A Key Pool Approach to Prevent a Sybil Attack in Wireless Sensor Networks

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Aditi Nandan Singh (131258)

Sarthak Sharma (131312)

Under the supervision of

Amol Vasudeva

to

Department of Computer Science & Engineering and Information Technology

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

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"A key pool approach to prevent a Sybil attack in Wireless Sensor Networks."** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** 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 2016 to May 2017 under the supervision of **Amol Vasudeva** (Assistant Professor, Computer Science & Engineering).

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

Aditi Nandan Singh, 131258

Sarthak Sharma, 131312

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

Amol Vasudeva

Assistant Professor

Computer Science & Engineering

Dated: 28-4-2017

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my guide Mr. Amol Vasudeva for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by his time to time shall carry me a long way in the journey of life on which I am about to embark. The in-time facilities provided by the Computer Science department throughout the project development are also equally acknowledgeable. At the end I would like to express my sincere thanks to all my friends and others who helped me directly or indirectly during this project work.

Date: 28-04-2017 **Aditi Nandan 131258**

Sarthak Sharma 131270

Contents

CHAPTER 1

INTRODUCTION

1.1 Introduction

1.1.1 Sybil Attack

Sybil attack was first introduced by J. R. Douceur. According to him, the Sybil attack is an attack in which a single node can rule in the system by presenting multiple fake identities and behaving like multiple legitimate nodes[8].

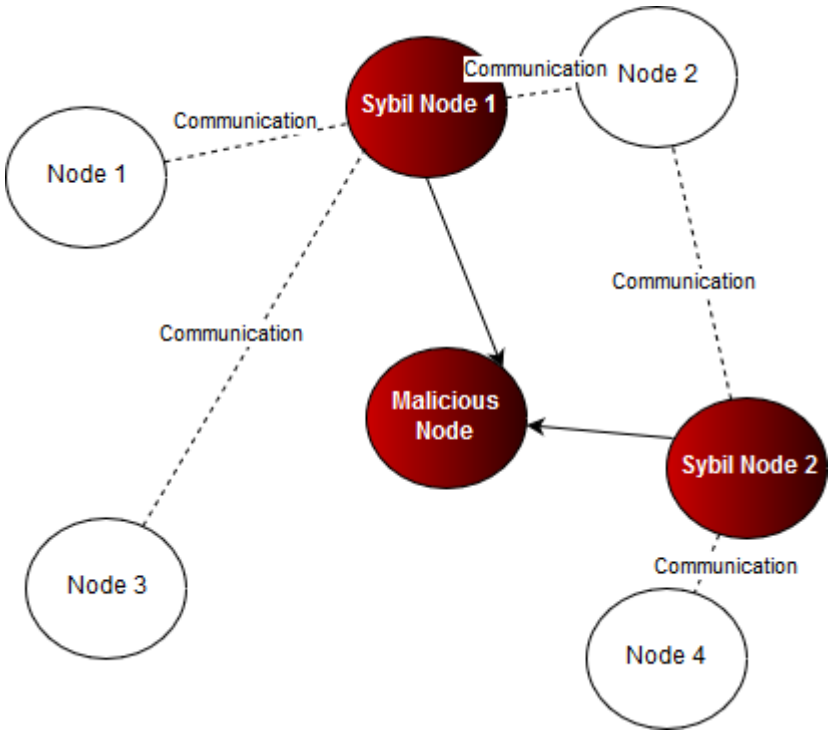


Figure 1: Sybil Nodes [4]

For a sensor node to communicate with other nodes in its vicinity, the only way is to broadcast messages on an open channel. But this methodology can be exploited if a node forges multiple fake identities. This node is called a malicious node and the subsequently forged nodes are called Sybil nodes. To the legitimate nodes communicating with the malicious node, it appears as if multiple nodes are there in the surrounding area, instead there is only one, as is shown in Figure 1.

Malicious node observes the way network functions and behaves and out of all nodes selects a target nodes, whose identity it impersonates to the other legitimate nodes

1.1.2.1 Dimensions of Sybil Attack

There are 3 dimensions to the Sybil attack i.e. communication, participation and identity[9].

I. Communication

-Direct Communication

- Legitimate nodes communicate with Sybil nodes directly.

- Indirect Communication

- Malicious nodes claim to be able to contact a Sybil nodes.
- All the messages which are to be delivered to the Sybil node are routed through the malicious node.
- Malicious nodes pretend to forward the message to a Sybil node.

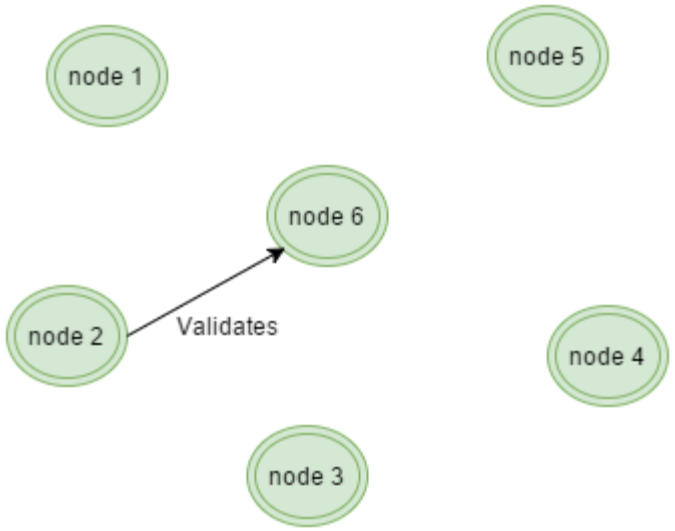


Figure 2: Direct Communication

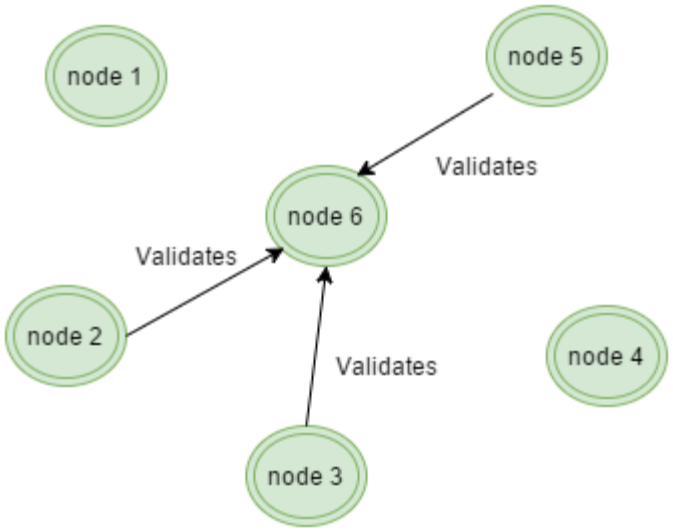


Figure 3: Indirect Communication

II. Participation

- **Simultaneous**
    - All the Sybil identities of the malicious node participate in the network at once.
    - It appears as though all the Sybil identities are present at once, but it is the malicious node cycling through them.

- **Non-Simultaneous**
    - Malicious node uses a small number of identities at a time but has a cache of a large number of identities.
    - The attacker cycles through the identities by leaving the network with one and then joining with the other.

III. Identity

- **Fabricated Identity**
    - The attacker creates random identities.

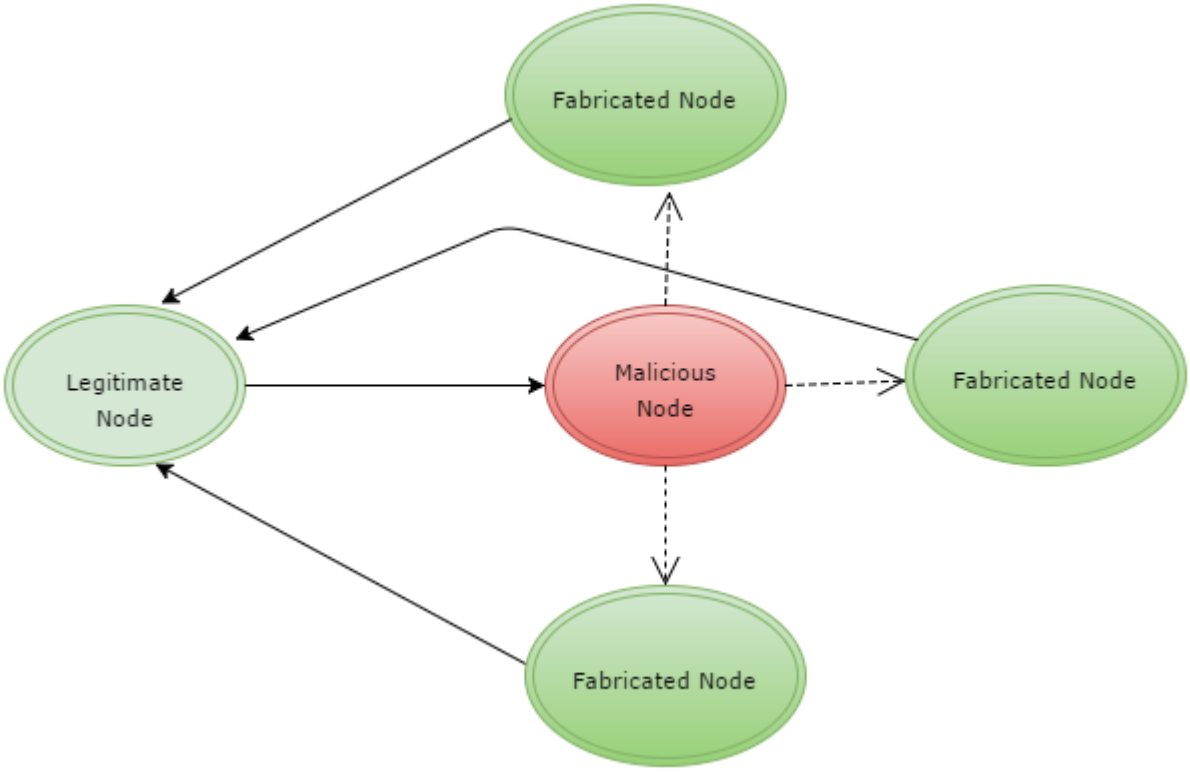


Figure 4: Fabricated Identity

- **Stolen Identity**
    - Sybil nodes are assigned identities.
    - An attacker steals the identity of a Sybil node.
    - Sybil attack is even more difficult to detect if the attacker has disabled or destroyed the original legitimate node.

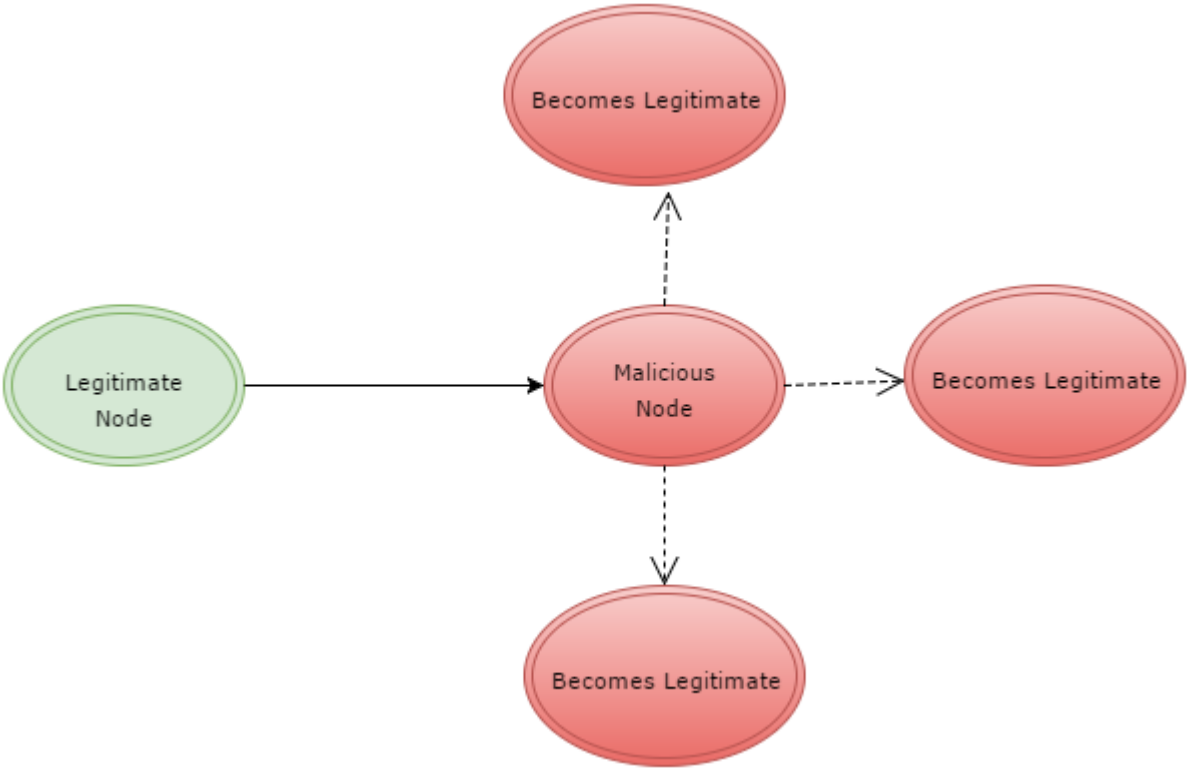


Figure 5: Stolen Identity

1.1.2.2 Sybil attack effect

If even a single malicious node is able to convince the network of its multiple identities the system is heavily compromised. Even worse, if the malicious node is able to become a cluster head it can take control of a large amount of the network. Once the Sybil attack succeeds, it has the ability to open the doors for many other attacks also making the system insecure[9].

I. Resource allocation

Once Sybil attack succeeds it has considerable influence over the resource distribution of the network. For example, there is a limited amount of bandwidth in a sensor network, so it is often assigned to each node per time basis. But, a malicious node can impersonate multiple Sybil nodes and gain a disproportionate amount of time using the identities of multiple legitimate nodes. Which in turn create resource availability issues in the network.

