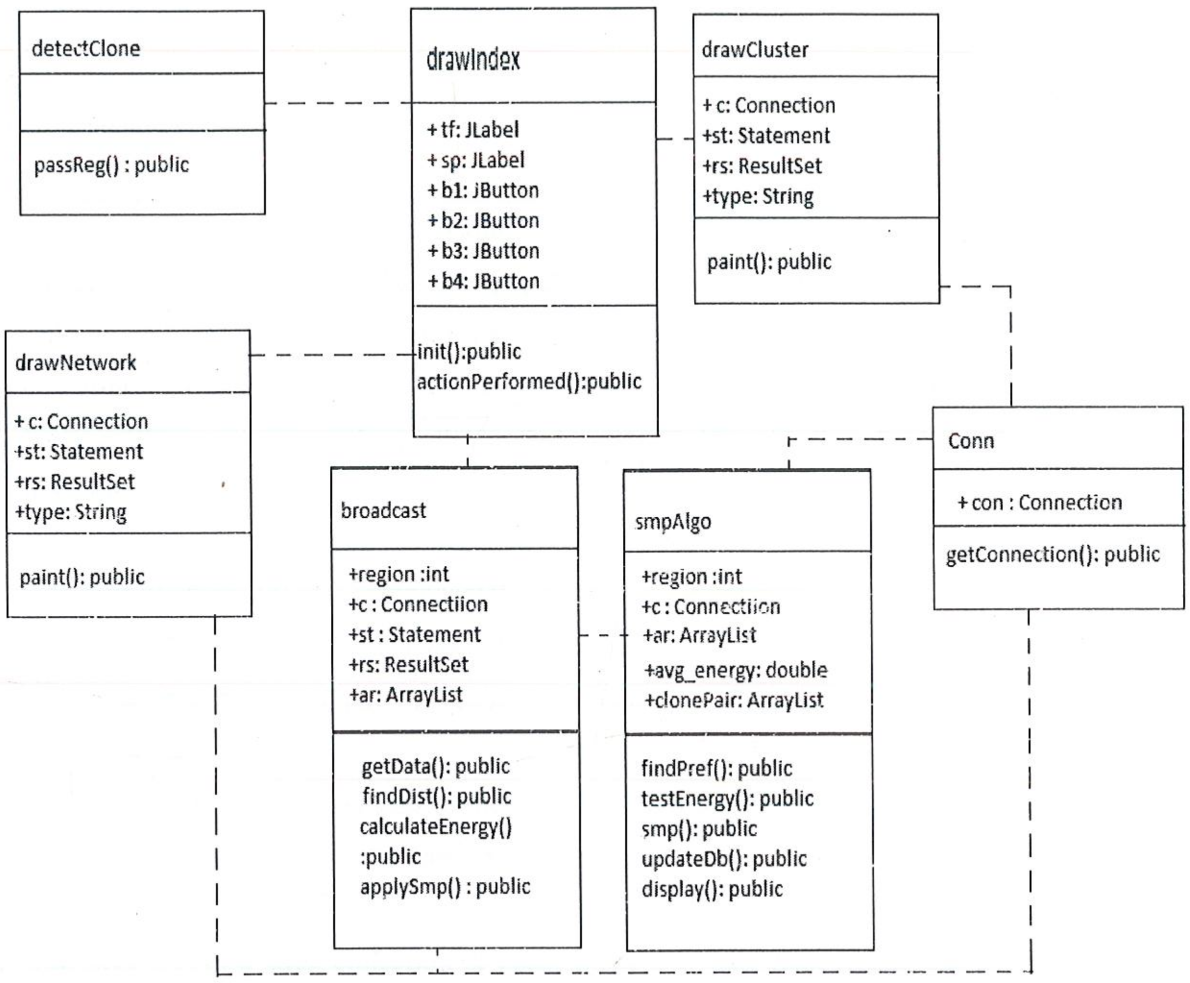


Figure 7: Class Diagram

3.4 IMPLEMENTATION

For implementation I have selected Java as our platform and I have implemented this using Eclipse *Kepler*. I have implemented this project using *Swings* because of various GUI features available in it. The code is given in Appendix.

3.4.1 Platform

A *platform* is the hardware or software environment in which a program runs. I've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do?, highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.

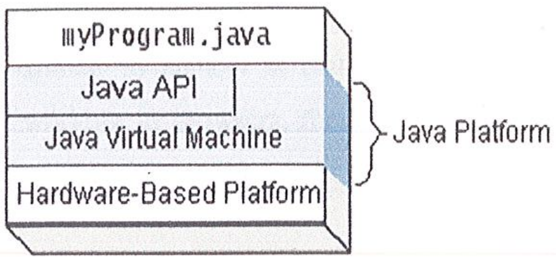


Figure 8: Java Platform

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

3.4.2 Features of Java

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provide a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.

- **Applets**: The set of conventions used by applets.

- **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.

- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.

- **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components**: Known as JavaBeansTM, can plug into existing component architectures.

- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).

- **Java Database Connectivity (JDBCTM)**: Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.

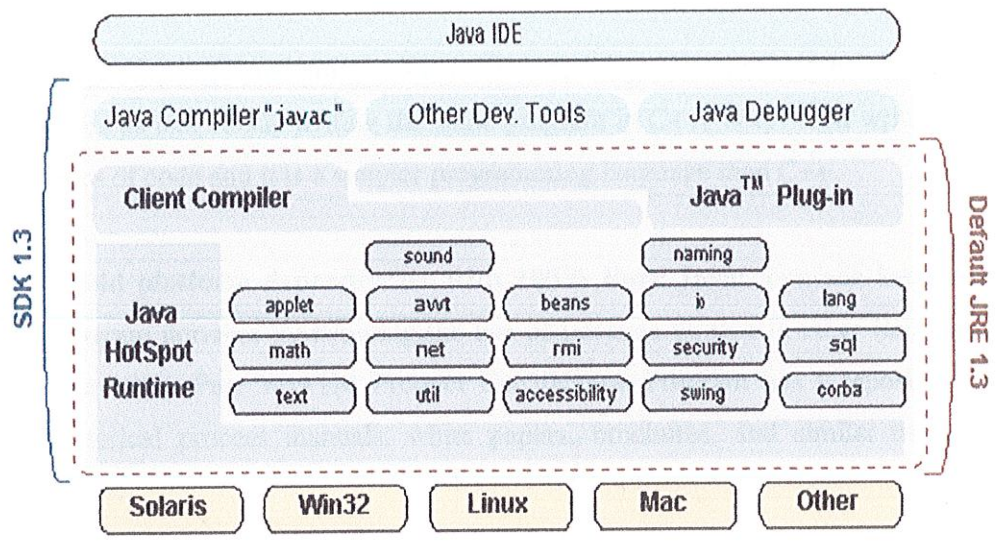


Figure 9: Java Architecture

3.4.3 Advantages of Java Technology

Java Technology makes your programs better and requires less effort than other languages. I believe that Java technology will help you do the following:

- **Get started quickly**: Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.

- **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.

- **Write better code**: The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.

- **Develop programs more quickly**: Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.

- **Avoid platform dependencies with 100% Pure Java**: You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure JavaTM Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.

- **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine-independent bytecodes, they run consistently on any Java platform.

- **Distribute software more easily**: You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded "on the fly," without recompiling the entire program.

3.4.4 JavaBeans

The JavaBeans API makes it possible to write component software in the Java programming language. Components are self-contained, reusable software units that can be visually composed into composite components, applets, applications, and servlets using visual application builder tools. JavaBean components are known as Beans.

Components expose their features (for example, public methods and events) to builder tools for visual manipulation. A Bean's features are exposed because feature names adhere to specific design patterns. A "JavaBeans-enabled" builder tool can then examine the Bean's patterns, discern its features, and expose those features for visual manipulation. A builder tool maintains Beans in a palette or toolbox. You can select a Bean from the toolbox, drop it into a form, modify it's appearance and behavior, define its interaction with other Beans, and compose it and other Beans into an applet, application, or new Bean. All this can be done without writing a line of code.

The following list briefly describes key Bean concepts,

- Builder tools discover a Bean's features (that is, its properties, methods, and events) by a process known as *introspection.* Beans support introspection in two ways:

    - By adhering to specific rules, known as *design patterns*, when naming Bean features.

