Secure Data Collection in Wireless Sensor Networks Using Randomized Dispersive Routes

CONTENTS

<table>
<thead>
<tr>
<th>Chapter No</th>
<th>Topic Name</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>List of Acronyms</td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td>List of Figures</td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1 Introduction of the system</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2 Objective of the Project</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1.3 Organization of Dissertation</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>LITERATURE SURVEY</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2.1 Introduction</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2.2 Existing of literature survey</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>SYSTEM ANALYSIS</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3.1 Existing System</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3.2 Disadvantages of existing system</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>3.3 Proposed System</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3.4 Problem Definition</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>3.5 Modules of the project</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>REQUIREMENT ANALYSIS</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>4.1 Hardware Requirements</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>4.2 Software Requirements</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>SYSTEM DESIGN</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>5.1 Data flow Diagram</td>
<td>18</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.2</td>
<td>UML Diagrams</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>TECHNOLOGY DESCRIPTION</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>SYSTEM DEVELOPMENT</td>
<td>44</td>
</tr>
<tr>
<td>8</td>
<td>EXPERIMENTAL RESULTS</td>
<td>54</td>
</tr>
<tr>
<td>9</td>
<td>TESTING</td>
<td>68</td>
</tr>
<tr>
<td>10</td>
<td>CONCLUSION AND FUTURE WORK</td>
<td>73</td>
</tr>
<tr>
<td>11</td>
<td>BIBLIOGRAPHY</td>
<td>74</td>
</tr>
<tr>
<td>Acronym Name</td>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>WSNs</td>
<td>Wireless sensor networks</td>
</tr>
<tr>
<td>2</td>
<td>DFD</td>
<td>Data Flow Diagram</td>
</tr>
<tr>
<td>3</td>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>4</td>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>5</td>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>6</td>
<td>UTF</td>
<td>Unicode Transformation Format</td>
</tr>
<tr>
<td>7</td>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>8</td>
<td>CN</td>
<td>Compromised node</td>
</tr>
<tr>
<td>9</td>
<td>DOS</td>
<td>Denial of service</td>
</tr>
<tr>
<td>10</td>
<td>PRP</td>
<td>Purely random propagation</td>
</tr>
<tr>
<td>11</td>
<td>DRP</td>
<td>Directed random propagation</td>
</tr>
<tr>
<td>12</td>
<td>NRRP</td>
<td>No repetitive random propagation</td>
</tr>
<tr>
<td>13</td>
<td>MTRP</td>
<td>Multicast tree-assisted random propagation</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure No</th>
<th>Figure Name</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5.1 Topology Construction</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>3.5.3 Select Random Path</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>4.1. Data flow Diagram</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>4.2 UML Diagrams</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>6.2.1 Jsp Processing</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>8.4.1 Levels of Testing</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>9.1 Input form</td>
<td>53</td>
</tr>
<tr>
<td>8</td>
<td>9.2 GetNodeInfo Form</td>
<td>52</td>
</tr>
<tr>
<td>9</td>
<td>9.3 Topology Creation</td>
<td>56</td>
</tr>
<tr>
<td>10</td>
<td>9.4 Login Form</td>
<td>58</td>
</tr>
<tr>
<td>11</td>
<td>9.5 Node Name</td>
<td>59</td>
</tr>
<tr>
<td>12</td>
<td>9.6 Possible Routes</td>
<td>60</td>
</tr>
<tr>
<td>13</td>
<td>9.7 Dynamic Routing for Sending Data</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>9.8 Dynamic Routing</td>
<td>62</td>
</tr>
<tr>
<td>15</td>
<td>9.9 View Message</td>
<td>63</td>
</tr>
<tr>
<td>16</td>
<td>9.10 Message Transmission</td>
<td>64</td>
</tr>
<tr>
<td>17</td>
<td>9.6 Possible Routes</td>
<td>60</td>
</tr>
<tr>
<td>18</td>
<td>9.7 Dynamic Routing for Sending Data</td>
<td>61</td>
</tr>
<tr>
<td>19</td>
<td>9.8 Dynamic Routing</td>
<td>62</td>
</tr>
<tr>
<td>20</td>
<td>9.9 View Message</td>
<td>63</td>
</tr>
<tr>
<td>21</td>
<td>9.10 Message Transmission</td>
<td>64</td>
</tr>
<tr>
<td>22</td>
<td>9.11 Output form</td>
<td>65</td>
</tr>
</tbody>
</table>
ABSTRACT:

Various possible security threats that may be experienced by a wireless sensor network (WSN). Compromised-node and denial-of-service are two key attacks in wireless sensor networks (WSNs). In this thesis, we study routing mechanisms that circumvent (bypass) black holes formed by these attacks. We argue that existing multi-path routing approaches are vulnerable to such attacks, mainly due to their deterministic nature. So once an adversary acquires the routing algorithm, it can compute the same routes known to the source, and hence endanger all information sent over these routes. In this paper, we develop mechanisms that generate randomized multipath routes. Under our design, the routes taken by the “shares” of different packets change over time. So even if the routing algorithm becomes known to the adversary, the adversary still cannot pinpoint the routes traversed by each packet. Besides randomness, the routes generated by our mechanisms are also highly dispersive and energy-efficient, making them quite capable of bypassing black holes at low energy cost. Extensive simulations are conducted to verify the validity.

Chapter 1

Introduction

In this chapter we will discuss about the entire project document, we can learn about purpose, scope and motivation behind the development of this system, basic organization of dissertation and description of other chapters of this documentation.

1.1 Introduction of System

Compromised-node and denial-of-service are two key attacks in wireless sensor networks (WSNs). In this thesis, we study routing mechanisms that circumvent (bypass) black holes formed by these attacks. We argue that existing multi-path routing approaches are vulnerable to such attacks, mainly due to their deterministic nature. So once an adversary acquires the routing algorithm, it can compute the same routes known to the source, and hence endanger all information sent over these routes. In this paper, we develop mechanisms that generate randomized multipath routes. Under our design, the routes taken by the “shares” of different packets change over time. So even if the routing algorithm becomes known to the adversary, the adversary still cannot pinpoint the routes traversed by each packet. Besides randomness, the routes generated by our mechanisms are also highly dispersive and energy-efficient, making
them quite capable of bypassing black holes at low energy cost. Extensive simulations are conducted to verify the validity of our mechanisms.

The various possible security threats encountered in a wireless sensor network (WSN), in this Thesis, I am specifically interested in combating two types of attacks: compromised node (CN) and denial of service (DOS). In the CN attack, an adversary physically compromises a subset of nodes to eavesdrop information, whereas in the DOS attack, the adversary interferes with the normal operation of the network by actively disrupting, changing, or even paralyzing the functionality of a subset of nodes. These two attacks are similar in the sense that they both generate black holes: areas within which the adversary can either passively intercept or actively block information delivery. Due to the unattended nature of WSNs, adversaries can easily produce such black holes.

Severe CN and DOS attacks can disrupt normal data delivery between sensor nodes and the sink, or even partition the topology. A conventional cryptography-based security method cannot alone provide satisfactory solutions to these problems. This is because, by definition, once a node is compromised, the adversary can always acquire the encryption/decryption keys of that node, and thus can intercept any information passed through it. Likewise, an adversary can always perform DOS attacks (e.g., jamming) even if it does not have any knowledge of the underlying cryptosystem. One remedial solution to these attacks is to exploit the network's routing functionality. Specifically, if the locations of the black holes are known a priori, then data can be delivered over paths that circumvent (bypass) these holes, whenever possible.

In practice, due to the difficulty of acquiring such location information, the above idea is implemented in a probabilistic manner, typically through a two-step process. First, the packet is broken into M shares (i.e., components of a packet that carry partial information) using a $\delta T; MP$-threshold secret sharing mechanism such as the Shamir's algorithm. The original information can be recovered from a combination of at least T shares, but no information can be guessed from less than T shares. Second, multiple routes from the source to the destination are computed according to some multi-path routing algorithm.
These routes are node-disjoint or maximally node-disjoint subject to certain constraints (e.g., min-hop routes). The M shares are then distributed over these routes and delivered to the destination. As long as at least (or T ) shares bypass the compromised (or jammed) nodes, the adversary cannot acquire (or deny the delivery of) the original packet. We argue that three security problems exist in the above counter-attack approach. First, this approach is no longer valid if the adversary can selectively compromise or jam nodes. This is because the route computation in the above multi-path routing algorithms is deterministic in the sense that for a given topology and given source and destination nodes, the same set of routes is always computed by the routing algorithm.