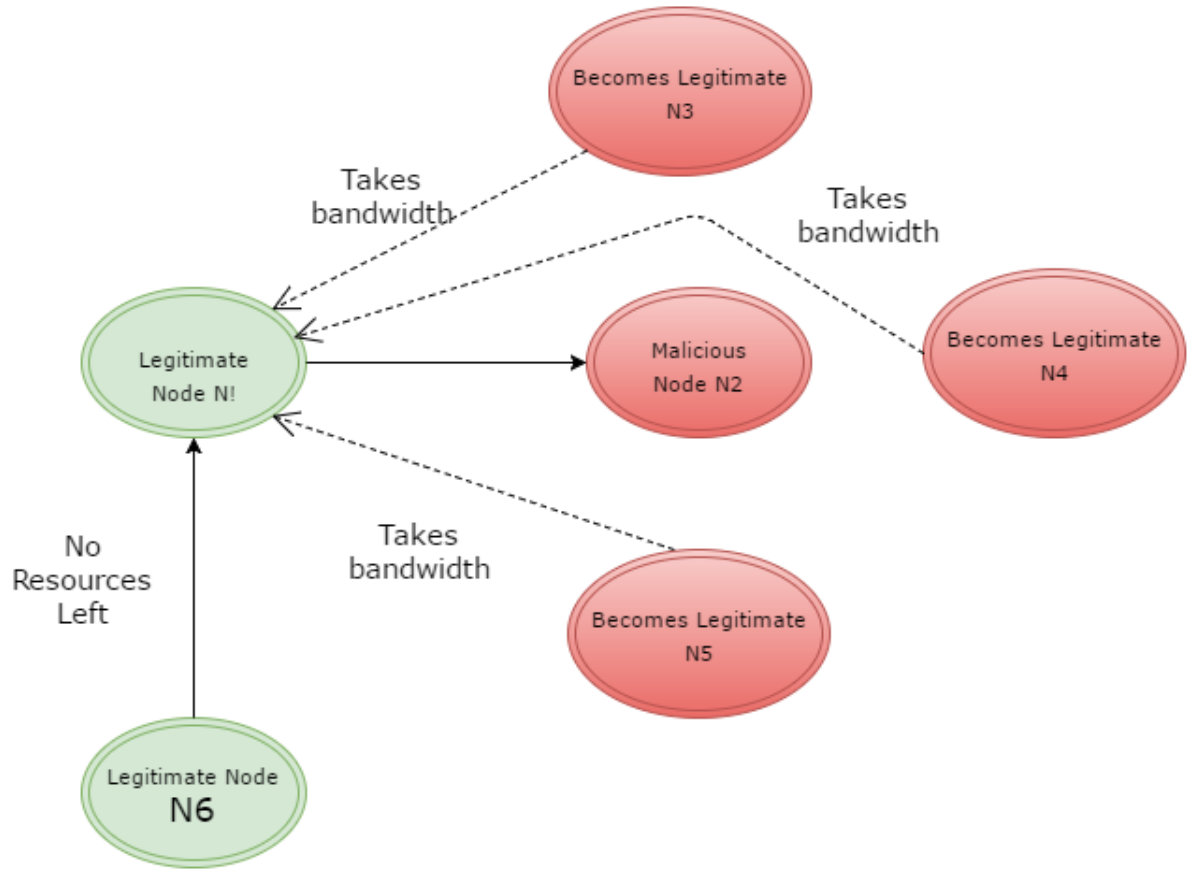


Figure 6: Sybil attack in resource allocation [1]

II. Voting

The sensor networks often use voting as a method to take many important decisions such as cluster head selection and reporting malicious nodes. In this case, since the malicious node has multiple Sybil identities, it holds a massive advantage over other nodes when it comes to voting. It actively participates in voting and often sends vote with the identity of legitimate nodes and affect the voting process. In addition to this, the malicious node can also act as a sinkhole if it's a Cluster Head and prevent the voting of a lot of nodes. It can also impact the voting process to ensure the malicious node is elected as Cluster Head.

III. Distributed Storage

Network nodes which use a distributed system of data storage are also severely affected by the Sybil attack. In the distributed system the nodes prefer to store data in multiple locations so as to ensure that if one site fails they can retrieve the data from another site. Since a malicious node claims the identity of various Sybil node the legitimate nodes often store data on Sybil nodes thinking that the data has been stored in a different location, but in reality, it is at the same location.

In figure 6 n0 is a malicious node impersonating n1, n2, n3. Processes p0 and p1 who need to store data in a distributed way choose to store data on n3and n2, and n2 and n1 respectively. But, in reality, they are all stored in n0.

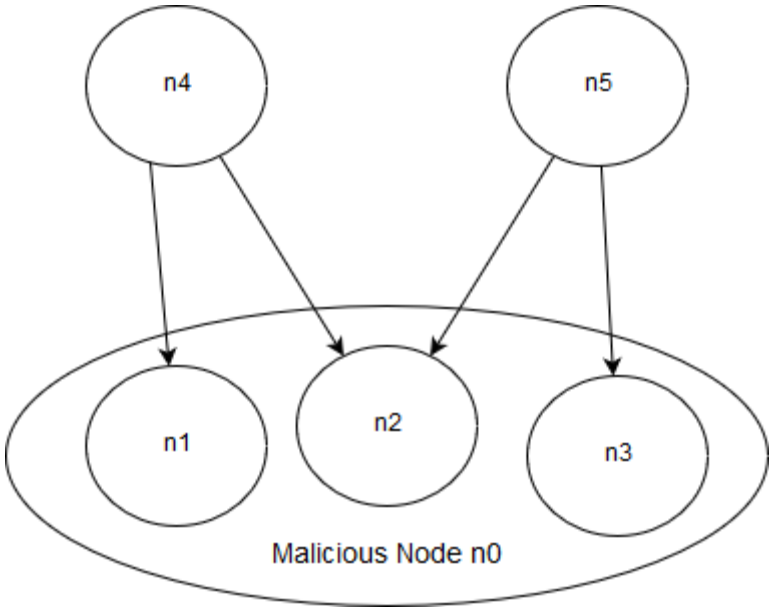


Figure 7: Sybil attack on distributed storage [2]

IV. Data Aggregation

In most application sensor networks are deployed to receive and collect data from its surroundings. The networks need immediate updates from the sensors of the nodes. A Sybil node can fabricate data sets and send false data to other nodes claiming it came from another node. This can create a lot of failures as this data can be used by another application to work on other things. Even worse it could be needed by the same application to process some other data.

Routing

To find the most efficient path to transport data from source to the destination we use routing protocols. A malicious node can paralyze the network by disrupting its routing protocol. When a malicious node is connected to a network through multiple Sybil identities, the routing protocol often chooses the Sybil node to route information. This wrong route selection often results in inefficient usage of bandwidth and poor network performance. In the following figure due there being the presence of two nodes n1 and n6 in place of the real node n5, it creates an ambiguous network routing path.

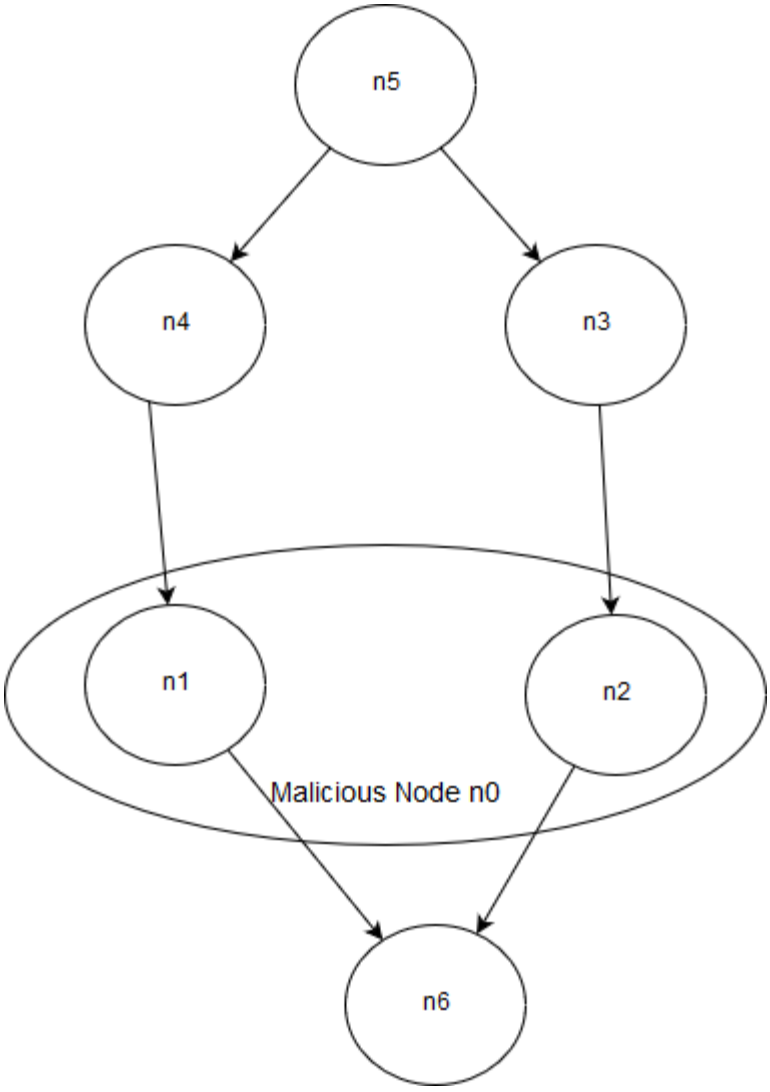


Figure 8: Sybil attack effect on Routing [3]

VI. Hidden Node

From the above attacks, the Sybil node increases its own credibility and trustworthiness in the network and decrease the overall reliability of the network. The network is much vulnerable to a various attack such as denial of service attacks, sinkhole attacks, and

black hole attack.

1.1.2 RANDOM KEY PRE-DISTRIBUTION SCHEME

Sensor networks despite being a promising technology suffer with many security limitations. Most of these arise due to limited resources such as memory, bandwidth, and transmission power. These limitations render public key cryptosystems impractical, making the nodes susceptible to aforementioned Sybil attack. To counter this Eschenauer and Gligor [2] proposed a new approach of key distribution using a random subset of keys from a key pool. The basic scheme of this approach was in 4 phases, i.e. Initialization, Node Deployment, Key Setup and Path Key Generation.

I. **Initialization**
   - A random set of Keys S out of the total possible key space is picked
   - For each node, we randomly select m keys called Key Ring from S and store in the node memory.
   - The criteria for selecting 2 random subsets of size m in S is the probability of both of them sharing at least one key.

II. **Deployment**
   - An identifier is assigned to each key before deployment.
   - Sensor nodes are deployed.

III. **Key Setup Phase**
   - Each of the nodes broadcast their respective identifiers to the extent of their range.
   - All the nodes containing shared keys in their Key Ring would either directly or indirectly as mentioned above would verify that their neighbor is a legitimate node or not.

IV. **Path Key Generation**
   - A secure link in the form of a connected graph is formed.
   - Nodes setup path keys with nodes in their vicinity whose shared key are not present with them
   - Source nodes generate a path through which it can communicate with the destination node using routing algorithms.

In the Key Setup Phase as mentioned above, there are two ways to verify a neighbor: direct validation and indirect validation. In direct validation, the claimant node and verifier node compare their Key Ring directly to check whether they have a common key or not. Whereas in indirect validation the claimant node is verified by many verifier nodes to check all the keys on the key ring of the claimant node. In retrospection, the indirect node is much more secure as compared to direct validation method as more than one common key is examined and the node is vetted by more nodes. The actual number of nodes who vetted the claimant node for it

to be verified can be subject to the total number of nodes and number of keys on a key ring. Whereas, direct validation is much faster. Therefore, the choice of method of validation depends on the application the sensor nodes are being used for.

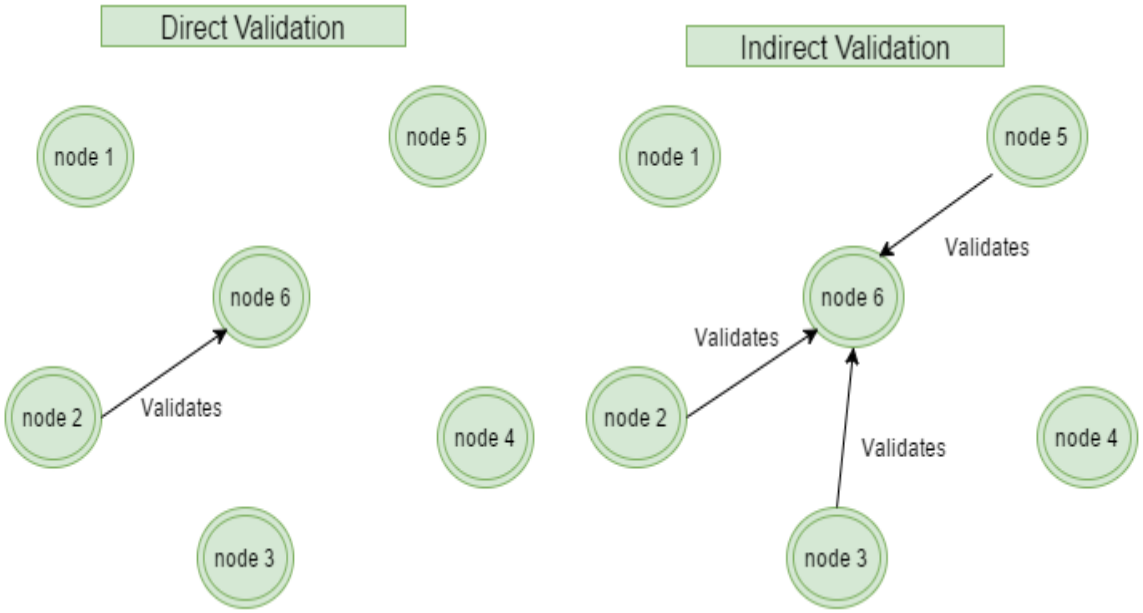


Figure 9: Direct and Indirect Validation

The above method is further improved upon in terms of security by using a special pseudo-random function in choosing the key rings of the respective nodes using their identifiers as seed. This could also be used to determine whether a node in the vicinity of another one has a shared key or not, greatly reducing the amount of communication needed to verify a node making the verification step faster and more efficient.