    - By explicitly providing property, method, and event information with a related *Bean Information* class.

- *Properties* are a Bean's appearance and behavior characteristics that can be changed at design time. Builder tools introspect on a Bean to discover its properties, and expose those properties for manipulation.

- Beans expose properties so they can be *customized* at design time. Customization is supported in two ways: By using property editors, or by using more sophisticated Bean customizers.

- Beans use *events* to communicate with other Beans. A Bean that wants to receive events (a listener Bean) registers its interest with the Bean that fires the event (a source Bean). Builder tools can examine a Bean and determine which events that Bean can fire (send) and which it can handle (receive).

- *Persistence* enables Beans to save and restore their state. Once you've changed a Beans properties, you can save the state of the Bean and restore that Bean at a later time, property changes intact. JavaBeans uses Java Object Serialization to support persistence.

- A Bean's *methods* are no different than Java methods, and can be called from other Beans or a scripting environment. By default all public methods are exported.

- Although Beans are designed to be understood by builder tools, all key APIs, including support for events, properties, and persistence, have been designed to be easily read and understood by human programmers as well.

3.4.4.1 New JavaBeans Features

Here is a list of upcoming Beans and Beans-related features:

- The **Java Activation Framework (JAF)**. The JAF is a data typing and command registry API. With the JAF you can discover an arbitrary data object's type. and look command applications or Beans that can process that data type, and activate the appropriate command at a user gesture. For example, a browser can identify a file's data type, and then launch the appropriate plug-in to view or edit the file. The JAF is a Java standard extension.

- The **Extensible Runtime Containment and Services Protocol**, also known as beancontext. Previous to this protocol a Bean only knew about and had access to the Java Virtual Machine (JVM) in which the Bean ran, and the core Java APIs. This protocol introduces a standard way for a Bean to discover and access attributes or services provided by a Bean's enclosing environment, and for a Bean's enclosing environment to discover and access a Bean's attributes and services. The java.beans.beancontext API introduces the ability to nest Beans and Bean contexts within a hierarchical structure. At runtime, a Bean can obtain services from its containing enviroment; the Bean can then make use of those services, or propagate those services to any Beans that it contains.

- **Drag and Drop Support**. The java.awt.dnd API provides support for drag and drop between Java applications and native platform applications

3.4.5 Java Swings

Swing is the principal GUI toolkit for the Java programming language. It is a part of the JFC (Java Foundation Classes), which is an API for providing a graphical user interface for Java programs. It is completely written in Java.

Swing library is an official Java GUI toolkit released by Sun Microsystems. It is used to create Graphical user interfaces with Java.

The main characteristics of the Swing toolkit:

- platform independent
- customizable
- extensible
- configurable
- lightweight

The Swing API has 18 public packages:

- javax.accessibility
- javax.swing
- javax.swing.border
- javax.swing.colorchooser
- javax.swing.event
- javax.swing.filechooser
- javax.swing.plaf
- javax.swing.plaf.basic
- javax.swing.plaf.metal
- javax.swing.plaf.multi
- javax.swing.plaf.synth
- javax.swing.table
- javax.swing.text
- javax.swing.text.html
- javax.swing.text.html.parser
- javax.swing.text.rtf
- javax.swing.tree
- javax.swing.undo

Swing is an advanced GUI toolkit. It has a rich set of widgets. From basic widgets like buttons, labels, scrollbars to advanced widgets like trees and tables. Swing itself is written in Java.

Swing is a part of JFC, Java Foundation Classes. It is a collection of packages for creating full featured desktop applications. JFC consists of AWT, Swing, Accessibility, Java 2D, and Drag and Drop. Swing was released in 1997 with JDK 1.2. It is a mature toolkit.

The Java platform has Java2D library, which enables developers to create advanced 2D graphics and imaging.

There are basically two types of widget toolkits.

- Lightweight
- Heavyweight

A heavyweight toolkit uses OS's API to draw the widgets. For example Borland's VCL is a heavyweight toolkit. It depends on WIN32 API, the built in Windows application programming interface. On Unix systems, I have GTK+ toolkit, which is built on top of X11 library. Swing is a lightweight toolkit. It paints its own widgets. Similarly does the Qt4 toolkit.

SWT library

There is also another GUI library for the Java programming language. It is called SWT. The Standard widget toolkit. The SWT library was initially developed by the IBM corporation. Now it is an open source project maintained by the Eclipse community. The SWT is an example of a heavyweight toolkit. It lets the underlying OS to create GUI. SWT uses the Java native interface to do the job.

3.4.6 Databases and the Web

Databases are at the root of all business computing today. At some point, you are going to want to integrate a company database with your Web site. On the other hand, perhaps when you have a large Web site, you will want to create a database that will let clients search the text of your HTML files.

There are several ways to achieve database and Web site integration. Lets concentrate on JDBC.

3.4.6.1 JDBC

In an effort to set an independent database standard API for Java, Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or drivers. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.
To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.
JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.
The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

- **SQLLevelAPI**

  The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to "generate" JDBC code and to hide many of JDBC's complexities from the end user.

- **SQLConformance**

  SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

- **JDBC must be implemental on top of common database interfaces**

  The JDBC SQL API must "sit" on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.

- **Provide a Java interface that is consistent with the rest of the Java system**

  Because of Java's acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

- **Keep it simple**

  This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

- **Use strong, static typing wherever possible**

  Strong typing allows for more error checking to be done at compile time; also, less errors appear at runtime.

- **Keep the common cases simple**

  Because more often than not, the usual SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

CHAPTER-4
PERFORMANCE ANALYSIS

4.1 ASSUMPTIONS

Following are the assumptions based on which the proposed algorithm has been implemented:

- The network is static.
- The deployment of the nodes is random.
- AZR-LEACH algorithm has been used for clustering the nodes in the network.
- Number of clusters formed in each test case is 9.
- The clone nodes have been randomly chosen within a region.
- After detection, mitigation of compromised nodes is handled automatically.
- The energy of the clone nodes is higher than the average energy of legitimate nodes.
- The algorithm runs over area of 600 X 600 units dimension.

4.2 ANALYSIS

In this sub-section, I will compare various existing node clone detection schemes among themselves and also with our proposed scheme. [1, 15]

Method	**Sensor Type**	**Approach Used**	**Communication cost**	**Memory Cost**	**Detection Level**
SET	Static	Centralized	O(n)	O(d)	High
Random Key Predistribution	Static	Centralized	O(nlogn)	O(d)	High
Real time clone detection using fingerprints	Static	Centralized	O(numm √n).log2M	O(d)+min(M, ϙlog2 M)	High

Fast and Effective mobile replica node detection	Mobile	Centralized	$O(n\sqrt{n})$	$O(n)$	High
Node-to-Network Broadcasting	Static	Distributed	$O(n^2)$	$O(d)$	High under ideal conditions
Deterministic Multicast	Static	Distributed	$O(g \ln g\sqrt{n} / d)$	$O(g)$	Low
Randomized Multicast	Static	Distributed	$O(n^2)$	$O(\sqrt{n})$	Medium
Line selected multicast	Static	Distributed	$O(n\sqrt{n})$	$O(\sqrt{n})$	Medium
RED	Static	Distributed	$O(\sqrt{n})$	$O(d\sqrt{n})$	High
Active detection	Static	Distributed	$O(\sqrt{n})$	$O(1)$	Low
Random-walk based detection	Static	Distributed	RAWL: $O(\sqrt{n} \log n)$ TRAWL: $O(\sqrt{n} \log n)$	RAWL: $O(\sqrt{n} \log n)$ TRAWL: $O(1)$	High
XED	Mobile	Distributed	$O(1)$	No storage required	Medium

Table 4: Comparative Analysis [3,5,8,11]

Notations:

1. n- number of nodes in the network
2. d- degree of neighboring nodes
3. g- number of witness nodes
4. m- group size
5. numm- total number of rows in the superimposed s-disjunct code

6. M- number of rows in the superimposed s-disjunct code
7. ω- column weight in the superimposed s-disjunct code

From the above table it can be concluded that centralized procedures reduces the complexity of clone detection as compared to the distributed techniques. The worst problem in the centralized scheme is its complete dependency on base station. If the base station or its neighbors are compromised then the entire network security is threatened. Also, when deterministic schemes are compared to non-deterministic schemes, the latter is more efficient and reliable as it makes difficult for the attacker to identify witness nodes. So, it can be concluded that a clone detection technique should be non-deterministic and full distributed (NDFD).