As a result, once the routing algorithm becomes known to the adversary (this can be done, e.g., through memory interrogation of the compromised node), the adversary can compute the set of routes for any given source and destination. Then, the adversary can pinpoint to one particular node in each route and compromise (or jam) these nodes. Such an attack can intercept all shares of the information, rendering the above counter-attack approaches ineffective. Second, as pointed out in , actually very few node-disjoint routes can be found when the node density is moderate and the source and destination nodes are several hops apart. For example, for a node degree of 8, on average only two node-disjoint routes can be found between a source and a destination that are at least 7 hops apart. There is also 30 percent probability that no node-disjoint paths can be found between the source and the destination . The lack of enough routes significantly undermines the security performance of this multipath approach. Last, because the set of routes is computed under certain constraints, the routes may not be spatially dispersive enough to circumvent a moderate-size black hole.

We explore the potential of random dispersion for information delivery in WSNs. Depending on the type of information available to a sensor; I develop four distributed schemes for propagating information “shares”: purely random propagation (PRP), directed random propagation (DRP), no repetitive random propagation (NRRP), and multicast tree-assisted random propagation (MTRP). PRP utilizes only one-hop neighborhood information and provides baseline performance. DRP utilizes two-hop neighborhood information to improve the propagation efficiency, leading to a smaller packet interception probability. The NRRP scheme achieves a similar effect, but in a different way: it records all traversed nodes to avoid traversing
them again in the future. MTRP tries to propagate shares in the direction of the sink, making the delivery process more energy efficient.

We theoretically evaluate the goodness of these dispersive routes in terms of avoiding black holes. We conduct asymptotic analysis (i.e., assuming an infinite number of nodes) for the worst-case packet interception probability and energy efficiency under the baseline PRP scheme. Our results can be interpreted as the performance limit of PRP, and a lower-bound on the performance of the more advanced DRP, NRRP, and MTRP schemes. Our analysis helps us better to understand how security is achieved under dispersive routing. Based on this analysis, I investigate the trade-off between the random propagation parameter and the secret sharing parameter. We further optimize these parameters to minimize the end-to-end energy consumption under a given security constraint.

We conduct extensive simulations to study the performance of the proposed schemes under more realistic settings. Our simulation results are used to verify the effectiveness of our design. When the parameters are appropriately set, all four randomized schemes are shown to provide better security performance at a reasonable energy cost than their deterministic counterparts. At the same time, they do not suffer from the type of attacks faced by deterministic multipath routing.

1.2 Objective of the project

The objective of our study to propose a randomized multi-path routing algorithm that can overcome the black holes formed by Compromised-node and denial-of-service attacks. Instead of selecting paths from a pre-computed set of routes, our aim is to compute multiple paths in a randomized way each time an information packet needs to be sent, such that the set of routes taken by various shares of different packets keep changing over time. As a result, a large number of routes can be potentially generated for each source and destination. To intercept different packets, the adversary has to compromise or jam all possible routes from the source to the destination, which is practically infeasible.
The objective of this work is to explore a security enhanced dynamic routing algorithm based on distributed routing information widely supported in existing wired and wireless networks. My aim at the randomization of delivery paths for data transmission to provide considerably small path similarity (i.e., the number of common links between two delivery paths) of two consecutive transmitted packets. Security has become one of the major issues for data communication over wired and wireless networks. Different from the past work on the designs of cryptography algorithms and system infrastructures, I will propose a dynamic routing algorithm that could randomize delivery paths for data transmission. The algorithm is easy to implement and compatible with popular routing protocols, such as the Routing Information Protocol in wired networks and Destination-Sequenced Distance Vector protocol in wireless networks, without introducing extra control messages. An analytic study on the proposed algorithm is presented, and a series of simulation experiments are conducted to verify the analytic results and to show the capability of the proposed algorithm.

1.3 Organization of Dissertation

There are Eleven Chapters in the ‘Secure Data Collection in Wireless Sensor Networks Using Randomized Dispersive Routes’. Each chapter deals with the explanation regarding the following. The following each paragraph describes the each chapter of this book.

The First Chapter gives with the introduction of the project like purpose of project, objective of Project and the organization of Dissertation of the project.

The Second Chapter deals with the Literature Survey that discusses the existing system and its limitations and the need for the proposed system.

The third Chapter deals with the Review State of Art includes Requirement Analysis, disadvantages of existing system and advantages with proposed system.
The **Fourth Chapter** describes system requirements and specifications.