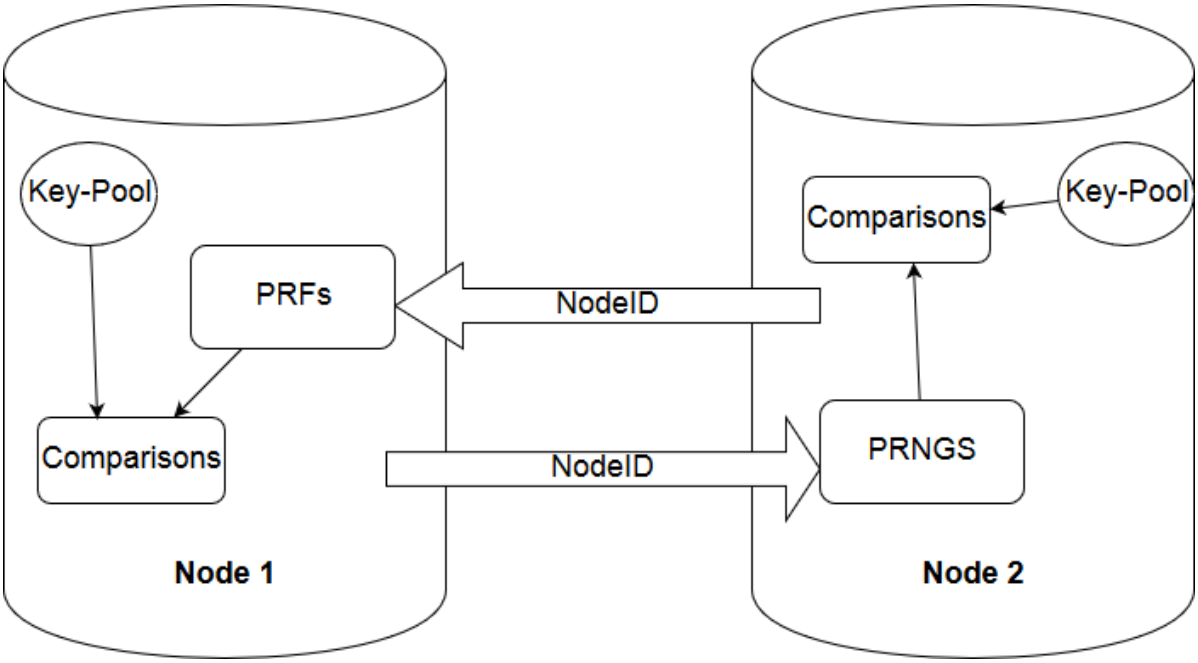


Figure 10: Demonstration of Random key pre-distribution

1.2 PROBLEM STATEMENT

The Random key pre-distribution scheme is a viable option if we wish to defend against Sybil attack. In this scheme, for each node we assign them a key pool of m keys. Using the PRF function which gives the output of the indices of the array of the key-pool Bi we can easily find common keys between the two nodes. After authentication we can establish a secure connection between them by an exchange of keys. The only limitation that his method has is the fact that if by farming a lot of information about common key indexes, the malicious node is able to create its own key pool.

1.3 OBJECTIVES

- To make a program to simulate a sensor network with proper communication channels.
- To program and insert a malicious node able to launch a Sybil attack.
- To propose an algorithm for implementation of Random Key Pre-distribution scheme.
- To study the effects of the said algorithm on the simulated network.
- To perform the implementation JavaScript with the use of the p5 module.

1.4 METHODOLOGY

This section deals with an explanation of the methodology used to build this project and achieve all the objectives. The used method is one which entails to the swift development of the project and also gives the best result possible. In order to implement this project, the methodology used is a modification of System Development Life Cycle (SDLC) involving 3 steps which are given as follows-

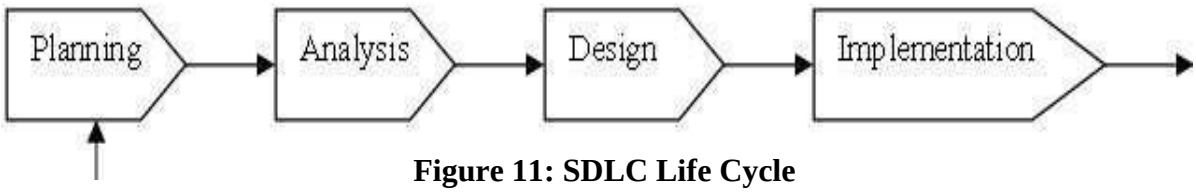


Figure 11: SDLC Life Cycle

I. Planning

- Collection of all the data available in public domain such as research papers
- Deciding the software and the hardware to use to implement the project

II. Implementing

- Deciding the various points on which we have to test the project
- Development of the project

III. Analysis

- Performance Analysis

1.5 ORGANIZATION

Chapter 1 highlights and underlines the Random Key Pre-Distribution system's characteristics. In this chapter, we discuss how Sybil Attack utilizes security flaws in sensor networks. The key focus, however, remains on how to implement a Random Key Pre-Distribution system using key pool approach to prevent Sybil attack.

The collection and review of all the collected literature from various journals, publishing websites and conferences are presented in **Chapter 2.**

Chapter 3 covers the development of system used, the algorithm which is proposed and its explanation.

Implementation of the algorithm and its performance analysis is given in **Chapter 4**. The simulation results and screenshots are shown in this chapter.

Chapter 5 entails the future scope and conclusion to this project's implementation to guide future research for this subject.

CHAPTER 2

LITERATURE SURVEY

To fully understand the project and research the best possible solution of the problem a number of magazines, scholarly journals and research papers have been read and investigated in detail. This chapter entails all the findings and important materials from the said sources to be able to design the solution to the stated problem.

Many scientists and research scholars have published various papers related to the given problem. This part contains all the essential extracts from those papers.

2.1 The Sybil Attack in Sensor Networks: Analysis & Defenses by J. Newsome, E. Shi, D. Song and A. Perrig [5]

Sensor networks one of the most disruptive technology and being an economically viable solution to Traffic monitoring, Border security and pollution testing have a large number of security concerns. Since instead of securely sending data to one source its broadcasts its data. It's hardware is also not secure to tampering. Also since it is needed to keep costs low for it to be economically viable they cannot use hardware able to efficiently run hardware which can both be able to run with low security and provide large computation power. Sin Sybil attack the malicious node acts as if it were not one but a number of nodes spoofing the other nodes identity or generating fake ones. Worst case scenario a malicious node can generate a number of such nodes using just one physical device.

Sybil attack taxonomy

Sybil attack was first introduced by J. R. Douceur [8]. According to Douceur, a single entity rule the most in the system by presenting multiple faked identities in Sybil attack[8].

Dimensions of Sybil Attack

A Sybil attack can be represented using the three dimensions that are Communication, participation, and identity.

Dimension	Types	
Communication	Direct Communication	Indirect Communication
	Nodes communicate directly to the malicious node.	Nodes do not communicate directly to the malicious node.
Participation	Simultaneous Participation	Non Simultaneous
	Participation of all Sybil nodes with legitimate nodes at once.	Participation of all Sybil nodes with legitimate nodes one by cycling through the fake identities at once.
Identity	Fabricated Identity	Stolen Identity
	Malicious node randomly generates an identity for the Sybil node.	Malicious node steals the identities of the legitimate node.

Table 2. Dimensions of Sybil attack

Effects of Sybil Attack--

The following shows the ways the Sybil node can cause damage to the network-

- **Resource Allocation** -- Once Sybil attack succeeds it impersonates multiple Sybil nodes and gain a disproportionate amount resource using the identities of multiple legitimate nodes. Which in turn create resource availability issues in the network.

- **Voting** -- The sensor networks often use voting to take decisions. The malicious node has multiple Sybil identities, therefore it holds a massive advantage over other nodes when it comes to voting. It sends vote with the identity of legitimate nodes and affects the voting process, acts as a sinkhole if it's a Cluster Head and prevent the voting of a lot of nodes.

- **Distributed Storage** -- A malicious node claims identity of various Sybil node the legitimate nodes often store data on Sybil nodes thinking that the data has been stored in a different location, but in reality, it is at the same location.

- **Routing** -- When a malicious node is connected to a network through multiple Sybil identities, the routing protocol often chooses the Sybil node to route information. This

wrong route selection often results in inefficient usage of bandwidth and poor network performance.

- **Data Aggregation** -- A Sybil node can fabricate data sets and send false data to other nodes claiming it came from another node. This can create a lot of failures as this data can be used by another application to work on other things.

Random Key Pre-distribution

To allow secure authentication and connection establishment between two nodes a promising technique named random key pre-distribution has been proposed by scholars. These techniques allow us to communicate with nodes securely. In this section, we will see how the key pre-distribution scheme works.

In this scheme, we assign a set of random keys or information which pertains to the key to every legitimate node. Therefore in key set-up phase, nodes can compute common keys with the help of provided pseudo random function thereby ensure node to node secrecy. The basic gist of this is-

1. Every key provided to the node is related to its node ID.

2. All the nodes must be able to authenticate the keys using the ID of the node which wishes to communicate with the.

Since the number of keys given is limited the probability that -

- The malicious node is able to guess the common key is minimized by maximizing the number of keys assigned to the nodes.
- The random ID generated will have the same set of key is minimized by minimizing the number of keys assigned to the nodes

So the number optimum so that it accomplishes both of the tasks.

In the Key Setup Phase as mentioned above, there are two ways to verify a neighbor: direct validation and indirect validation. In direct validation, the claimant node and verifier node compare their Key Ring directly to check whether they have a common key or not. Whereas in indirect validation the claimant node is verified by many verifier nodes to check all the keys on the key ring of the claimant node. In retrospection, the indirect node is much more secure as compared to direct validation method as more than one common key is examined and the node is vetted by more nodes. The actual number of nodes who vetted the claimant node for it to be verified can be subject to the total number of nodes and number of keys on a key ring. Whereas, direct validation is much faster. Therefore, the choice of method of validation depends on the application the sensor nodes are being used for.

Key pre-distribution techniques also have different variants. This may include –

- The basic key pool approach
- The single-space pairwise key distribution approach
- The multi-space pairwise key distribution approach

The researchers who have proposed these scheme to establish secret keys have studied this scheme to establish a connection between neighboring nodes. But the authors of this paper have studied the said scheme again so as to study the effect of these scheme with a Sybil attack underway. Then they compare its effectiveness against other key pre-distribution scheme.

Key Pool

All the previous research chooses to focus on pre-distribution aspect rather than authentication one. The authors now have modified the given pre-distribution scheme to now be used against Sybil attack and then analyze its effectiveness.

During the initialization phase, the system assigns random key- pools to each node. If two nodes are in the vicinity to each other and they have a common key they can easily authenticate each other and subsequently communicate.

We first assume during its usage when the nodes are initialized the node have their node IDs are sorted. The only drawback of this scheme is if the attacker is able to compromise multiple nodes, they get the ability to compromise the whole network as they get access to the key pool. Let $\Omega(ID) = \{K\beta1, K\beta2 , \ldots , K\beta k\}$ be the set of keys assigned to ID, where ID is the identity of the node, βi is the index of its ith key in the key pool. Let the function by which the key in the nodes key pool determined by be $\beta i = PRFH(ID)(i)$, where H is a hash function, and PRF is a pseudo-random function i.e node's ith key's index is determined by a pseudo-random function with function's key being H(ID), and i as its input. Many methods similar to this has been proposed to optimize this. The authors now set to prove that this method is effective with respect to Sybil attack. Here since the PRF function has its output from 1….k it's very tough to find a key which will give the same output which was generated by the key of the node. The feature which ensures this is the hash function's one-wariness which guarantees that even in a case if the attackers gain the keys to a PRF they would still be unable to find the pre-image to the same i.e. the identity of the node since they do not know of the function that has been used.

An assailant may endeavor to produce new characters to use in the Sybil assault. To do this, he should catch real nodes and perused off the keys, therefore developing a traded off key pool S. He will then endeavor to manufacture usable Sybil personalities. On the off chance that a made-up personality ID# can take an interest in the sensor arrange without being recognized in the key statement stage, we call it a usable Sybil character.

A usable Sybil character must have the capacity to pass the approval by different nodes. To approve a personality, the verifier challenges the character by asking for it to demonstrate that it has at least one keys it cases to have. In the event that $\exists Ki, Ki \in \Omega(ID\#), Ki \in /S$, and if some

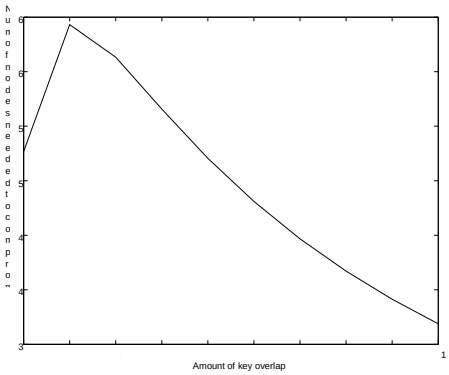
authentic substance E in the sensor arrange knows Ki, then E can find that ID' is duping by testing ID' utilizing Ki. To accomplish a universally reliable result, it is important to perform backhanded approval, which is the situation we should talk about.

Approval should be possible at various granularities. One extraordinary is the situation of full approval where the sensor arranges tries to confirm whatever a number of a node's keys as could be allowed, rendering a Sybil assault more troublesome. Practically speaking, in any case, full approval requires that each node challenges each other node in the system, which could bring about exorbitant correspondence overhead and the capability of DOS assaults. To stay away from these disadvantages we could confine the extent of approval. For example, we could restrain the approval procedure inside the region of the node being approved, for example, arbitrarily choosing d nodes out of its k-bounce neighborhood to mutually play out the approval. The bigger d and k are, an assailant will be less inclined to succeed; then again, the approval will be more costly.

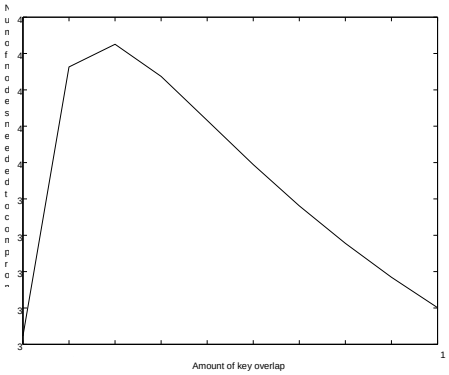
2.2 Random Key Pre-distribution Schemes for Sensor Networks by Haowen Chan, Adrian Perrig, and Dawn Song

2.2.1Q-COMPOSITE RANDOM KEY PRE-DISTRIBUTION SCHEME

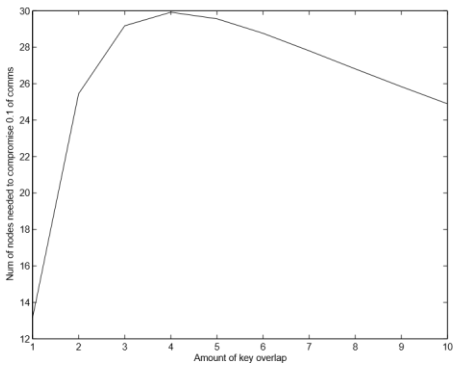
In the essential plan, two nodes need to locate a solitary basic key from their key rings with a specific end goal to build up correspondences. We propose an alteration to the essential plan whereby q regular keys ($q \geq 1$) are required, rather than only a solitary one.