The SMP algorithm is fully distributed. Every node participates in the detection procedure. Also, this scheme does not require any witness nodes. Therefore, this algorithm is NDFD and provides us with an optimum solution for static networks.

In SMP, every node is compared with every node in its preference list to check for a potential match. Therefore, it gives us a solution in O(nm) where n is the number of nodes in the network. Since every node maintains a preference list that includes every other node in the network, so memory required by the algorithm is O(nm). Memory cost is high as compared to other schemes. This cost can be reduced to some extent, if all those nodes whose parameters do not match with that of current nodes even slightly are eliminated from the preference list. Now every node need not check (m-1) nodes for a match. This will also reduce the amortized cost of the algorithm.

SMP uses brute force method to check for every potential match and every stable match is a clone. Therefore, this algorithm ensures high detection of clones in the network. Although other techniques offer better complexity but SMP is efficient and need not run frequently and is performed within clusters so high complexity can be overlooked considering its efficiency.

4.3 TESTING

Following are the test cases that have been run on the proposed algorithm:

4.3.1 Test Case 1:

- Total number of nodes: 90
- Number of nodes per cluster: 10
- Number of clones per cluster: 0
- Test Condition: none

Output:

- Program execution: successful
- Number of clones detected: 0
- Average execution time: 16 ms

4.3.2 Test Case 2:

- Total number of nodes: 90
- Number of nodes per cluster: 10
- Number of clones per cluster: 1
- Test Condition: none

Output:

- Program execution: successful
- Number of clones detected: 18
- Average execution time: 15 ms

4.3.3 Test Case 3:

- Total number of nodes: 90
- Number of nodes per cluster: 10
- Number of clones per cluster: 2
- Test Condition: none

Output:

- Program execution: successful
- Number of clones detected: 27
- Average execution time: 15.5 ms

4.3.4 Test Case 4:

- Total number of nodes: 90
- Number of nodes per cluster: 10
- Number of clones per cluster: 3
- Test Condition: none

Output:

  - Program execution: successful
  - Number of clones detected: 36
  - Average execution time: 16.5 ms

4.3.5 Test Case 5:

- Total number of nodes: 90
- Number of nodes per cluster: 10
- Number of clones per cluster: 4
- Test Condition: none

Output:

  - Program execution: successful
  - Number of clones detected: 45
  - Average execution time: 16 ms

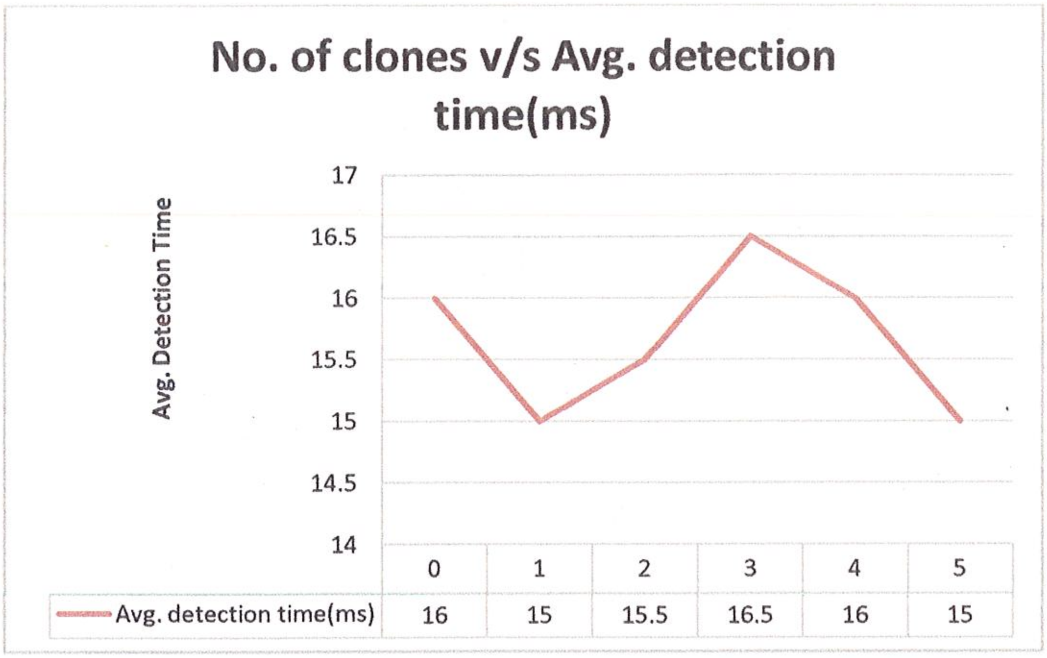
4.3.6 Test Case 6:

- Total number of nodes: 90
- Number of nodes per cluster: 10
- Number of clones per cluster: 5
- Test Condition: none

Output:

  - Program execution: successful
  - Number of clones detected: 36
  - Average execution time: 15 ms

Following is the graphical representation of the above test cases:



	0	1	2	3	4	5
Avg. detection time(ms)	16	15	15.5	16.5	16	15

Graph 1: No. of Clones v/s Avg. detection time

4.3.7 Test Case 7:

- Total number of nodes: 100
- Number of genuine nodes per cluster: 10
- Number of clones per cluster: random
- Test Condition: none

Output:

- Program execution: successful
- Number of clones detected: All clones **detected** successfully
- Average execution time: 26.88 ms

4.3.8 Test Case 8:

- Total number of nodes: 150

- Number of genuine nodes per cluster: 15
- Number of clones per cluster: random
- Test Condition: none

Output:

  - Program execution: successful
  - Number of clones detected: All clones **detected** successfully
  - Average execution time: 37.67 ms

4.3.9 Test Case 9:

- Total number of nodes: 200
- Number of genuine nodes per cluster: 15
- Number of clones per cluster: random
- Test Condition: none

Output:

  - Program execution: successful
  - Number of clones detected: All clones **detected** successfully
  - Average execution time: 65.33 ms

4.3.10 Test Case 10:

- Total number of nodes: 300
- Number of genuine nodes per cluster: 30
- Number of clones per cluster: random
- Test Condition: none

Output:

  - Program execution: successful
  - Number of clones detected: All clones **detected** successfully
  - Average execution time: 89.88 ms

4.3.11 Test Case 11:

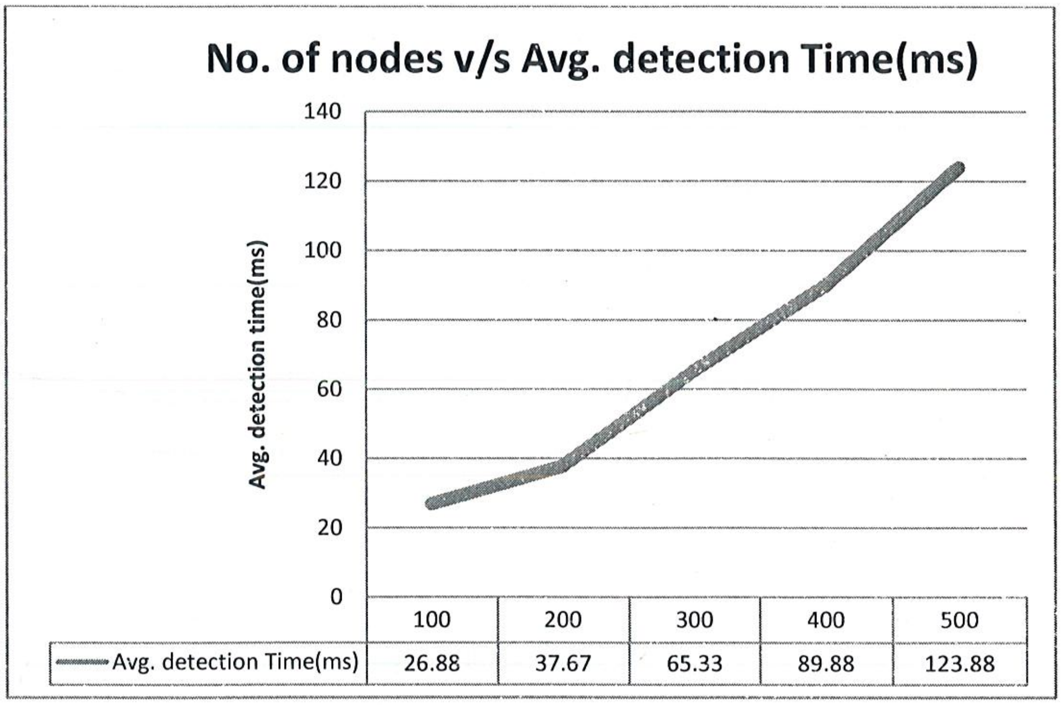
- Total number of nodes: 400

- Number of genuine nodes per cluster: 40
- Number of clones per cluster: random
- Test Condition: none