The **Fifth Chapter** describes gives the details about the how the system is analyzed and functionality is explained using Use Case diagrams. This contains Use Case diagrams, block diagrams, Sequence diagram and Activity diagram.

The **Sixth Chapter** describes about the technology used to develop the project.

The **Seventh Chapter** describes about the system implementation i.e. sample code of the project. Conclusion of the project to further use in the future.

The **Eighth Chapter** describes the types of testing and type of testing followed while testing the application and it contains both Class and System testing. It includes Graphical user interface of the project those show the complete idea of executing the project in step by step manner.

The **Tenth Chapter** describes about Conclusion of the project to further use in the future.

The **Last Chapter** presents the references from where we got the needed information about everything presented in the above chapters.

---

**Chapter 2**

**Literature Survey**

**2.1 Introduction**

Of the various possible security threats that may be experienced by a wireless sensor network (WSN), in this thesis we are specifically interested in combating two types of attacks: the compromised-node (CN) attack and the denial-of-service (DOS) attack.

The CN attack refers to the situation when an adversary physically compromises a subset of nodes to eavesdrop information, whereas in the DOS attack, the adversary interferes with the normal operation of the WSN by actively disrupting, changing, or even destroying the functionality of a subset of nodes in the system. These two attacks are similar in the sense that they both generate black holes: areas within which the adversary can either passively intercept or actively block information delivery.
Due to the unattended nature of WSNs, adversaries can easily produce such black holes. Severe CN and DOS attacks can disrupt normal data delivery between sensor nodes and the sink, or even partition the topology. A conventional cryptography-based security method cannot alone provide satisfactory solutions to these problems. This is because, by definition, once a node is compromised, the adversary can always acquire the encryption/decryption keys of that node, and thus can intercept any information passed through it. At the same time, an adversary can always perform certain form of DOS attack (e.g., jamming) even if it does not have any knowledge of the crypto-system used in the WSN.

One remedial solution to these attacks is to exploit the network’s routing functionality. Specifically, if the locations of the black holes formed by the compromised (or jammed) nodes are known a priori, then information can be delivered over paths that circumvent (bypass) these holes, whenever possible. In practice, due to the difficulty of acquiring such location information, the above idea is implemented in a probabilistic manner, typically through a two-step process: secret sharing and multi-path routing. First, an information (e.g., a packet) is broken into $M$ shares (i.e., components of a packet that carry partial information) using a $(T,M)$-threshold secret-sharing mechanism such as the Shamir’s algorithm.

The original information can be recovered from a combination of at least $T$ shares, but no information can be guessed from less than $T$ shares. Then, multiple routes from the source to the destination are computed according to some multi-path routing algorithm. These routes are node-disjoint or maximal node-disjoint subject to certain constraints (e.g., minhop routes). The $M$ shares are then distributed across these routes and delivered to the destination, following different paths. As long as at least $M - T + 1$ (or $T$) shares bypass the compromised (or jammed) nodes, the adversary cannot acquire (or deny the delivery of) the original information packet.

We argue that three security problems exist in the above counter-attack approach. First, this approach is no longer valid if the adversary can selectively compromise or jam nodes. This is because the route computation in the above multipath routing algorithms is deterministic in the sense that for a fixed topology, a fixed set of routes are always computed by the routing algorithm for given source and destination. Therefore, even if the shares can be distributed over different routes, overall they are always delivered over the same set of routes that are computable by the algorithm. As a result, once the routing algorithm becomes open to the adversary (this can
be done, e.g., through a memory interrogation of the compromised nodes), the adversary can by itself compute the set of routes for any given source and destination.

Then the adversary can pinpoint to one particular node in each route and compromise (or jam) these nodes. Such an attack can intercept all shares of the information, rendering the above counter-attack approaches ineffective. Second, as pointed out in, actually very few node-disjoint routes can be found when node density is moderate and source and destination nodes are several hops apart. For example, for a node degree of 8, on average only two node-disjoint routes can be found between a source and a destination that are at least 7 hops apart. There is also a 30% possibility that no node-disjoint paths can be found between the source and the destination. The lack of enough routes significantly undermines the security performance of this multipath approach. Last, even worse, because the set of routes is computed under certain constraints, the routes may not be spatially dispersive enough to circumvent a moderate-sized black hole.