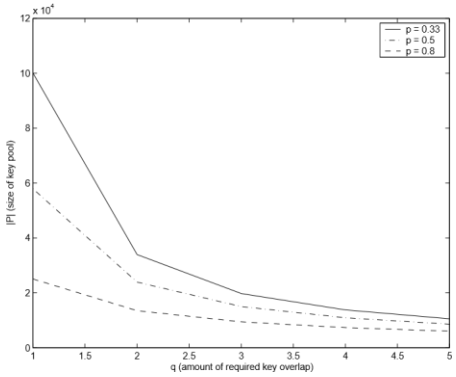
(a) m =200, p =0.33



(b) m =200, p =0.5



(c) m =200, p =0.8



(d) Key pool size $|P|$ vs q for p =0.33, 0.5,0.8.

Figure 12[9]: Figures (a)–(c) above mirror the number of nodes that the enemy needs to catch before it can break any given connection with likelihood 0.1, for different measures of required key cover (q = 1 speaks to the fundamental plan). m is the number of keys put away in every node. p is the required likelihood of any two neighbors having the capacity to set up a connection (see Equation 2). Figure (d) mirrors the required key pool measure (in 10,000s of keys) for different measures of key cover, for different p.

By expanding the measure of key cover required for key-setup, the versatility of the system against node catch and penetration can be expanded.

Figure 12 mirrors the inspiration for the q-composite keys conspire. As the measure of required key cover expands, it turns out to be exponentially harder for an assailant with an offered key set to reprieve a connection. Be that as it may, keeping in mind the end goal to save the given likelihood p of two nodes sharing adequate keys to set up a protected connection, it is important to lessen the span of the key pool |P|. This enables the aggressor to pick up a bigger example of P by breaking fewer nodes. The exchange of these two contradicting components brings about an ideal measure of key cover so as to represent the best impediment to an aggressor for some coveted certainty of breaking a connection.

From Figure 11 it can be effectively observed that as p builds, the ideal measure of cover increments. Along these lines, the q-composite plan is most suited for situations where nodes require a high likelihood of interfacing with any given neighbor.

Description of the $_q$-composite keys scheme

1) Initialization and Key Setup

In the initialization phase, we pick a requested arrangement of P irregular keys out of the aggregate key space, where |P| is processed as depicted later. For every node, we select m arbitrary keys from P and store them into the node's key ring in the request in which they occurred in P.

In the key-setup stage, every node must find all basic keys it has with each of its neighbors. This can be refined with a basic neighborhood communicate of every single key identifier that a node has. While communicate based key revelation is trifling to actualize, it has the impediment that an easygoing meddler can distinguish the key arrangements of the considerable number of nodes in a system and in this way pick an ideal arrangement of nodes to bargain so as to find a substantial subset of the key pool P. A more secure, however slower, strategy for key disclosure is issue m customer perplexes (one for each of the m keys) to each neighboring node. Any node that reacts with the right response to the customer bewilder is along these lines distinguished as knowing the related key.

After key disclosure, every node can distinguish each neighbor node with which it shares at any rate q keys. Let the number of real keys shared be q0, where $q0 \geq q$. Another correspondence connect key K is produced as the hash of all mutual keys, e.g., K = hash(k1||k2||...||kq0). The keys are hashed in some accepted request, for instance, in view of the request they happen in the first key pool P. Key-setup is not performed between nodes that share less than q keys.

2) Computation of key pool size

Give m a chance to be the number of keys that a solitary node can hold in its key ring. For a picked cover parameter q, we wish to create a pool P of keys, where |P| is picked with the end goal that the likelihood of any two nodes having in any event q enters in like manner is equivalent to some given likelihood p. p is gotten from the physical particulars of the system by means of Equation 2 (see Section 3).

We figure |P| as takes after. Give p(i) a chance to be the likelihood that any two nodes have precisely i enters in like manner. Any given node has distinctive methods for picking its m keys from the key pool of size |P|. Thus, the aggregate number of courses for both nodes to pick m keys each is. Assume the two nodes have I enters in like manner.

There are approaches to pick the i regular keys. After the i normal keys have been picked, there stay 2(m − i) unmistakable keys in the two key rings that must be picked from the rest of the pool of |P| − i keys. The number of approaches to do this is. The 2(m − i) particular keys should then be apportioned between the two nodes similarly. The number of such equivalent allotments is. Henceforth the aggregate number of approaches to pick two key rings with i enters in like manner is the result of the previously mentioned terms, i.e., Thus, we have

$$p(i) = \frac{\binom{|P|}{i}\binom{|P|-i}{2(m-i)}\binom{2(m-i)}{m-i}}{\binom{|P|}{m}^2} \tag{3}$$

Let $p_{connect}$ be the probability of any two nodes sharing sufficient keys to form a secure connection. $p_{connect}$ = 1− (probability that the two nodes share insufficient keys to form a connection), hence

$$p_{connect} = 1 - (p(0) + p(1) + \cdots + p(q-1)) \tag{4}$$

For a given key ring size m, minimum key overlap q, and minimum connection probability p, we choose the largest |P| such that $p_{connect} \geq p$.

The variation of |P| with key overlap q and probability of connection p is shown in Figure 1d.

B. Evaluation of the $_q$-composite random key distribution scheme

We assess the q-composite irregular key dispersion conspire as far as versatility against node catch and the most extreme system estimate upheld. We take note of that this plan has no resistance against node replication since node degree is not compelled and there is no restriction on the number of times each key can be utilized. The plan can just bolster node disavowal by means of a trusted base station. Such a renouncement plan is depicted by Eschenauer and Gligor in their portrayal of the essential plan.

1) Resilience against node capture in q-composite keys schemes

The q-composite key plan reinforces the system's strength against node catch when the number of nodes caught is low. Give the number of traded off nodes a chance to be x. Since every node contains m keys, the likelihood that a given key has not been traded off is. Thus the normal portion of aggregate keys traded off is. For any correspondence connect between two nodes, if its connection key was the hash of qs shared keys, then the likelihood of that connection being traded off is. Consequently, we have that the portion of aggregate interchanges traded off is

$$\sum_{i=q}^{m} \left(1 - \left(1 - \frac{m}{|P|}\right)^{x}\right)^{i} \frac{p(i)}{p_{connect}}$$

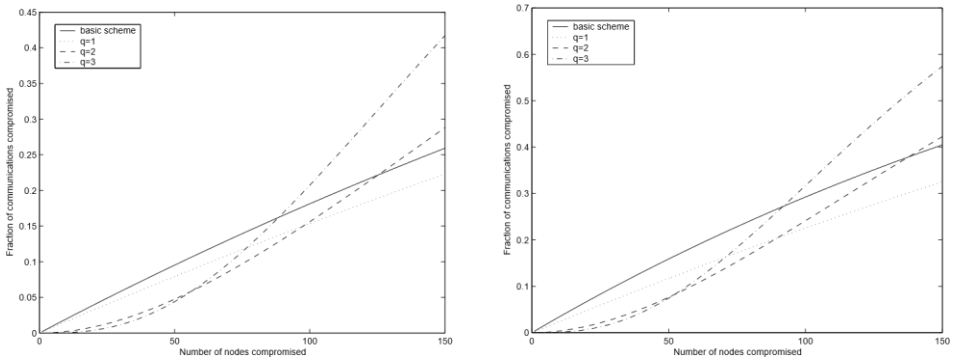
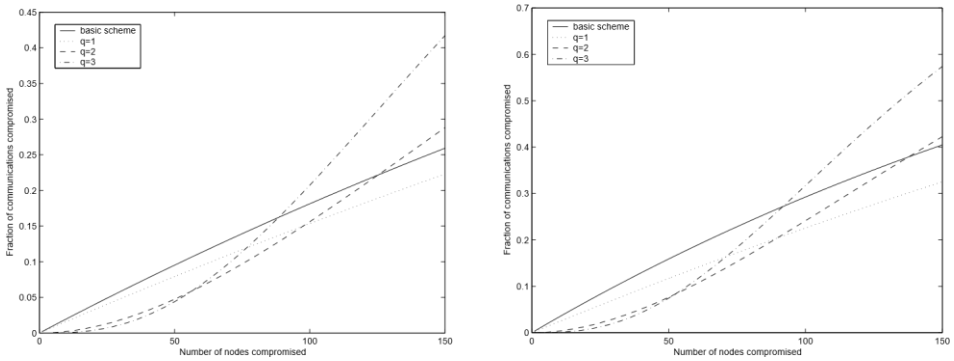


Figure 2 demonstrates the portion of extra interchanges (i.e., outer correspondences in the system free of the caught nodes) that a foe can trade-off in view of the data recovered from x number of caught nodes. Note that the x-tomahawks are outright quantities of nodes traded off (i.e., free of the real aggregate size of the system) while the y-tomahawks are parts of the aggregate system correspondences bargained. It is along these lines instantly obvious that the plans are not vastly adaptable - a tradeoff of x number of nodes will dependably uncover y division of the aggregate correspondences in the system paying little respect to how huge the system is. A strategy to assess the biggest supportable system size of the different plans is talked about later.

It can be seen that the q composite keys plans offer more prominent versatility against node catch when the number of nodes caught is little. For instance, for q = 2, the measure of extra correspondences bargained when 50 nodes have been traded off is 4.74%, instead of 9.52% for the essential plan. In any case, when extensive quantities of nodes have been traded off, the q-composite keys plans have a tendency to uncover bigger divisions of the system to the enemy. By expanding q, we make it harder for an enemy to get little measures of introductory data from the system by means of few beginning node catches. This comes at the cost of making the system more helpless once countless have been ruptured. This is an alluring property in light of the fact that the bringing of introductory result down to the foe of littler scale organize

breaks makes it important for them to focus on assaulting a huge extent of the system. This speaks to a bigger duty of time and assets and adequately dissuades little scale assaults.



In evaluating the adequacy of the q-composite plan, we take note of the convergence purposes of the lines for q-composite and the essential plan in Figure 2[2]. For instance, in 2a, q = 2 is more terrible than the fundamental plan after around 90 nodes have been traded off. The q = 2 composite plan ought to consequently just be actualized in such a system in the event that it is trusted that the foe either does not have the assets or does not have the inspiration to thoroughly trade-off altogether more than 90 nodes. One potential method for restricting the enemy's inspiration to assault is to constrain the aggregate number of nodes in the system. In any case, there are numerous situations (e.g., security related sensors, and some military applications) where notwithstanding restricting the size of the sensor arrange organization may not be an adequate to constrain an enemy's impetus for mounting a critical assault against the system. In all cases, along these lines, a cautious investigation of the application ought to, therefore, be led before the q-composite plan is chosen.

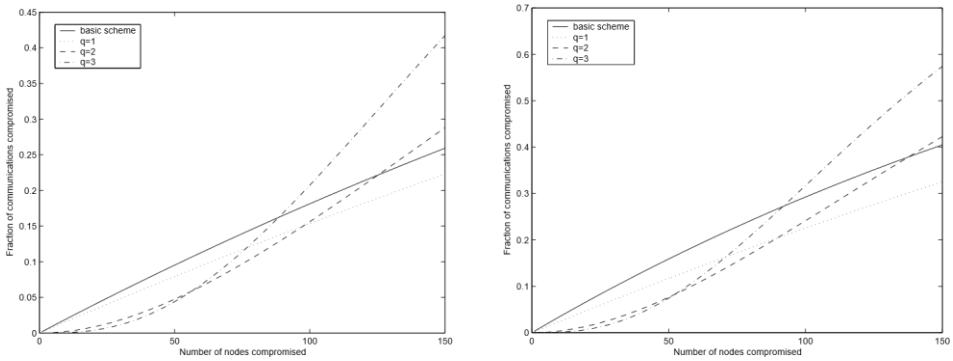
2) Maximum supportable network sizes for the q-composite keys scheme

Since a settled number of traded off nodes makes a small amount of the rest of the system end up plainly shaky, these arbitrary key conveyance plans can't be utilized for subjectively substantial systems if versatility against node catch is to be held. For instance, in the essential plan, the catch of 50 nodes bargains roughly.

9.5% of correspondences in the system. For a system of 10,000 nodes this means a rough

(a) m =200, p =0.33 (b) m =200, p =0.5

Figure 13[2]: The figures demonstrate the likelihood that a particular arbitrary correspondence



interface between two irregular nodes A, B can be unscrambled by the foe when the foe has caught some arrangement of x nodes that does exclude An or B. m is the number of keys put away in every node. p is the likelihood of any two neighbors having the capacity to set up a safe connection.

The result of 10% of interchanges traded off for a cost to the aggressor of catching only 0.5% of aggregate nodes, speaking to an enormous welcome to assault for potential enemies.

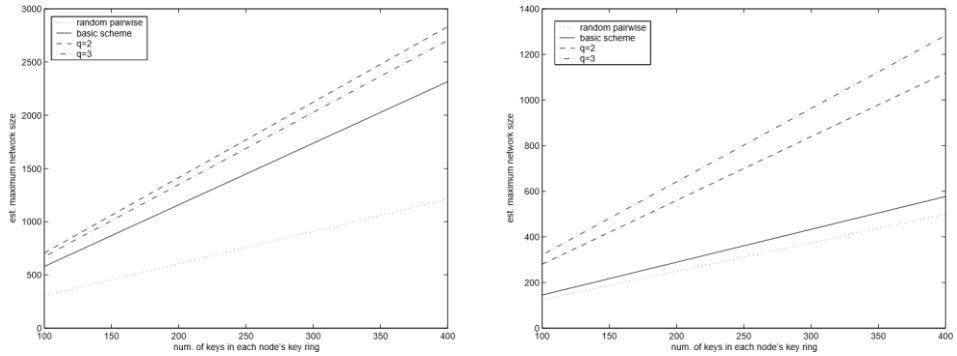
We can gauge a system's most extreme upheld estimate by encircling the accompanying prerequisite:

Limited global payoff requirement: Suppose the enemy has caught a few nodes, however is just ready to break some division fc ≤ fthreshold of all correspondences. We require that each resulting node that is bargained to the foe enables them to soften the same number of connections up whatever is left of the system, on desire, as the normal availability level of a solitary node.