Output:

- Program execution: successful
- Number of clones detected: All clones **detected** successfully
- Average execution time: 123.88 ms

Following is the graphical representation for test cases 7-11 :-



	100	200	300	400	500
Avg. detection Time(ms)	26.88	37.67	65.33	89.88	123.88

Graph 2: No. of nodes v/s Avg. detection time

4.3.12 Test Case 12:

- Total number of nodes: 90
- Number of genuine nodes per cluster: 10
- Number of clones per cluster: random
- Test Condition: Some of the clones are present at the boundary of the clusters.

Output:

- Program execution: successful
- Number of clones detected: All clones **detected** successfully
- Average execution time: 26.77 ms

4.3.13 Test Case 13:

- Total number of nodes: 90
- Number of genuine nodes per cluster: 10
- Number of clones per cluster: random
- Test Condition: Some of the clones are present in another cluster than the one in which the compromised node is present.

Output:

- Program execution: successful
- Number of clones detected: Clones outside the cluster (containing the original node) are **not detected.**
- Average execution time: 27.14 ms

4.4 RESULTS

The algorithm successfully detected all the clones for the above mentioned test cases. The only exception is Test Case 13 in which the clone nodes are located outside the cluster containing legitimate node. But this is not a cause of concern, since all the clone nodes are generally located in the vicinity of the legitimate node.

In addition to this, our results can be verified using energy test in which the energy level of the detected clones is tested against the average energy level. The clones with energy higher than the average energy pass the test indicating they are the clones deployed by the adversary and the clones with energy less than the average energy fail this test indicating they are the legitimate nodes that have been compromised.