2.2 Existing of literature survey

We consider a three-phase approach for secure information delivery in a WSN: secret sharing of information, randomized propagation of each information share, and normal routing (e.g., min-hop routing) toward the sink. More specifically, when a sensor node wants to send a packet to the sink, it first breaks the packet into M shares, according to a \((T, M)\)-threshold secret sharing algorithm, e.g., Shamir’s algorithm. Each share is then transmitted to some randomly selected neighbor. That neighbor will continue to relay the share it has received to other randomly selected neighbors, and so on. In each share, there is a TTL field, whose initial value is set by the source node to control the total number of random relays. After each relay, the TTL field is reduced by 1. When the TTL value reaches 0, the last node to receive this share begins to route it toward the sink using min-hop routing. Once the sink collects at least T shares, it can reconstruct the original packet. No information can be recovered from less than T shares.
The effect of route depressiveness on bypassing black holes. Where the dotted circles represent the ranges the secret shares can be propagated to in the random propagation phase. A larger dotted circle implies that the resulting routes are geographically more dispersive, it is clear that the routes of higher depressiveness are more capable of avoiding the black hole. Clearly, the random propagation phase is the key component that dictates the security and energy performance of the entire mechanism.

**Random Propagation of Information Shares**

To diversify routes, an ideal random propagation algorithm would propagate shares as dispersively as possible. Typically, this means propagating the shares farther from their source. At the same time, it is highly desirable to have an energy-efficient propagation, which calls for limiting the number of randomly propagated hops. The challenge here lies in the random and distributed nature of the propagation: a share may be sent one hop farther from its source in a given step, but may be sent back closer to the source in the next step, wasting both steps from a security standpoint. To tackle this issue, some control needs to be imposed on the random propagation process.

**Purely Random Propagation (Baseline Scheme)**

In PRP, shares are propagated based on one-hop neighborhood information. More specifically, a sensor node maintains a neighbor list, which contains the ids of all nodes within its transmission range. When a source node wants to send shares to the sink, it includes a TTL of initial value $N$ in each share. It then randomly selects a neighbor for each share, and unicasts the share to that neighbor. After receiving the share, the neighbor first decrements the TTL. If the new TTL is greater than 0, the neighbor randomly picks a node from its neighbor list (this node cannot be the source node) and relays the share to it, and so on. When the TTL reaches 0, the final node receiving this share stops the random propagation of this share, and starts routing it toward the sink using normal min-hop routing. The WANDERER scheme [2] is a special case of PRP with $N = \infty$. 
The main drawback of PRP is that its propagation efficiency can be low, because a share may be propagated back and forth multiple times between neighboring hops. As shown in the analysis and simulations in subsequent sections, increasing the TTL value does not fully address this problem. This is because the random propagation process reaches steady state under a large TTL, and its distribution will no longer change even if the TTL becomes larger.

Chapter 3
SYSTEM ANALYSIS

3.1 Existing System

SPREAD algorithm attempts to find multiple most-secure and node-disjoint paths. The security of a path is defined as the likelihood of node compromise along that path, and is labeled as the weight in path selection. A modified Dijkstra algorithm is used to iteratively find the top-K most secure node-disjoint paths. The H-SPREAD algorithm improves upon SPREAD by simultaneously accounting for both security and reliability requirements. Distributed Bound-Control and Lex-Control algorithms, which computes multiple paths, respectively, in such a way that the performance degradation (e.g., throughput loss) is minimized when a single-link attack or a multi-link attack happens, respectively. Flooding is the most common randomized multi-
path routing mechanism. As a result, every node in the network receives the packet and retransmits it once. To reduce unnecessary retransmissions and improve energy efficiency, the Gossiping algorithm was proposed as a form of controlled flooding, whereby a node retransmits packets according to a pre-assigned probability.

Parametric Gossiping was proposed in to overcome the percolation behavior by relating a node’s retransmission probability to its hop count from either the destination or the source. A special form of Gossiping is the Wanderer algorithm, whereby a node retransmits the packet to one randomly picked neighbor. When used to counter compromised-node attacks, flooding, Gossiping, and parametric Gossiping actually help the adversary intercept the packet, because multiple copies of a secret share are dispersed to many nodes.

3.2 Disadvantages of existing system

- Existing randomized multi-path routing algorithms in WSNs have not been designed with security considerations in mind, largely due to their low energy efficiency.
- Multi-path routing mechanism, Gossiping algorithm has a percolation behavior, in that for a given retransmission probability, either very few nodes receive the packet, or almost all nodes receive it.
- The Wanderer algorithm has poor energy performance, because it results in long paths.