As it were, given that the system is still generally secure (fc ≤ fthreshold), we might want that, by and large, in the wake of catching some node, the foe does not take in more about whatever remains of the system than they find out about the interchanges of the node itself. By means of this necessity, littler scale assaults on a system must be for the most part monetarily legitimized by the immediate interchanges estimation of the individual nodes traded off as opposed to the measure of data that the caught keys can uncover in whatever remains of the system, in this manner restricting the impetus of an enemy to start an assault.

We can consequently assess the most extreme passable sizes for the systems all together that our prerequisite remains constant.

Give x a chance to be the number of nodes traded off with the end goal that some part fthreshold of the aggregate correspondences in the system has been bargained. Let the normal availability level of a solitary node be d. The enemy in this manner holds a normal xd associations in which the traded off nodes are straightforwardly included. We require that the number of extra connections traded off somewhere else in the system be not as much as this

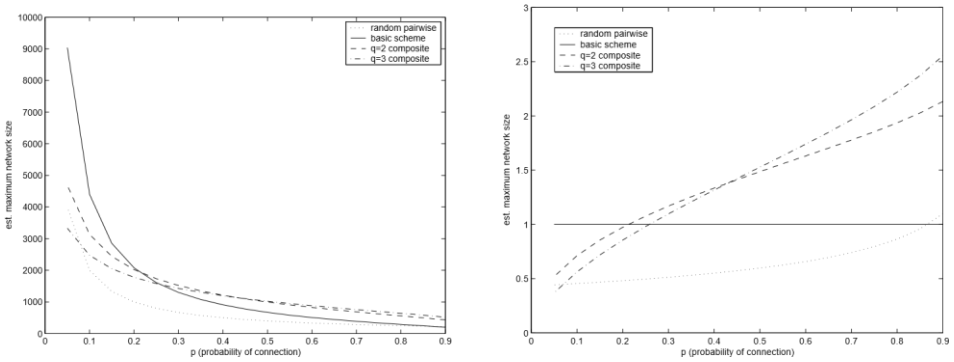


number of straightforwardly bargained joins. There are nd2 add up to joins in the system. Thus, the necessity is that xd. Streamlining,

$$n \leq 2x(1 + \frac{1}{f_{threshold}}) \tag{5}$$

Figure 14[2] demonstrates the evaluated greatest system sizes for the essential irregular keys conspire and also for a few parameters of the q-composite keys plot. We take note of that the most extreme system sizes scale straightly with key ring size m. Likewise, as association likelihood between two nodes p builds, the most extreme supportable system measure for the q-composite keys plans with bigger q perform progressively well against the essential plan. This takes after from our perception that the q-composite keys plans perform relatively better against the fundamental plan when p is expansive.

(a) p =0.33, *fthreshold* =0.1 (b) p =0.8, *fthreshold* =0.1



(c) m =200, *fthreshold* =0.1 (d) Maximum network sizes of the *q*composite scheme as a fraction of the basic scheme, m =200, *fthreshold* =0.1, various p

Figure 15[2]: Maximum network sizes for various network parameters

CHAPTER 3

SYSTEM DEVELOPMENT

3.1 Algorithm for Key Distribution among Nodes

Assumptions

1. Let *N* be the number of nodes in a Mobile Sensor Network, at any instant of time, with their IDs as
$$X^i : i = 1,2,\ldots\ldots, N.$$
2. There are P number of keys in the Pool, represented by K^i: i=1, 2, 3……., M.
3. Each index in pool represents its respective key for a specific node.
4. A Pseudo Random Function (PRF) is used which can distribute the keys evenly. It takes id as the argument i.e. PRF (id) and outputs key index.

Steps

1. Node A (X^1) tries to communicate with Node B (X^2). Now Node B validates the identity of Node B. Both nodes broadcast their id's.
2. For validating they find common keys between themselves. Key indexes can be calculated with PRF (id) function.
3. Node A calculates the key indexes of Node B with PRF (2).
4. After finding the key indexes of Node B, Node A finds common key indexes with it.
5. If Node A finds the common key, it sends encrypted key and key index to Node B.
6. Node B then verifies the key index with its PRF (1) function and validates the session.

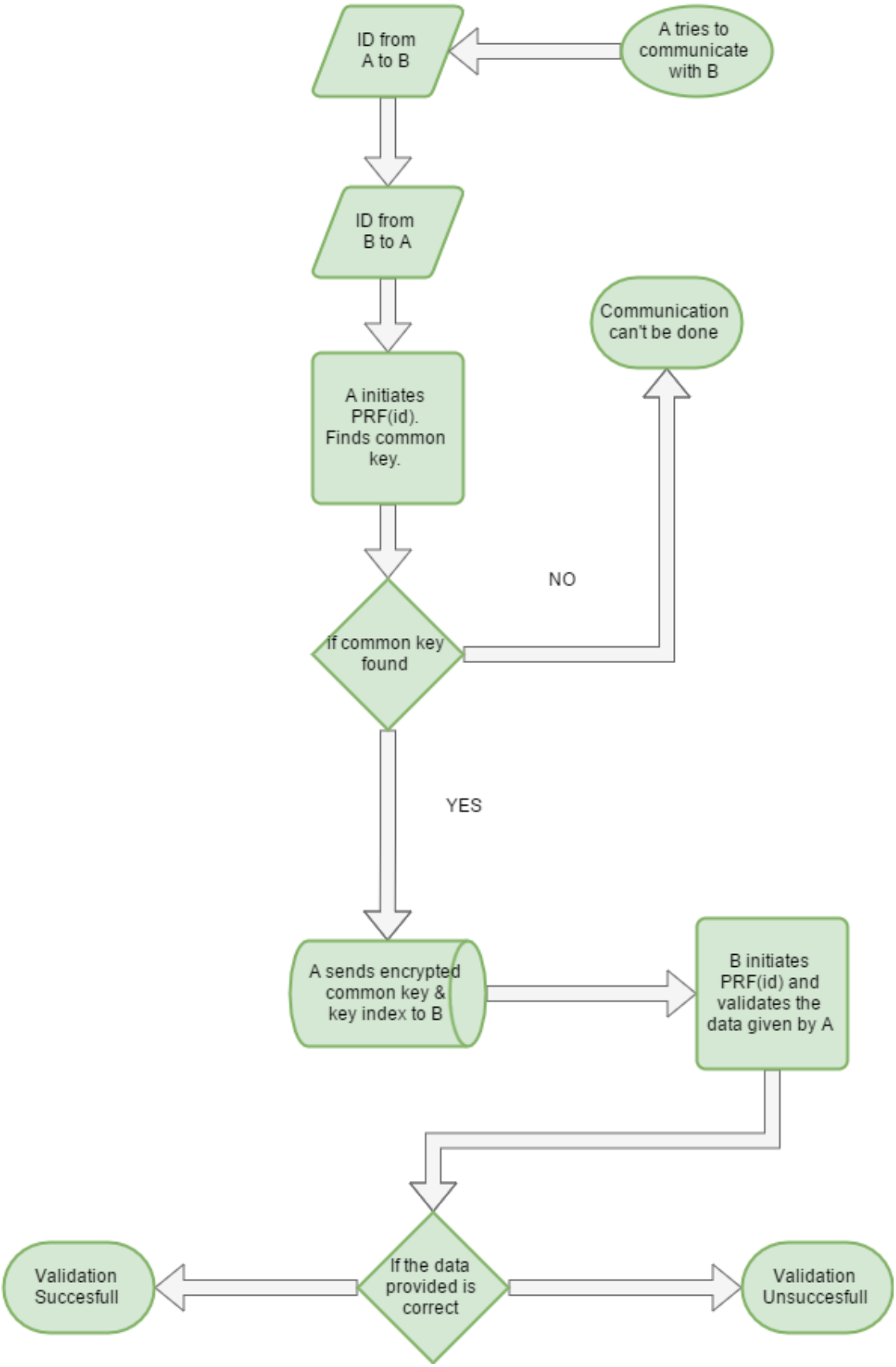


Figure 16: Algorithm Flow Chart

3.2 Problems with the existing System

The biggest challenge with key pool approach is the size of the pool. It can't be too big because that will need a lot of resources and it can't be too small because it will be easy to crack.

The other problem is that the battery of the malicious node will also get drained after a particular period of time, due to regular communication with the target node. Thus, a malicious node is required to be equipped with the more resources in terms of memory, battery power, and processing power.

As an alternative, once the battery of the malicious node has been drained to a certain threshold level, this malicious node can be substituted by another fresh malicious node. In doing so, the previous malicious node will have to transfer its complete information to the new node before being getting disabled.

3.3 Software Development model

The model used to develop the simulation of our application need to interactive, therefore the best approach which is suited for this kind of development, **Prototype** model is chosen.

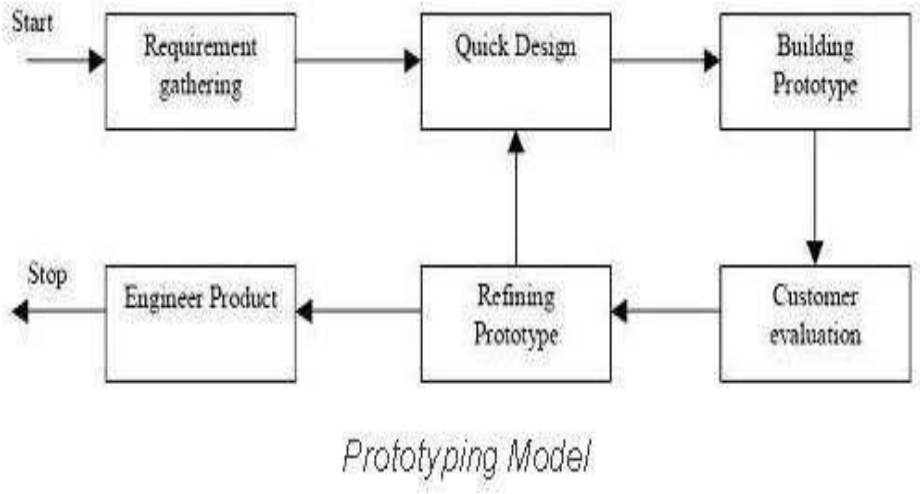


Figure 17: Prototype Model

3.4 Diagrams

3.4.1 Data Flow Diagram

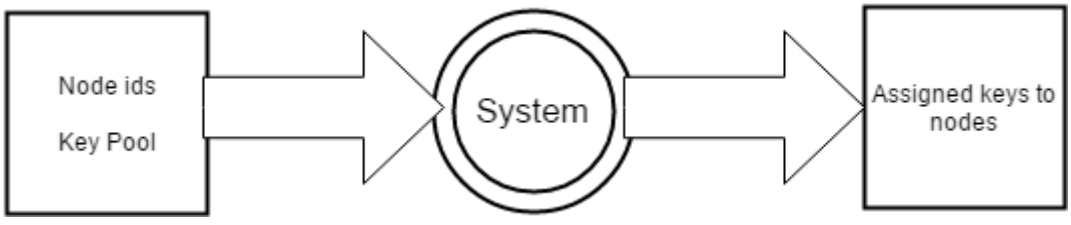


Figure 18: Zero level DFD

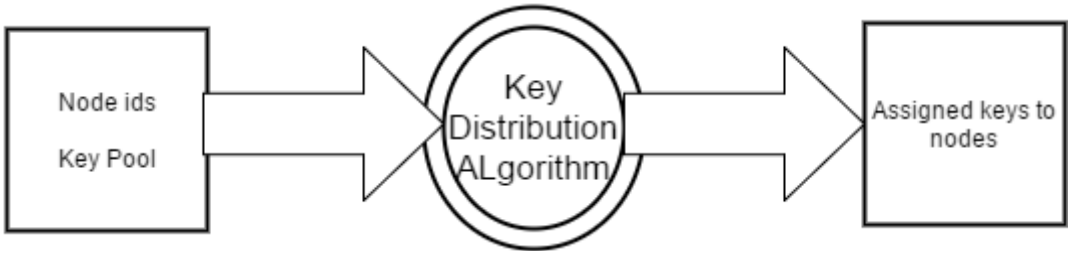


Figure 19: First level DFD

3.4.2 Use Case Diagram

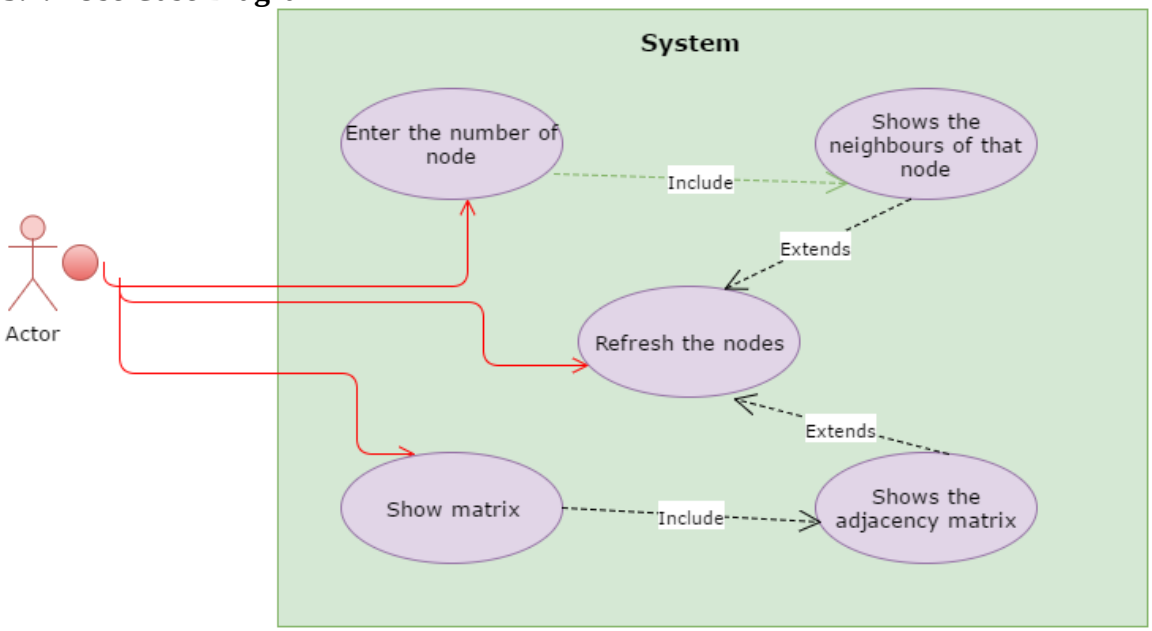


Figure 20: Use Case Diagram

3.5 HARDWARE REQUIREMENTS

The minimum requirements needed to perform operations are

- Intel I3 or above processor
- 1GB or above RAM
- Secondary memory capacity of 1 GB
- Firefox 51.0 or above, Chrome 57.0.2987.133 or above

3.6 SOFTWARE REQUIREMENTS

The software required to perform the implementation are

- Windows or Linux Operating System (Ubuntu, Fedora)
- JavaScript
- p5.js
- Eclipse / NetBeans IDE

CHAPTER 4

PERFORMANCE ANALYSIS

4.1 Complexity of Algorithm to make adjacency matrix.

- Time Complexity: O (n^2)
- As for every n nodes, we will check (n-1) nodes, whether they can communicate or not, so it will be O (n^2).