4.5 SNAPSHOTS

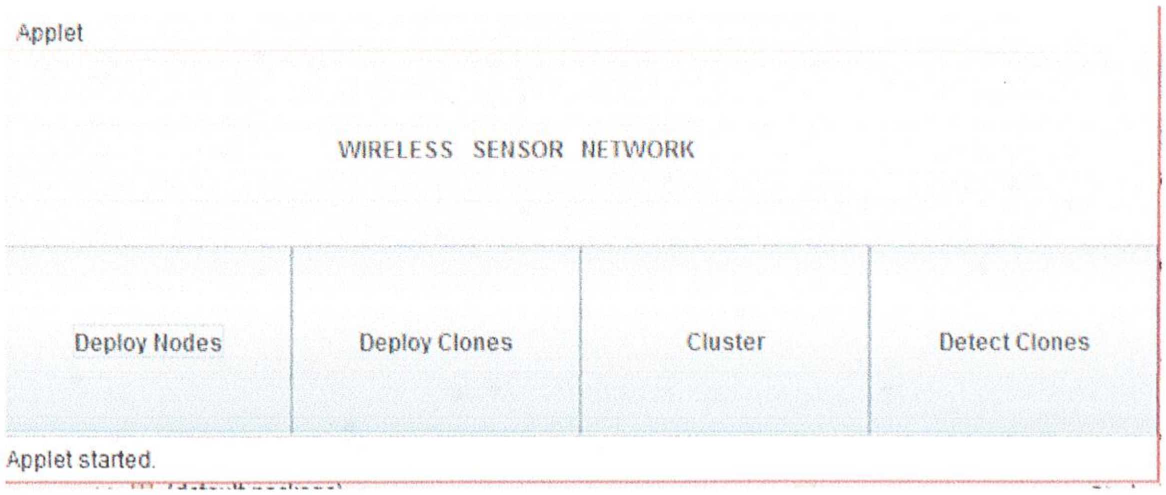


Figure 10 : Menu

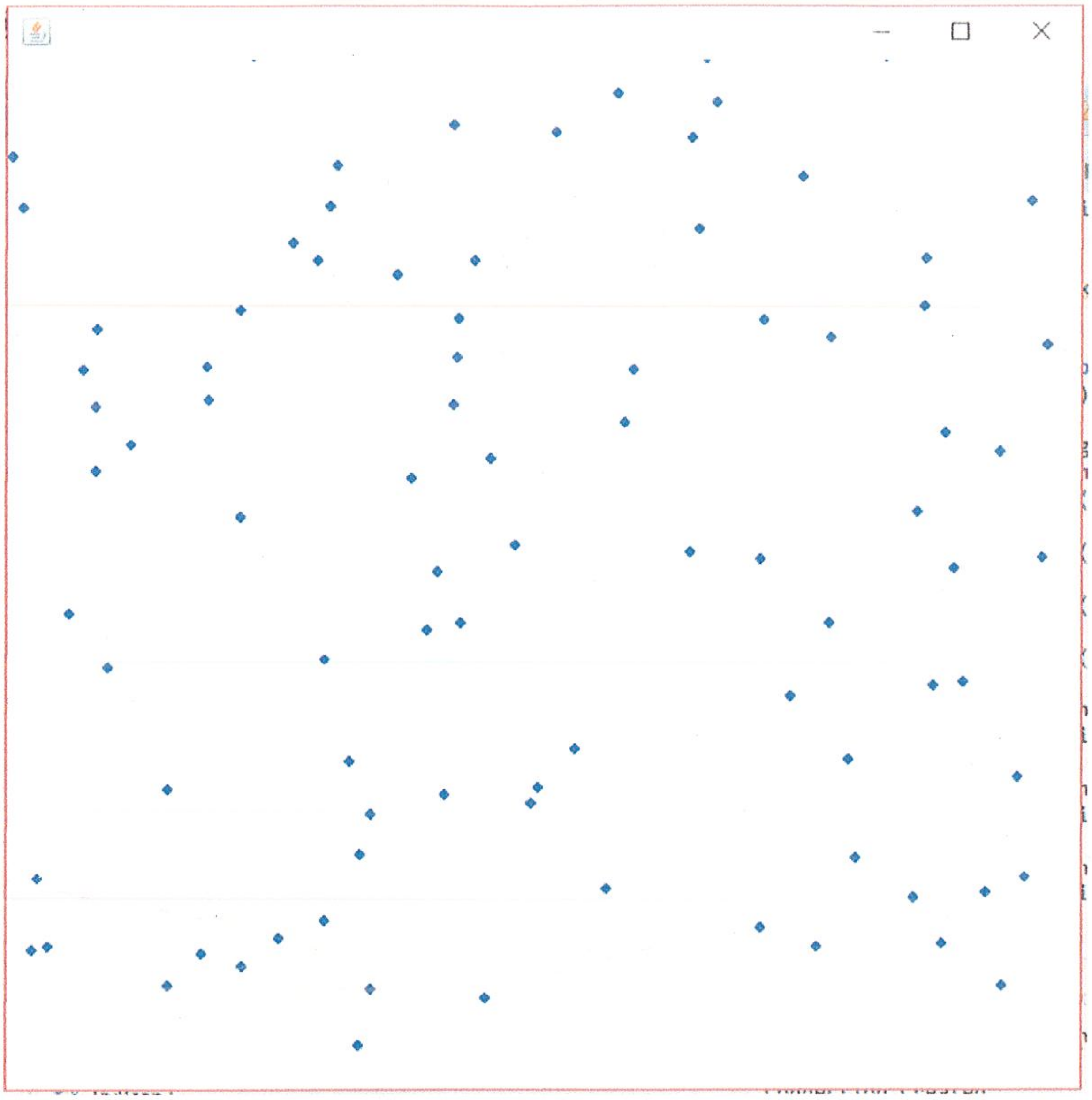


Figure 11 : Deployed nodes

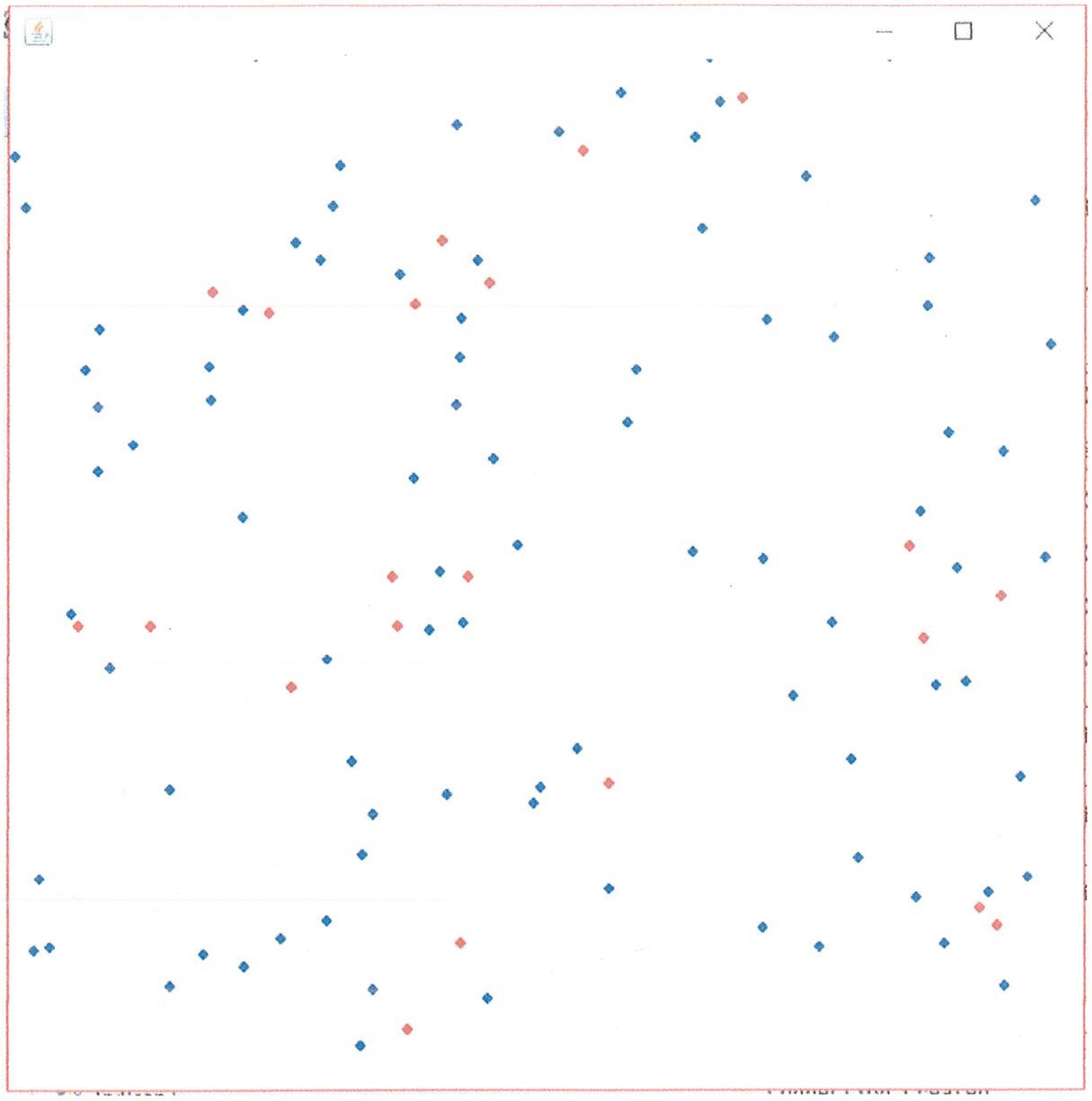


Figure 12 : Deployed Clones

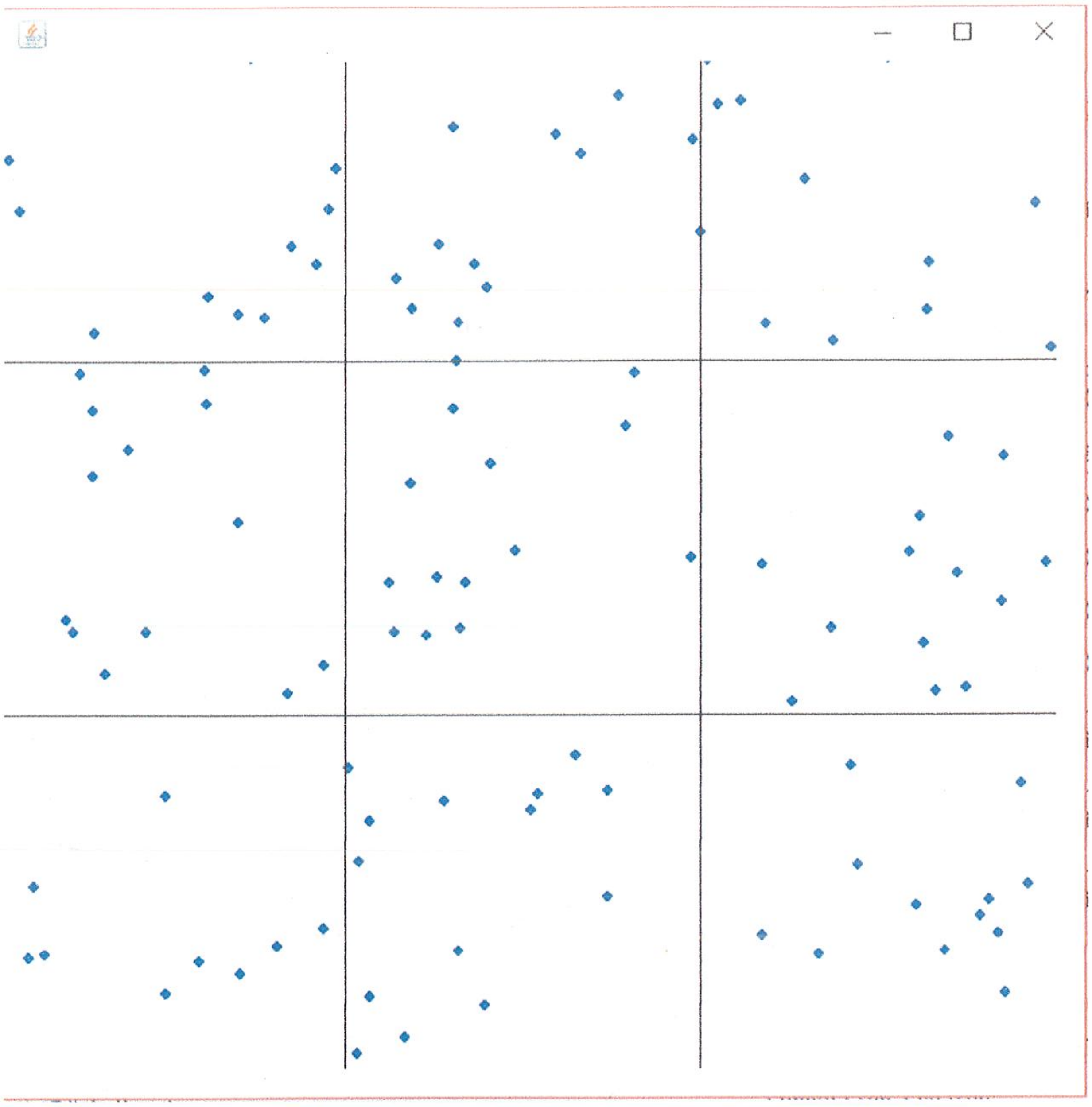


Figure 13: Clustering

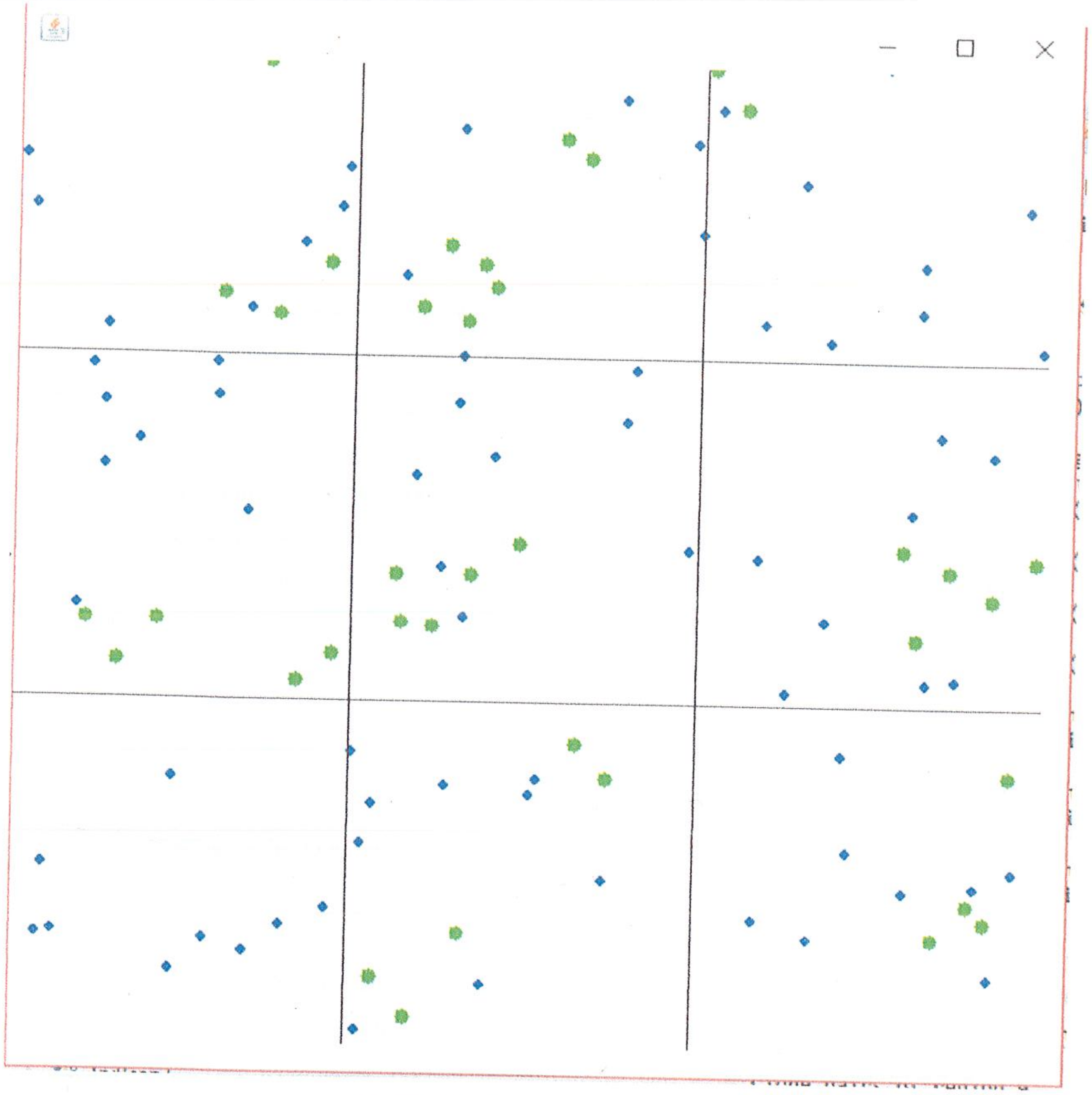


Figure 14: Clones Detected

CHAPTER-5
CONCLUSION

5.1 CONCLUSIONS

Security of WSNs is becoming a serious concern because of the increased number of real life applications of WSN in security critical areas like military and health care. Adversaries are always trying to infiltrate the sensor network with its clone nodes and disrupt the operations of the system. Many clone detection techniques have been proposed in the past both centralized and distributed schemes. In this paper, I have proposed a distributed scheme to detect clone nodes in a wireless sensor network using a variant of stable marriage problem algorithm. In this algorithm, every node participates equally in the detection process, hence avoiding the problem of single fault error. It takes into consideration every possible pair which can be clones, ensuring high detection rate within $O(m^2)$ time in a cluster. It has $O(m^2)$ memory requirement but this can be reduced with few modifications as discussed above. Memory requirement is directly proportional to number of clones in the network. Generally number of clones is very small so memory required is negligible. This algorithm is efficient and reliable and can help mitigate the problem of clone nodes to a great extent.

5.2 FUTURE SCOPE

The proposed algorithm for detection of clone nodes in static WSN can also be used for detection of clone nodes in mobile WSN. This can be implemented using any of the other existing clustering techniques. Further improvements can be done in order to reduce communication and memory costs.

REFERENCES

1. Vandana Mohindru, Yashwant Singh, "*Survey on Security Attacks In Wireless Sensor Networks: Node Clone Attack Perspective*".