3.3 Proposed System

Our proposed solution is to establish a randomized multi-path routing algorithm that can overcome the black holes formed by Compromised-node and denial-of-service attacks. Instead of selecting paths from a pre-computed set of routes, our aim is to compute multiple paths in a randomized way each time an information packet needs to be sent, such that the set of routes taken by various shares of different packets keep changing over time. As a result, a large number of routes can be potentially generated for each source and destination. To intercept different packets, the adversary has to compromise or jam all possible routes from the source to the destination, which is practically infeasible.

We propose a randomized multipath routing algorithm that can overcome the above problems. In this algorithm, multiple paths are computed in a randomized way each time an information packet needs to be sent, such that the set of routes taken by various shares of
different packets keep changing over time. As a result, a large number of routes can be potentially generated for each source and destination. To intercept different packets, the adversary has to compromise or jam all possible routes from the source to the destination, which is practically not possible.

Because routes are now randomly generated, they may no longer be node-disjoint. However, the algorithm ensures that the randomly generated routes are as dispersive as possible, i.e., the routes are geographically separated as far as possible such that they have high likelihood of not simultaneously passing through a black hole. Considering the stringent constraint on energy consumption in WSNs, the main challenge in our design is to generate highly dispersive random routes at low energy cost. As explained later, such a challenge is not trivial. A naive algorithm of generating random routes, such as Wanderer schemes (a pure random-walk algorithm), only leads to long paths (containing many hops, and therefore, consuming lots of energy) without achieving good dispersiveness. Due to security considerations, i also require that the route computation be implemented in a distributed way, such that the final route represents the aggregate decision of all the nodes participating in the route selection. As a result, a small number of colluding/compromised nodes cannot dominate the selection result. In addition, for efficiency purposes, i also require that the randomized route selection algorithm only incurs a small amount of communication overhead.

3.4 Problem Definition

Compromised node and denial of service are two key attacks in wireless sensor networks (WSNs). Our model circumvents black holes formed by these attacks. For this, i explore the potential of random dispersion for information delivery in WSNs. Depending on the type of information available to a sensor, i develop our distributed scheme for propagating information “shares” called purely random propagation (PRP). PRP utilizes only one-hop neighborhood information and provides baseline performance. To diversify routes, an ideal random propagation algorithm would propagate shares as depressively as possible.

We consider the problem of deciding the parameters for secret sharing (M) and random propagation (N) to achieve a desired security performance. To obtain the maximum protection of the information, the threshold parameter should be set as \( T = M \). Then, increasing the number
of propagation steps (N) and increasing the number of shares a packet is broken into (M) has a similar effect on reducing the message interception probability. Specifically, to achieve a given $P_{S}^{(\text{max})}$ for a packet, i could either break the packet into more shares but restrict the random propagation of these shares within a smaller range, or break the packet into fewer shares but randomly propagate these shares into a larger range. Therefore, when the security performance is concerned, a trade-off relationship exists between the parameters M and N. On the other hand, although different combinations of M and N may contribute to the same $P_{S}^{(\text{max})}$, their energy cost may be different, depending on the parameters $L_s$, $L_p$, and $q$. This motivates us to include their energy consumption into consideration when deciding the secret sharing and random propagation parameters: I can formulate an optimization problem to solve for the most energy-efficient combination of M and N subject to a given security constraint. Formally, this is given as follows:

$$
\begin{align*}
\text{minimize} & \quad Q^{(\text{PRP})}(M, N) \\
\text{subject to} & \quad P_{s}^{(\text{max})}(M, N) \leq P_{S}^{(\text{req})}, \\
& \quad 1 \leq M \leq M_{\text{max}}, \\
& \quad 1 \leq N \leq N_{\text{max}},
\end{align*}
$$

(29)

Where M and N are variables and $P_{S}^{(\text{req})}$ is the given security requirement. The upper bounds, $M_{\text{max}}$ and $N_{\text{max}}$, are dictated by practical considerations such as the hardware or energy constraints. Because the range of M and N that are of practical interest is not large, e.g., at most few of tens, the space of feasible $(M, N)$ is moderate. Thus, the optimal $(M^o, N^o)$ can be solved by the exhaustive search algorithm.