4.2 Application

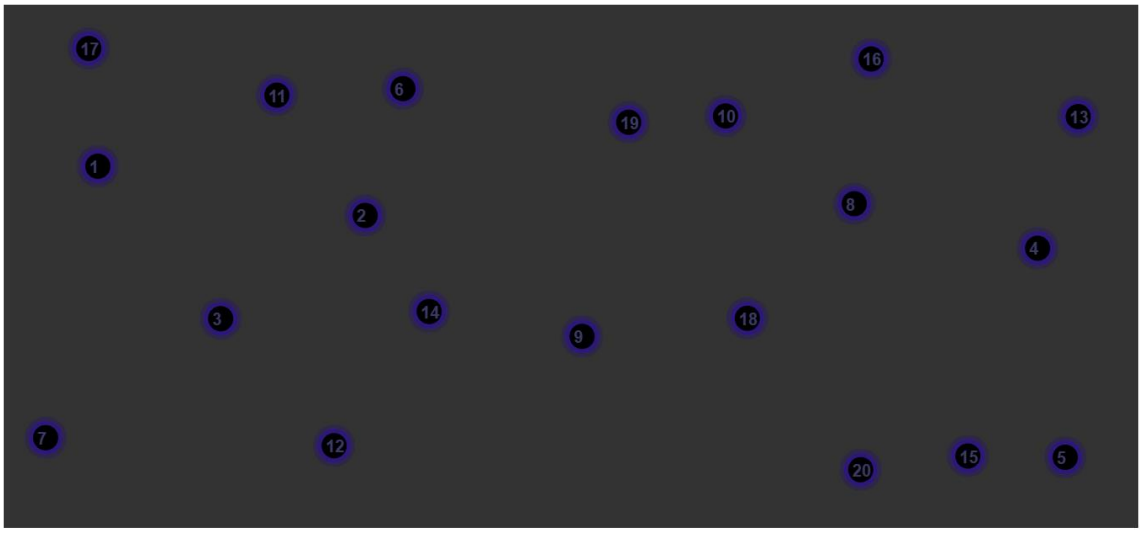


Figure 20: The Application

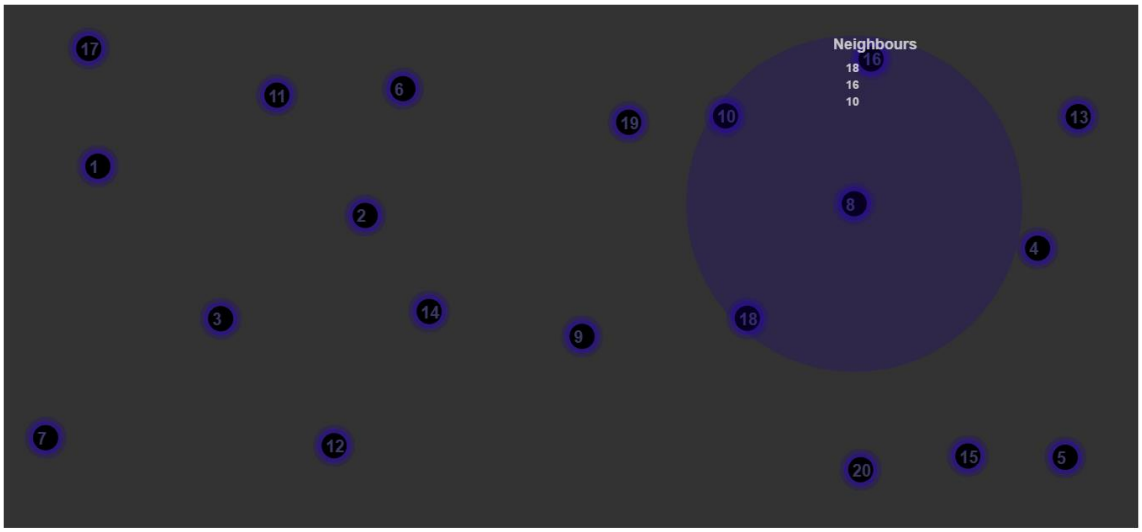
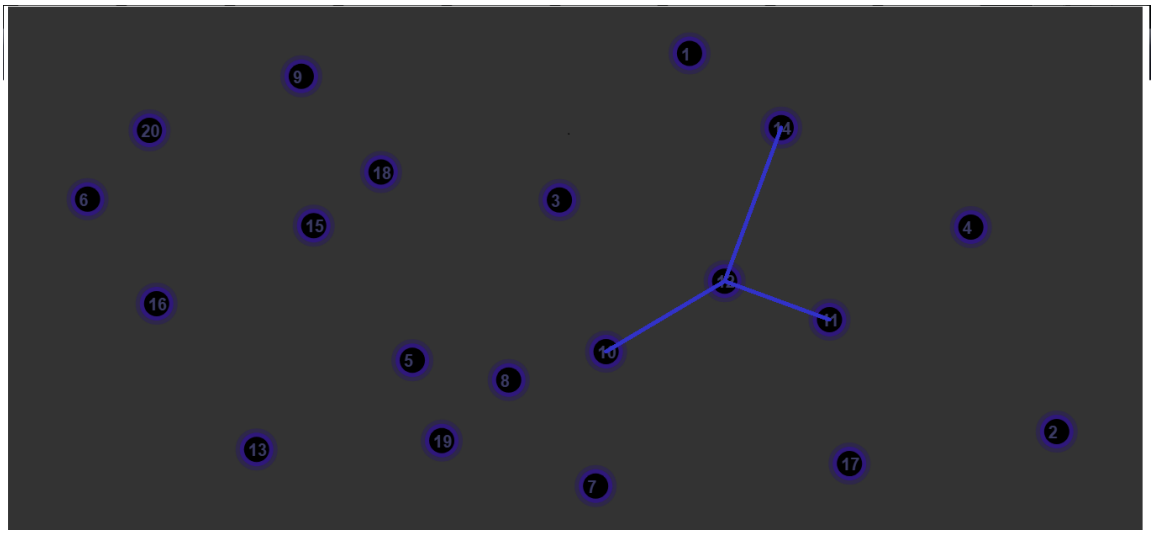
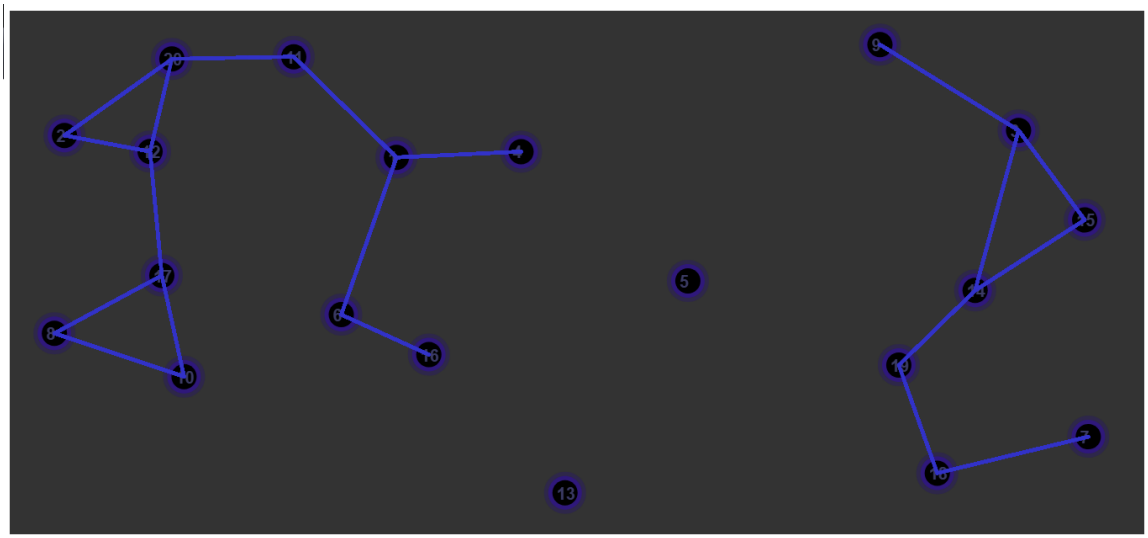


Figure 21: Selecting a node to take a look at the nodes in its range





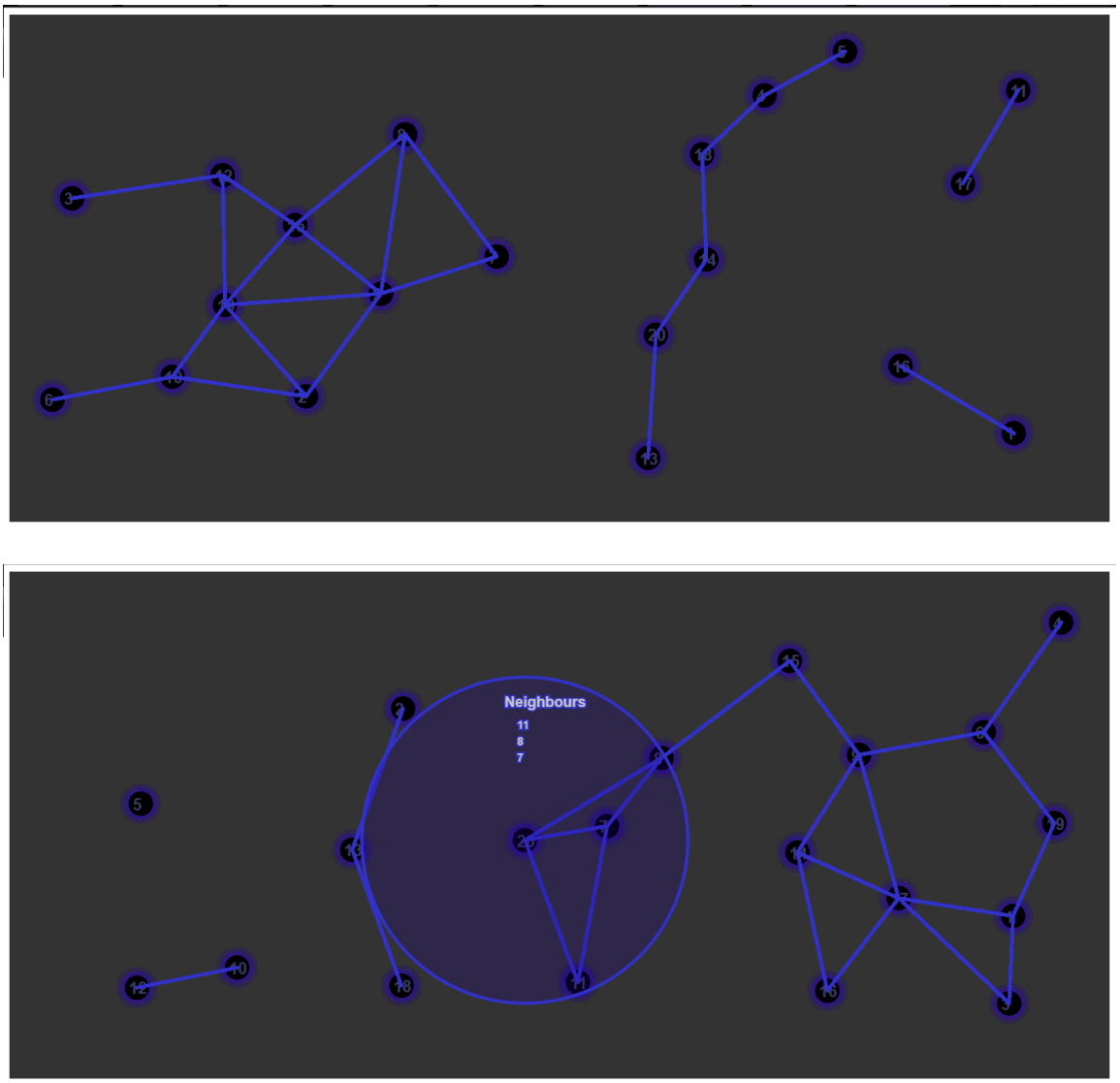


Figure 22 a-d: a-c shows how two nodes communicate with each other. D shows communication between all nodes with reference to nodes range