2. John A. Stankovic, "*Wireless Sensor Networks*", in Computer, vol. 41 issue 10, University of Virginia, June 19, 2006, pp. 92-95.
3. Wazir Zada Khan, Mohammed Y. Aalsalem, Mohammed Nauful Bin Mohammed Saad, and Yaang Xing, "*Detection and Mitigation of Node Replication Attacks in Wireless Sensor Networks: A Survey*", International Journal of Distributed Sensor Networks, vol. 2013, article id 149023, 22 March 2013, pp. 22.
4. Teodore-Grigore Lupu, *"Main Types of Attacks In Wireless Sensor Networks*", Proceedings of the 9th WSEAS international conference on signal, speech and image processing, and 9th WSEAS international conference on Multimedia, internet & video technologies, Department of Computer and Software Engineering, University Politehnica of Timisoara, 3 March 2009, pp. 180-185 .
5. Neenu George, T.K. Parani, "*Detection of Node Clones in Wireless Sensor Network Using Detection Protocols*", International Journal of Engineering Trends and Technology, vol. 8, no. 6, 6 February, 2014, pp. 6.
6. Narasimha Shashidhar, Chadi Kari, and Rakesh Verma, "*Epidemic Node Replica Detection in Sensor Networks, Securing the Sensor Network Cyberspace against Replica Attacks*", Second ASE International Conference on Big Data Science and Computing, Stanford, CA, vol. 4, May 2014.
7. Tamara Bonaci, Phillip Lee, Linda Bushnell, Radha Poovendran, *"Distributed Clone Detection in Wireless Sensor Networks: An Optimization Approach"* (Invited paper), University of Washington, Seattle, WA, pp. 6.
8. Sagar Chandrahas Jagtap, Geetika Narang, "*Survey on Distributed Detection of Clone Attacks in Wireless Sensor Networks*", Proceedings of 5th SARC-IRF International Conference, New Delhi, India, 25 May 2014, pp. 13-18.
9. Kai Xing, Fang Liu, Xiuzhen Cheng, David H.C. Du, *"Real-time Detection of Clone Attacks in Wireless Sensor Networks",* 28th International Conference on Distributed Computing Systems, 2008, pp. 10.