### 3.5 Modules of the project

#### 3.5.1 Topology Construction

In this module, we construct a topology structure. Here i use mesh topology because of its unstructured nature. Topology is constructed by getting the names of the nodes and the
connections among the nodes as input from the user. While getting each of the nodes, their associated port and ip address is also obtained. For successive nodes, the node to which it should be connected is also accepted from the user. While adding nodes, comparison will be done so that there would be no node duplication. Then i identify the source and the destinations.

![Flowchart](image)

**Fig 3.5.1 Topology Construction**

### 3.5.2 Randomized Multipath Routing

We achieve randomized multipath routing that can overcome the Compromised Node attack Denial of Service attack. Here multiple paths are computed in a randomized way each time an information packet needs to be sent, such that the set of routes taken by various shares of different packets. As a result, a large number of routes can be potentially generated for each source and destination. To intercept different packets, the adversary has to compromise or jam all possible routes from the source to the destination, which is practically not possible.
3.5.3 Select Random Path

Fig 3.5.3 Select Random Path

3.5.4 Pure Random Propagation

Pure Random Propagation (PRP), shares are propagated based on one-hop neighborhood information. More specifically, a sensor node maintains a neighbor list, which contains the ids of all nodes within its transmission range. When a source node wants to send data to destination, it includes a TTL of initial value N in each share. It then randomly selects a neighbor for each share, and unicasts the share to that neighbor. After receiving the share, the neighbor first decrements the TTL. If the new TTL is greater than 0, the neighbor randomly picks a node from
its neighbor list (this node cannot be the source node) and relays the share to it, and so on. When the TTL reaches 0, the final node receiving this share stops the random propagation of this share, and starts routing it toward the sink using normal min-hop routing.

3.5.5 SECURED DELIVERY OF PACKETS

In this module we can maintain the routing table; here we add one more column to maintain the packet delivery ratio. In this one we can maintain how many packets are transmitted over each path. It will be useful for to identify any path can handle number packets. I can stop transmission some amount of time period over that path. So the hacker cannot identify in which path the message is transmitted and also we can easily transmit the data securely. To reduce unnecessary retransmissions and improve energy efficiency, the Gossiping algorithm was proposed as a form of controlled flooding, whereby a node retransmits packets according to a pre-assigned probability. It is well known that the Gossiping algorithm has a percolation behavior, in that for a given retransmission probability, either very few nodes receive the packet, or almost all nodes receive it.
3.5.6 Module INPUT/OUTPUT

Module 1

Given Input:

Topology creation

Output:

Creation of nodes

Module 2

Given Input:

Destination node

Output:

Select destination

Module 3

Given Input:

Randomized multipath creation

Output:

Randomized path between given created nodes
In this chapter requirements which are necessary for the proposed system are defined. That is hardware & software technologies are identified to develop the modules of the project.

### 4.1 Hardware Requirements

- **Processor**: Any Processor above 500 MHz
- **Ram**: 128Mb.
- **Hard Disk**: 10 Gb.
- **Compact Disk**: 650 Mb.
- **Input device**: Standard Keyboard and Mouse.
- **Output device**: VGA and High Resolution Monitor.

### 4.2 Software Requirements

- **Operating System**: Windows Family.
- **Language**: JDK 1.5
- **Data Bases**: Microsoft Sql Server
- **Front End**: Java Swing
5.1 Data Flow Diagram

![Diagram of Data Flow Diagram]

Fig 5.1 Dataflow Diagram
5.2 UML Diagrams

5.2.1 Use Case Diagram

![Use Case Diagram for modules](image_url)

Fig. 5.2.1 Use Case Diagram for modules
5.2.2 Class Diagram

The diagram represents class diagram for secure data collection in wireless sensor networks using randomized dispersive routes and consisting of three classes namely TopologyConstruction, RandomPathSelection, and TransmitMessage.

Fig. 5.2.2 Class diagram for modules
5.2.3 Sequence diagram

A sequence diagram that emphasize the time ordering of messages. It consists of objects represented as rectangles with names underlined and messages represented as solid lines and time represented as vertical dashing lines.

![Sequence Diagram]

Fig. 5.2.3 Sequence Diagram
5.2.4 Collaboration Diagram

![Collaboration Diagram](image)

Fig 5.2.4 Collaboration Diagram
5.2.5 Activity Diagram

Fig 5.2.5 Activity Diagram for login form
In this chapter the technology which is necessary to develop the project has been introduced. This project has developed in JDK 6.1 with swings and for designing Rational rose is used.