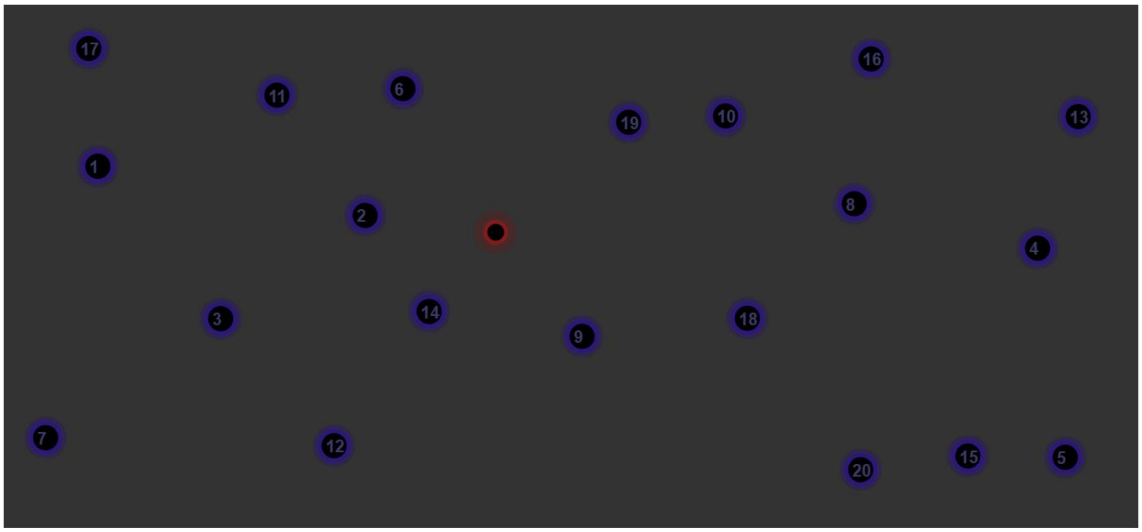


Figure 23: Malicious node's appearance

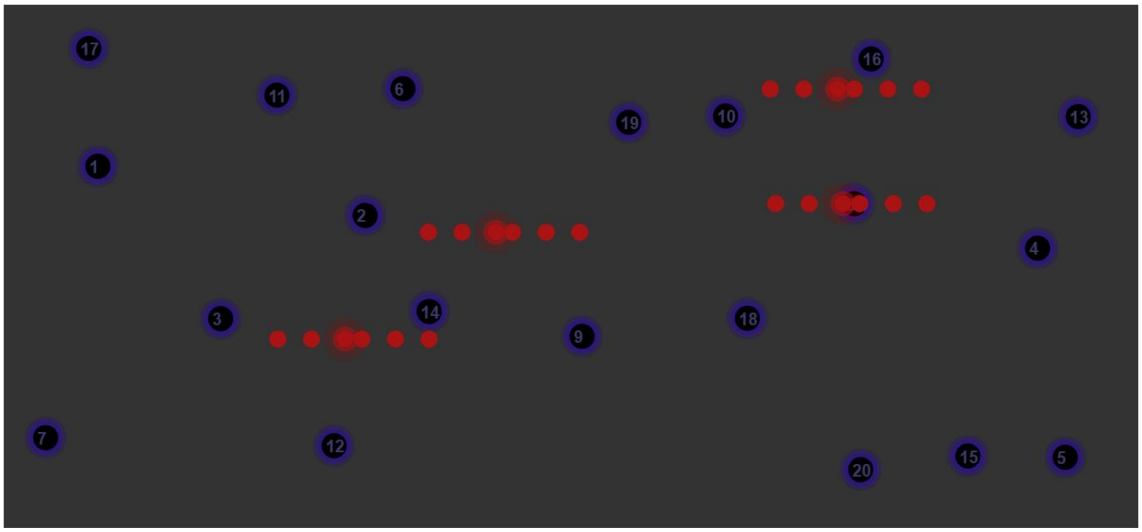


Figure 24: Multiple nodes

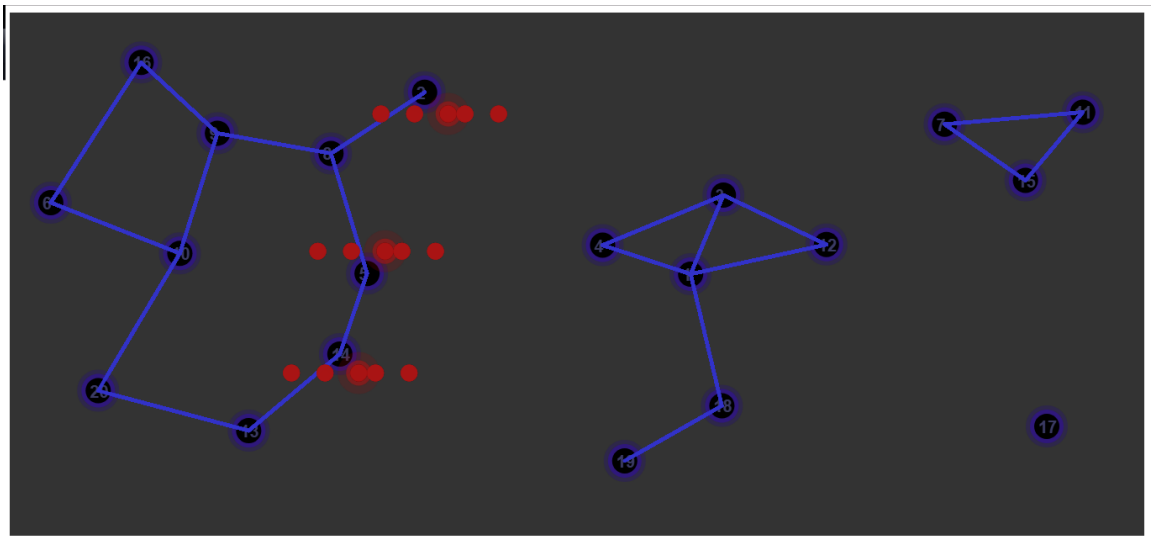


Figure 25: Multiple nodes with communication

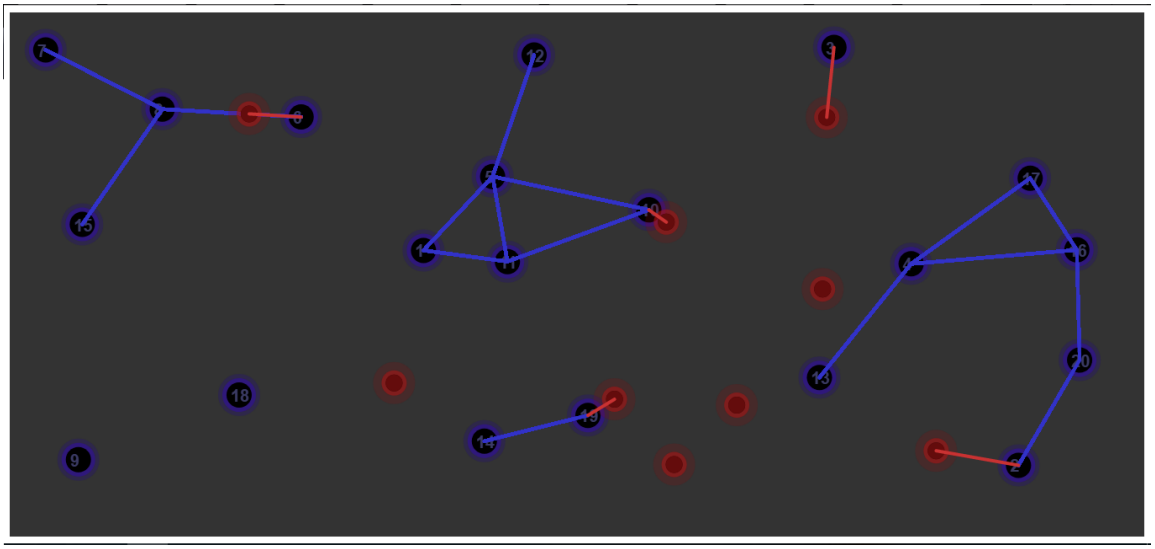


Figure 26: Multiple malicious nodes attack

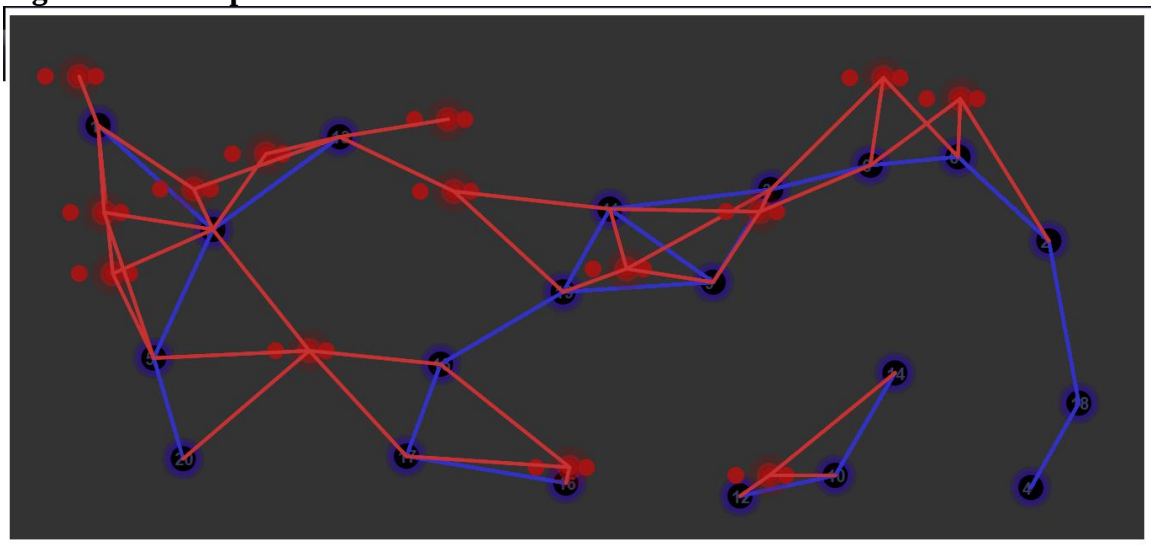


Figure 27: Multiple malicious nodes attack with multiple Sybil nodes

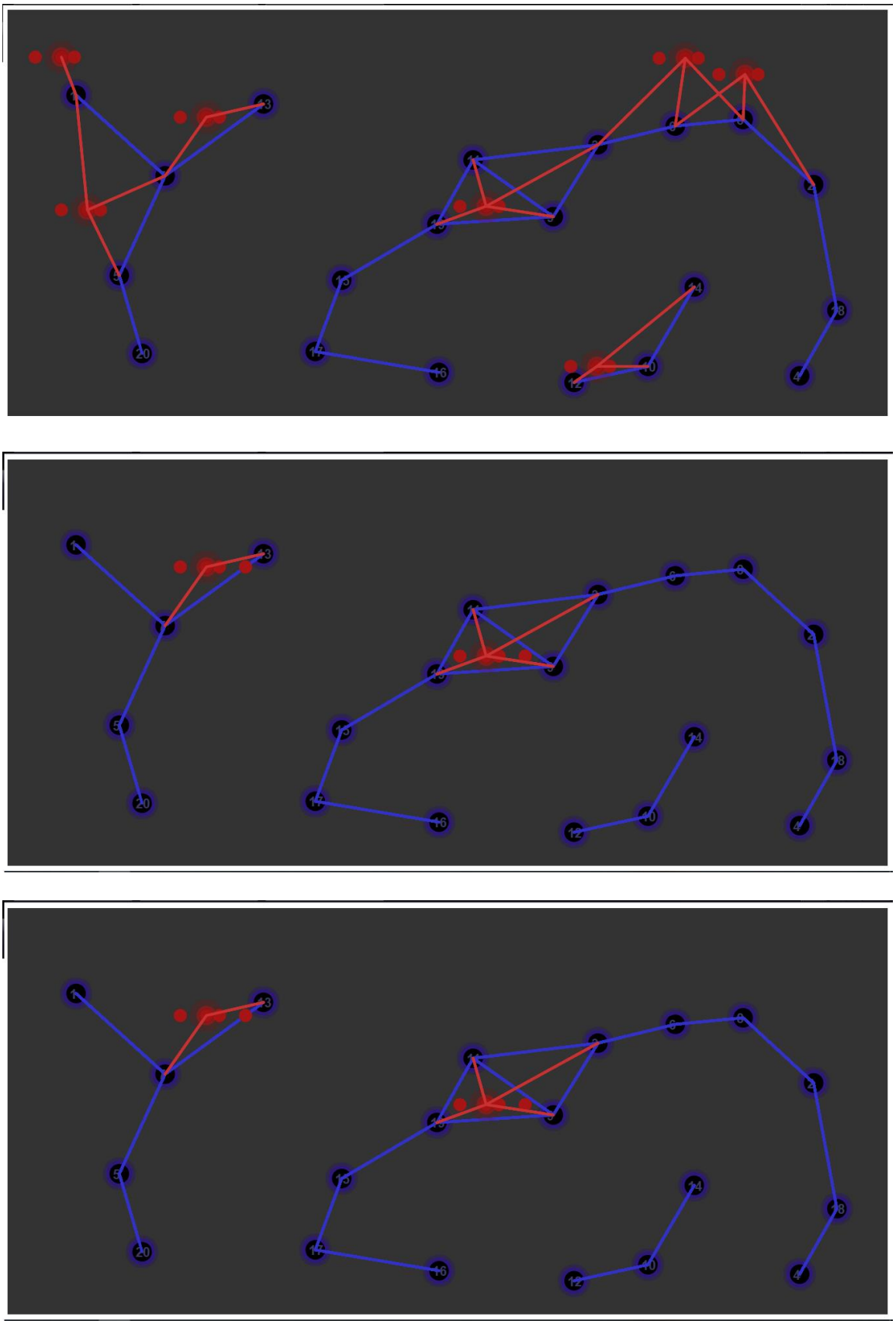


Figure 28 a-c: Defense of sensor network against Sybil attack

4.2.6 Output (Adjacency matrix)

The attacker will see the Adjacency matrix and will attack the node having the highest degree.

In the Adjacency matrix, we will show 1 for the nodes which are adjacent and 0 if they are not adjacent.