10. Jun-Won Ho, Matthew Wright, Member, IEEE, and Sajal K. Das, Senior Member, IEEE, "*Fast Detection of Mobile Replica Node Attacks in Wireless Sensor Networks Using Sequential Hypothesis Testing*", IEEE Transactions on Mobile Computing, University of Texas, vol. 10, no. 6, June 2011.
11. Moirangthem Marjit Singh, Ankita Singh and Jyotsna Kumar Mandal, *"Towards Techniques of Detecting Node Replication Attack in Static Wireless Sensor Networks"*, International Journal of Information and Computation Technology, vol. 4, no. 2, 2014, pp. 153-164.
12. Mrs. Suvarna Game, Mr. Chandrashekhar Raut, *"Protocols for detection of node replication attack on wireless sensor network",* IOSR Journal of computer Engineering, vol. 16, issue 1, ver. 2, Jan. 2014, pp. 01-11.
13. P. Edith Linda, R.Sangeetha, *"An Approach To Detect Node Replication In Mobile Sensor Networks- Survey",* International Journal of Computer Technology and Applications, vol. 5(5), Sep-Oct 2014, pp. 1582-1587.
14. Bryan Parno, Adrian Perrig, Virgil Gligor, *"Distributed Detection of Node Replication Attacks in Sensor Networks*", SP '05 Proceedings of the 2005 IEEE Symposium on Security and Privacy, 2005, pp. 49-63.
15. V.Manjula and Dr.C.Chellappan, *"Replication Attack Mitigations for Static and Mobile WSN"*, International Journal of Network Security and its Applications (IJNSA), vol. 3, no. 2, march 2011, pp. 122-133.
16. C. Geetha, *"A Review on Replica Node Detection Algorithms in Wsns*", International Journal of Computational Engineering Research, vol. 3, issue 8, August 2013, pp. 84-89.
17. Yingpei Zeng, Jiannong Cao, Shigeng Zhang, Shanqing Guo, Li Xie, *"Random-walk based approach to detect clone attacks in wireless sensor networks",* IEEE Journal on Selected Areas in Communications, vol. 28, issue 5, June 2010, pp. 677-691.
18. Gale, David & Lloyd S. Shapely, *"College Admissions and the Stability of Marriage"*, The American Mathematical Monthly, vol. 69, no. 1, Jan. 1962, pp. 9-15.

19. Dan Gusfield, Robert W. Irving, *"The Stable Marriage Problem – Structure and Algorithms",* MIT Press Cambridge, MA, USA, 1989.
20. Hosam AlHakai, Feng Chen, Helge Janicke, *"An Extended Stable Marriage Problem Algorithms For Clone Detection"*, International Journal of Software Engineering and Applications (IJSEA), vol. 5, no. 4, Aug 2014, pp. 103-122.
21. Emily Riehl, *"A Solution To Stable Marriage Problem"*, Harvard University, 6 March 2013.

APPENDIX

SOURCE CODE

a) **Broadcast.java**

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import bean.node;
public class broadcast {

        int region;
        Connection c;
        Statement st;
        ResultSet rs;
        ArrayList<node> ar;
        double distance[][];

        broadcast(int region)
        {
                this.region=region;
                ar=new ArrayList<>();
                c=Conn.getConnection();
        }

        public void getData()
        {
                try {
                        st=c.createStatement();
                        rs=st.executeQuery("select * from clone where Region='"+region+"'");
                        while(rs.next())
                        {
                                node nd=new node();
                                nd.setNodeId(rs.getInt("NodeId"));
                                nd.setNodeKey(rs.getString("NodeKey"));
                                nd.setLocX(rs.getInt("LocX"));
                                nd.setLocY(rs.getInt("LocY"));
                                nd.setEnergy(rs.getDouble("Energy"));
                                nd.setClone(rs.getString("Clone"));
                                nd.setRegion(rs.getInt("Region"));
                                if(rs.getDouble("Energy")>0)
                                 ar.add(nd);
                                else
                                {
                                        st.executeUpdate("update clone set Clone='D' where
LocX='"+rs.getInt("LocX")+"' and LocY='"+rs.getInt("LocY")+"'");
                                }
                        }
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
        public void findDist()
        {
                int n=ar.size();
                distance=new double[n][n];
                int x1,x2,y1,y2;
                for(int i=0;i<n;i++)
                {
                        for(int j=i;j<n;j++)
                        {
                                if(i==j)
                                        distance[i][j]=0;
                                else
                                {
                                x1=ar.get(i).getLocX();
                                x2=ar.get(j).getLocX();
                                y1=ar.get(i).getLocY();
                                y2=ar.get(j).getLocY();
                                distance[i][j]=Math.sqrt(Math.pow((x1-x2),2)+Math.pow((y1-
y2),2));
```

```
                                        distance[j][i]=distance[i][j];
                                        }
                                }
                        }

                }
                public void calculateEnergy()
                {
                        double ETX=0.0000005; //transmission energy
                        double ERX=0.0000005; //reception energy
                        double Efs=0.00000000001; //amplification energy
                        double EDA=5*0.00000001; //data aggregation energy
                        double Ten,Ren,en;
                        int n=ar.size();
                        PreparedStatement ps=null;
                        try {
                         ps=c.prepareStatement("update clone set Energy=? where LocX=? and
LocY=?");
                        } catch (SQLException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        for(int i=0;i<n;i++)
                        {
                                for(int j=i;i<n;i++)
                                {
                                        Ten=( (ETX+EDA)*(4000)  + Efs*4000*( distance[i][j] *
distance[i][j] ));
                                        Ren=( (ERX)*(4000)  + Efs*4000*( distance[i][j] *
distance[i][j] ));
                                        en=Ten+Ren;
                                        node nd=ar.get(i);
                                        nd.setEnergy(nd.getEnergy()-en);
                                        ar.set(i,nd);
                                        nd=ar.get(j);
                                        nd.setEnergy(nd.getEnergy()-en);
                                        ar.set(j,nd);
                        }
                }
                for(int i=0;i<n;i++)
                {
                        int x,y;
                        x=ar.get(i).getLocX();
                        y=ar.get(i).getLocY();
                        double e=ar.get(i).getEnergy();
                        try {
                                ps.setDouble(1,e);
                                ps.setInt(2,x);
                                ps.setInt(3,y);
                                ps.executeUpdate();
                        } catch (SQLException e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }

                }

        }

        public void applySmp()
        {
                smpAlgo sa=new smpAlgo(region,ar);
          sa.findPref();
          sa.smp();
          sa.updateDb();
          sa.display();
        }

}
```

b) Conn.java

```
import java.sql.*;
public class Conn {
        static Connection con=null;
        static public Connection getConnection()
        {
                if(con==null)
                {
                        try {
                                Class.forName("com.mysql.jdbc.Driver");
                                System.out.println("Driver Loaded");

        con=DriverManager.getConnection("jdbc:mysql://localhost:3306/wsn","root","");
                                System.out.println("Connection Created");
                        } catch (ClassNotFoundException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (SQLException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
                return con;
        }
}
```

c) **detectClone.java**

```
import java.util.Date;
public class detectClone {

        public void passReg()
        {
                long st,et;
                for(int i=1;i<=9;i++)
                {
                        st=new Date().getTime();
                        broadcast obj=new broadcast(i);
                        obj.getData();
                        obj.findDist();
                        obj.calculateEnergy();
                        obj.applySmp();
                        et=new Date().getTime();
```

```
                                System.out.println("Elapsed Time for"+i+": "+(et-st));

                        }
                }
        }
```

d) drawCluster.java

```
        import java.awt.Color;
        import java.awt.Container;
        import java.awt.FlowLayout;
        import java.awt.Graphics;
        import java.sql.Connection;
        import java.sql.Statement;
        import java.sql.ResultSet;
        import java.sql.SQLException;
        import javax.swing.JApplet;
        import javax.swing.JFrame;
        import javax.swing.JPanel;


        public class drawCluster extends JFrame {
                Connection c;
                Statement st;
                ResultSet rs;
                String type=null;

                public drawCluster(String type)
                {

                        JPanel jp=new JPanel();
                        getContentPane().add(jp);
                        //Container cp=getContentPane();
                        setSize(625,625);
                        //cp.setLayout(new FlowLayout());
                        this.type=type;
                }
                public void paint(Graphics g)
                {
                    g.setColor(Color.BLUE);

                    try {
                            c=Conn.getConnection();
                            st=c.createStatement();
                            if(type.equals("gen"))
                            {
                            rs=st.executeQuery("select LocX,LocY from clone");
                            while(rs.next())
                            {
                                    int x,y;
                                    x=rs.getInt("LocX");
                                    y=rs.getInt("LocY");
                                    g.fillOval(x, y, 6, 6);

                            }
                            }
                            if(type.equals("clone"))
                            {
                                    g.setColor(Color.BLUE);
                                    rs=st.executeQuery("select LocX,LocY from clone where
            Clone='N'");

                                    while(rs.next())
                                    {
                                            int x,y;
                                            x=rs.getInt("LocX");
                                            y=rs.getInt("LocY");
                                            g.fillOval(x, y, 6, 6);

                                    }
                                            g.setColor(Color.GREEN);
                                            rs=st.executeQuery("select LocX,LocY from clone
            where Clone='Y'");
                                            while(rs.next())
                                            {
                                                    int x,y;
                                                    x=rs.getInt("LocX");
```

```
                                        y=rs.getInt("LocY");
                                        g.fillOval(x, y, 9, 9);
                        }
                    }

            } catch (SQLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            }
            g.setColor(Color.BLACK);
            for(int c=0;c<=400;c+=200)
            {

                            g.drawLine(0, c, 600, c);
            }
            for(int r=0;r<=400;r+=200)
            {

                            g.drawLine(r, 0, r, 600);
            }

        }

}
```

e) **drawIndex.java**

```
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;


public class drawIndex extends JApplet implements ActionListener{

        JLabel tf,sp;
        JButton b1,b2,b3,b4;
        public void init()
        {
                Container cp=getContentPane();
                cp.setLayout(new GridLayout(2,4));
                sp=new JLabel();
                cp.add(sp);
                tf=new JLabel("        WIRELESS   SENSOR");
                cp.add(tf);
                sp=new JLabel("NETWORK");
                cp.add(sp);
                sp=new JLabel();
                cp.add(sp);
                b1=new JButton("Deploy Nodes");
                b1.addActionListener(this);
                cp.add(b1);
                b2=new JButton("Deploy Clones");
                b2.addActionListener(this);
                cp.add(b2);
                b3=new JButton("Cluster");
                b3.addActionListener(this);
                cp.add(b3);
                b4=new JButton("Detect Clones");
                b4.addActionListener(this);
                cp.add(b4);
        }
        public void actionPerformed(ActionEvent ae)
        {
                if(ae.getSource()==b1)
                {

                        drawNetwork an=new drawNetwork("gen");
                        an.setVisible(true);

                }
                if(ae.getSource()==b2)
                {

                        drawNetwork an=new drawNetwork("clone");
                        an.setVisible(true);

                }
                if(ae.getSource()==b3)
                {

                        drawCluster ac=new drawCluster("gen");
                        ac.setVisible(true);

                }
                if(ae.getSource()==b4)
                {

                        detectClone dc=new detectClone();
                        dc.passReg();
                        drawCluster ac=new drawCluster("clone");
                        ac.setVisible(true);

                }
        }

}
```

f) **drawNetwork.java**

```
import java.awt.Color;
import java.awt.Graphics;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.JFrame;
import javax.swing.JPanel;


public class drawNetwork extends JFrame{
        Connection c;
        Statement st;
        ResultSet rs;
        String type;

        public drawNetwork(String type)
        {
                JPanel jp=new JPanel();
                getContentPane().add(jp);
                //Container cp=getContentPane();
                setSize(625,625);
                this.type=type;

        }
        public void paint(Graphics g)
        {


                try {
                        c=Conn.getConnection();
                        st=c.createStatement();

                                g.setColor(Color.BLUE);
                        rs=st.executeQuery("select LocX,LocY from clone where Clone='N'");
                        while(rs.next())
                        {
                                int x,y;
                                x=rs.getInt("LocX");
                                y=rs.getInt("LocY");
                                g.fillOval(x, y, 6, 6);
                        }
                        if(type.equals("clone"))
                        {
                                g.setColor(Color.RED);
                                rs=st.executeQuery("select LocX,LocY from clone where
Clone='Y'");
                                while(rs.next())
                                {
                                        int x,y;
                                        x=rs.getInt("LocX");
                                        y=rs.getInt("LocY");
                                        g.fillOval(x, y, 6, 6);

                                }
                        }
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
}
```

g) smpAlgo.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import bean.node;
public class smpAlgo {

        int region;
        ArrayList<node> ar;
        ArrayList<ArrayList<node>> clonePair= new ArrayList<>();
        Connection con=null;
        double avg_energy;

        smpAlgo(int region,ArrayList<node> ar)
        {
                this.region=region;
                this.ar=new ArrayList<node>(ar);

        }

        public void findPref()
        {
                ArrayList<Integer> al1;

                for(int i=0;i<ar.size();i++)
                {
                        int flag=0;
                        al1=new ArrayList<>();
                        for(int j=0;j<ar.size();j++)
                        {
                                if(i!=j)
                                {
                                        if(ar.get(i).getNodeId()==ar.get(j).getNodeId() &&
ar.get(i).getNodeKey().equals(ar.get(j).getNodeKey()))
                                        {
                                                al1.add(j);
                                                flag=1;
                                        }
                                }
                        }
                        if(flag==1)
                         ar.get(i).setAl(al1);
                }
        }

        public String testEnergy(int i,int j)
        {
                if(clonePair.get(i).get(j).getEnergy()>avg_energy)
                {
                        return "Pass";
                }
                return "Fail";
        }

        public void smp()
        {
                ArrayList<node> cp;
                for(int i=0;i<ar.size();i++)
                {
                        cp=new ArrayList<>();
                  if(ar.get(i).getAl()!=null)
                  {
                        cp.add(ar.get(i));
                        for(int j=0;j<ar.get(i).getAl().size();j++)
                        {
                                int ind=ar.get(i).getAl(j);
                                //System.out.println(ind);
                                cp.add(ar.get(ind));
                                //System.out.println(ar.get(ind).getNodeId());
                                ar.get(ind).setAl(null);
                        }

                  clonePair.add(cp);
                  }
                }

                try {
                        con=Conn.getConnection();
                        Statement st=con.createStatement();
                        ResultSet rs=st.executeQuery("select avg(Energy) as en from clone
where Region='"+region+"'");
                        if(rs.next())
                        avg_energy=rs.getDouble("en");

                } catch (SQLException e) {
```

```
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

public void updateDb()
{
        con=Conn.getConnection();
        PreparedStatement ps=null;
        try {
         ps=con.prepareStatement("update clone set Clone='Y' where NodeId=?");
        } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
        for(int i=0;i<clonePair.size();i++)
        {

                        try {
                                ps.setInt(1,clonePair.get(i).get(0).getNodeId());
                                ps.execute();
                        } catch (SQLException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                }

        }
}

public void display()
{

        System.out.println("Clone pairs of region "+region);
        if(clonePair.size()==0)
                System.out.println(" No clones found!");

        for(int i=0;i<clonePair.size();i++)
```

```
                {
                        for(int j=0;j<clonePair.get(i).size();j++)
                        {
                        System.out.println(" "+clonePair.get(i).get(j).getNodeId()+" at location
("+clonePair.get(i).get(j).getLocX()+","+clonePair.get(i).get(j).getLocY()+")");
                        System.out.println("Energy test:"+testEnergy(i,j));
                        }
                }
                System.out.println();
        }
}
```

h) node.java

```
package bean;
import java.util.ArrayList;
public class node {
        private int NodeId, LocX,LocY,Region;
        private double Energy;
        private ArrayList<Integer> al=null;
        public ArrayList<Integer> getAl() {
                return al;
        }
        public int getAl(int i) {
                return al.get(i);
        }
        public void setAl(ArrayList<Integer> al) {
                this.al = al;
        }
        public int getNodeId() {
                return NodeId;
        }
        public void setNodeId(int nodeId) {
                NodeId = nodeId;
        }
        public int getLocX() {
                return LocX;
        }
        public void setLocX(int locX) {
```

```
		LocX = locX;
	}
	public int getLocY() {
		return LocY;
	}
	public void setLocY(int locY) {
		LocY = locY;
	}
	public int getRegion() {
		return Region;
	}
	public void setRegion(int region) {
		Region = region;
	}
	public double getEnergy() {
		return Energy;
	}
	public void setEnergy(double energy) {
		Energy = energy;
	}
	public String getNodeKey() {
		return NodeKey;
	}
	public void setNodeKey(String nodeKey) {
		NodeKey = nodeKey;
	}
	public String getClone() {
		return Clone;
	}
	public void setClone(String clone) {
		Clone = clone;
	}
	private String NodeKey, Clone;
}
```