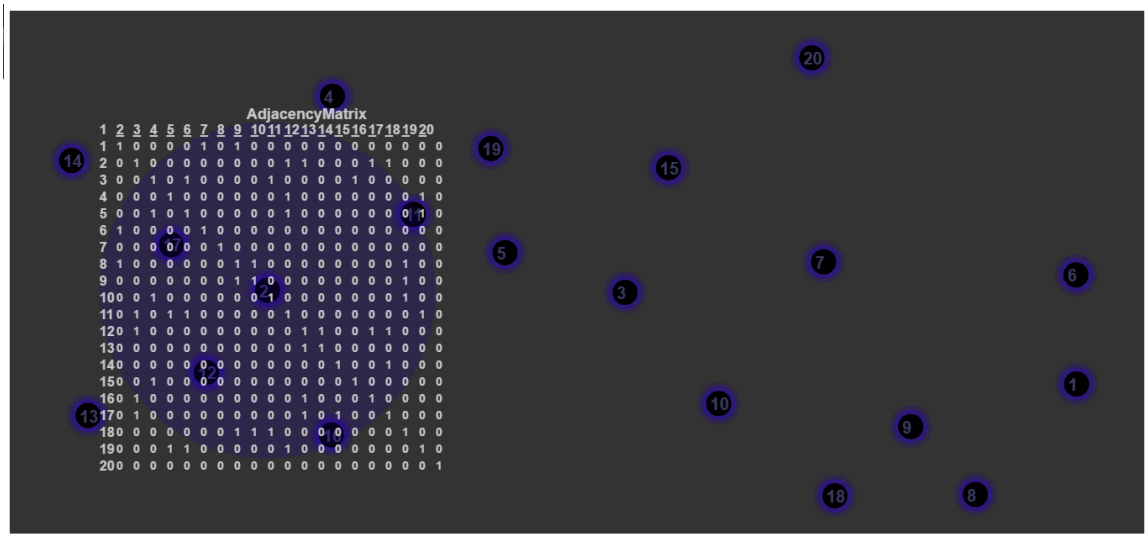


Figure 29: Adjacency matrix

CHAPTER 5

Conclusion

Although the Random key pre-distribution scheme is quite successful in preventing the Sybil attack it has its advantages and flaws. Since it only needs a pseudo-random function and a key-pool it accomplishes the requirement of it not being resource intensive and takes less space in memory. Despite this it also provides a fair amount of security to its users. Its greatest strength lies in the fact that it can also be implemented in devices with low power requirement. Its main flaw lies in the fact that the number of keys that is needed by the key-pool must be optimum i.e. it must neither be so large that the nodes have a tough time getting common keys between them and also it must not be small enough that it can be easily broken by the malicious node gathering data from the network. Further research on this value of number of keys in a key-pool per node with respect to the number of nodes which need to communicate is needed. Furthermore, there is also a need to devise methods so that the malicious node cannot construct its key-pool just by collecting the data. One of the things that can be the most effective approach with this scheme is that since it is not resource intensive it can be used in conjunction with other schemes to provide even more security. Overall, for this scheme to be practical and actually deployed in the field with other sensor nodes there is a need of more research.

REFERENCES

[1] Random Key Predistribution Schemes for Sensor Networks by Haowen Chan, Adrian Perrig, and Dawn Song

[2]A Key-Management Scheme for Distributed Sensor Networks by Laurent Eschenauer and Virgil D. Gligor

[3] Alex Hinds, Michael Ngulube, Shaoying Zhu, and Hussain Al-Aqrabi "A Review of Routing Protocols for Mobile Ad-Hoc Networks (MANET)", International Journal of Information and Education Technology, Vol. 3, No. 1, February 2013

[4] V.Preetha and Dr K.Chitra, "Clustering & Cluster Head Selection Techniques in Mobile Adhoc Networks", International Journal of Innovative Research in Computer and Communication Engineering, Vol. 2, Issue 7, July 2014

[5] Amol Vasudeva and Manu Sood, "SYBIL ATTACK ON LOWEST ID CLUSTERING ALGORITHM IN THE MOBILE AD HOC NETWORK", International Journal of Network Security & Its Applications (IJNSA), Vol.4, No.5, September 2012

[6] C. -K. Toh, (2002), "Ad Hoc Mobile Wireless Networks: Protocols and Systems", *Prentice Hall, PTR.*

[7] E. M. Royer and C. -K. Toh, (1999), "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks," *IEEE Personal Communications*.

[8] J. R Douceur, (2002), "The Sybil Attack", IPTPS '01: Revised Papers from theFirst International Workshop on Peer-to-Peer Systems, pp. 251–260, SpringerVerlag, London, UK.

[9] J. Newsome, E. Shi, D. Song, and A. Perrig, (2004), "The Sybil Attack in Sensor Networks:Analysis & Defenses ", IPSN '04. *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pp. 259–268, ACM, Berkeley, California, USA.

Appendices

Source Code

Index.html

```
<html>
<head>
  <script src="p5\p5-zip\p5.js"></script>
  <script src="Sensor.js"></script>
  <script src="main.js"></script>
  <script src="circle.js"></script>
  <script src="sybil.js"></script>
</head>
<body>
</body>
</html>
```

Main.js

```
var snodes=[];//array of sybil nodes
var sensors=[];//creating a sensors array
var circles=[];//array of circles
var flag2=0;//show wheter clicked or not
var flag=1;//flag to know whether show or hide
var count=0;//counter to check how many times clicked
var matrix=new Array(20);//creating a 2d array
var no;//this variable shows the node which is clicked
var du=0;//how many times the sybil node is duplicated
var conn=0;//connect all nodes
for(var i=0;i<20;i++)
{
```

```
  matrix[i]=new Array(20);
  for(var j=0;j<20;j++)
  {
    matrix[i][j]=0;//initializing the array
  }
}
function setup()
{
  var c;//started
  c=createCanvas(1350,622);
  for(var i=0;i<20;i++)
  {
    var test=0;//variable for checking collision
    var s=new sensor(random(40,1300),random(40,580));//new sensor node created
    for(var j=0;j<sensors.length;j++)//testing new functionality for distinct nodes
    {
      console.log("checking collision");
      if(Math.sqrt(Math.pow(s.x1-sensors[j].x1,2)+Math.pow(s.y1-sensors[j].y1,2))<100)
      {
        console.log("collision occured");
        i--;
        test=1;//collision occured
        break;
      }
    }
    if(test==0)//if there was no clash
    {
      sensors.push(s);//sensor object pushed into sensors array
```

```
    }
  }
  nmatrix(matrix);//creating a matrix of all the neighbours
}

function draw()//drawing on the screen
{
  background(51);
  for(var i=0;i<20;i++)
  {
    sensors[i].colours();
    sensors[i].showid(i);
    if(conn==1)//if right key is pressed
    {
      sensors[i].showline(matrix,sensors,conn);//just for showing communication
    }
  }
  showarea();
  for(var i=0;i<snodes.length;i++)
  {
    snodes[i].create();
    for(var j=0;j<du;j++)
    {
      snodes[i].duplicate(du);
      snodes[i].showline(sensors);
    }
  }
}
```

```
function nmatrix(matrix)//stores all position of the nodes in matrix array
{
  for(var i=0;i<20;i++)
  {
    for(var j=0;j<20;j++)
    {
       if(Math.sqrt(Math.pow(sensors[i].x1-sensors[j].x1,2)+Math.pow(sensors[i].y1-
sensors[j].y1,2))<200)//range of the sensor
      {
        matrix[i][j]=1;//neighbour found
      }
    }
  }
}
function mousePressed()
{
  count++;
  flag2=1;//we are ready to draw a circle
  if(flag==1)
  {
  for(var i=0;i<20;i++)
    {
     if(abs(mouseX-sensors[i].x1)<=40      &&      abs(mouseY-sensors[i].y1)<=40)//checking
where we clicked
      {
         no=i;//node saved in no
         var ccreate=new circ(sensors[no].x1,sensors[no].y1);
         circles.push(ccreate);
      }
```

```
    }
  }
  else
  {
    circles.pop();//deleting the already created
  }
}
function showarea()//this function shows the area which is affected
{
  if(flag2==1)
  {
    if(count%2==0)
    {
  if(flag==0)//already showing
    {
      flag=1;
    }
  }
  else//need to show
    {
      //pushing
      circles[0].show1();
      //circles[0].show2(matrix,no);//showing the neighbours
      circles[0].show2(matrix,no);//checking the matrix
      flag=0;
    }
  }
}
```

```
function keyPressed()
{
 if(keyCode==UP_ARROW)
 {
  var s=new sybil();
  snodes.push(s);
 }
 if(keyCode==DOWN_ARROW)
 {
  du++;
  console.log(du);
 }
 if(keyCode==39)//right arrow key
 {
  conn=1;
 }
 if(keyCode==68)
 {
    for(var i=0;i<snodes.length;i++)
    {
      snodes.pop();
    }
  }
 }
```

Circle.js

```
var snodes=[];//array of sybil nodes
var sensors=[];//creating a sensors array
var circles=[];//array of circles
```

```
var flag2=0;//show wheter clicked or not
var flag=1;//flag to know whether show or hide
var count=0;//counter to check how many times clicked
var matrix=new Array(20);//creating a 2d array
var no;//this variable shows the node which is clicked
var du=0;//how many times the sybil node is duplicated
var conn=0;//connect all nodes
for(var i=0;i<20;i++)
{
 matrix[i]=new Array(20);
 for(var j=0;j<20;j++)
 {
  matrix[i][j]=0;//initializing the array
 }
}
function setup()
{
 var c;//started
 c=createCanvas(1350,622);
 for(var i=0;i<20;i++)
 {
  var test=0;//variable for checking collision
  var s=new sensor(random(40,1300),random(40,580));//new sensor node created
  for(var j=0;j<sensors.length;j++)//testing new functionality for distinct nodes
  {
   console.log("checking collision");
   if(Math.sqrt(Math.pow(s.x1-sensors[j].x1,2)+Math.pow(s.y1-sensors[j].y1,2))<100)
   {
```

```
        console.log("collision occured");
        i--;
        test=1;//collision occured
        break;
      }
    }
    if(test==0)//if there was no clash
    {
      sensors.push(s);//sensor object pushed into sensors array
    }
  }
  nmatrix(matrix);//creating a matrix of all the neighbours
}

function draw()//drawing on the screen
{
  background(51);
  for(var i=0;i<20;i++)
  {
    sensors[i].colours();
    sensors[i].showid(i);
    if(conn==1)//if right key is pressed
    {
      sensors[i].showline(matrix,sensors,conn);//just for showing communication
    }
  }
  showarea();
  for(var i=0;i<snodes.length;i++)
```

```
  {
   snodes[i].create();
   for(var j=0;j<du;j++)
   {
    snodes[i].duplicate(du);
    snodes[i].showline(sensors);
   }
  }
}
function nmatrix(matrix)//stores all position of the nodes in matrix array
{
  for(var i=0;i<20;i++)
  {
   for(var j=0;j<20;j++)
   {
      if(Math.sqrt(Math.pow(sensors[i].x1-sensors[j].x1,2)+Math.pow(sensors[i].y1-
sensors[j].y1,2))<200)//range of the sensor
    {
     matrix[i][j]=1;//neighbour found
    }
   }
  }
}
function mousePressed()
{
  count++;
  flag2=1;//we are ready to draw a circle
  if(flag==1)
```

```
 {
 for(var i=0;i<20;i++)
  {
   if(abs(mouseX-sensors[i].x1)<=40 && abs(mouseY-sensors[i].y1)<=40)//checking where we clicked
    {
       no=i;//node saved in no
       var ccreate=new circ(sensors[no].x1,sensors[no].y1);
       circles.push(ccreate);
    }
  }
 }
 else
 {
  circles.pop();//deleting the already created
 }
}
function showarea()//this function shows the area which is affected
{
 if(flag2==1)
 {
  if(count%2==0)
  {
 if(flag==0)//already showing
  {
   flag=1;
  }
 }
```

```
  else//need to show
    {
      //pushing
      circles[0].show1();
      //circles[0].show2(matrix,no);//showing the neighbours
      circles[0].show2(matrix,no);//checking the matrix
      flag=0;
    }
  }
}
function keyPressed()
{
  if(keyCode==UP_ARROW)
  {
    var s=new sybil();
    snodes.push(s);
  }
  if(keyCode==DOWN_ARROW)
  {
    du++;
    console.log(du);
  }
  if(keyCode==39)//right arrow key
  {
    conn=1;
  }
  if(keyCode==68)
  {
```

```
    for(var i=0;i<snodes.length;i++)
    {
      snodes.pop();
    }
   }
  }
```

Sensor.js

```
function sensor(x,y)
{
 this.x1=x;//x position sensor
 this.y1=y;//y position
 this.id;//id of sensor
 this.sid;//sender id
 this.ck;//common keys
 this.msg;//sender message
 this.neighbours=[];//neighbours of this sensor
 this.keys=[];//stores all the keys
 this.colours=function()
 {
  fill(30,0,150,60);
  noStroke();
  ellipse(this.x1,this.y1,50,50);
  fill(50,0,200,100);
  noStroke();
  ellipse(this.x1,this.y1,40,40);
  var c=color('red');
  fill(0);
  noStroke();
  ellipse(this.x1,this.y1,30,30);
 }
 this.showid=function(a)
 {
  fill(150,150,255,100);
  textSize(20);
  textStyle(BOLD);
  text(a+1,this.x1-10,this.y1-10,50,50);
 }
 this.showline=function(a,b)//test function for communication
 {
  for(var j=0;j<20;j++)
  {
  for(var i=0;i<20;i++)
   {
    if(a[j][i]==1 && i!=j)//neighbour found
     {
      stroke(50,50,200);
      strokeWeight(4);
      line(b[j].x1,b[j].y1,b[i].x1,b[i].y1);
     }
    }
   }
}
}
```

Sybil.js

```
function sybil()

{

 this.id=[];//all the id's this node contains
```

```
this.x=random(40,1200);
this.y=random(40,550);
var k=0;//count of duplicate function
this.create=function()
{
  fill(150,10,10,50);
  noStroke();
  ellipse(this.x,this.y,50,50);
  fill(170,20,20,150);
  noStroke();
  ellipse(this.x,this.y,30,30);
  fill(0);
  noStroke();
  ellipse(this.x,this.y,20,20);
}
this.duplicate=function(du)//duplicates the sybil node
{
  console.log(du);
  fill(170,20,20,150);
  noStroke();
  for(var i=0;i<du;i++)
  {
    if(i%2==0)
    {
      ellipse(this.x-i*20,this.y,20,20);
    }
    else
    {
```

```
        ellipse(this.x+i*20,this.y,20,20);
      }
    }
  }
  this.showline=function(a)
  {
    for(var i=0;i<20;i++)
    {
      if(dist(this.x,this.y,a[i].x1,a[i].y1)<200)
      {
        console.log("entred");
        stroke(200,50,50);
        strokeWeight(4);
        line(this.x,this.y,a[i].x1,a[i].y1);
      }
    }
  }
}
```