### 6.1 ABOUT JAVA:

Java is a programming language originally developed by Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture.

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any supported hardware/operating-system platform. One should be able to write a program once, compile it once, and run it anywhere.

This is achieved by compiling the Java language code, not to machine code but to Java bytecode – instructions analogous to machine code but intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a JRE installed on their own machine, or in a Web browser.

Standardized libraries provide a generic way to access host specific features such as graphics, threading and networking. In some JVM versions, bytecode can be compiled to native code, either before or during program execution, resulting in faster execution.

A major benefit of using bytecode is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would, and Java suffered a reputation for poor performance. This gap has been narrowed by a number of optimization techniques introduced in the more recent JVM implementations.
One such technique, known as (just-in-time compilation) JIT, translates Java bytecode into native code the first time that code is executed, then caches it. This result in a program that starts and executes faster than pure interpreted code can, at the cost of introducing occasional compilation overhead during execution. More sophisticated VMs also use dynamic recompilation, in which the VM analyzes the behavior of the running program and selectively recompiles and optimizes parts of the program. Dynamic recompilation can achieve optimizations superior to static compilation because the dynamic compiler can base optimizations on knowledge about the runtime environment and the set of loaded classes, and can identify hot spots - parts of the program, often inner loops, that take up the most execution time. JIT compilation and dynamic recompilation allow Java programs to approach the speed of native code without losing portability.

Another technique, commonly known as static compilation, or ahead-of-time (AOT) compilation, is to compile directly into native code like a more traditional compiler. Static Java compilers translate the Java source or bytecode to native object code. This achieves good performance compared to interpretation, at the expense of portability; the output of these compilers can only be run on a single architecture. AOT could give Java something like performance, yet it is still not portable since there are no compiler directives, and all the pointers are indirect with no way to micro manage garbage collection.

Java's performance has improved substantially since the early versions, and performance of JIT compilers relative to native compilers has in some tests been shown to be quite similar. The performance of the compilers does not necessarily indicate the performance of the compiled code; only careful testing can reveal the true performance issues in any system.

One of the unique advantages of the concept of a runtime engine is that errors (exceptions) should not 'crash' the system. Moreover, in runtime engine environments such as Java there exist tools that attach to the runtime engine and every time that an exception of interest occurs they record debugging information that existed in memory at the time the exception was thrown (stack and heap values). These Automated Exception Handling tools provide 'root-cause' information for exceptions in Java programs that run in production, testing or development environments.
6.1.1 Features of Java:

Distributed

Java has an extensive library of routines for coping with TCP/IP protocols like HTTP and FTP. Java applications can open and access across the Net via URLs with the same ease as when accessing the local file system.

We have found the networking capabilities of Java to be both strong and easy to use. Anyone who has tried to do Internet programming using another language will revel. How simple Java makes onerous tasks will like opening a socket connection.

Robust

Java is intended for writing programs that must be readable in a variety ways. Java puts a lot of emphasis on early checking for possible problems, later dynamic checking, and eliminating situations that are error prone. The single biggest difference between Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data.

The Java compiler detects many problems that in other languages would only show up at runtime. As for the second point, anyone who has spent hours chasing a memory leak cost by a printer bug will be very happy with this feature of Java.

Java gives you the best of both worlds. You need not pointers for everyday constructs like string and arrays. You have the power of pointers if you need it, for example, for like lists. And you have always-complete safety, since you can never access a bad pointer or make memory allocation errors.

Secure

Java is intended to be used in networked/distributed environment toward that end; a lot of emphasis has been placed on security. Java enables the contraction of virus-free, temper-free systems.

Here is a sample of what Java’s security features are supposed to keep a Java programming from doing:
1. Overrunning the runtime stack.
2. Corrupting memory outside its own process space.
3. Reading or writing local files when invoked through a security-conscious class loaders like Web browser.

**Architecture Neutral**

The compiler generates an architecture neutral object file format - the compiled code is executable on many processors, given the presence of Java runtime system... The Java compiler does this by generating byte code instructions which have nothing to do with particular computer architecture. Rather they were designed to be both easy to any machine and easily translated into native machine code on the fly.

Twenty years ago, the UCSD Pascal system did the same thing in a commercial product and, even before that, Nicholas Worth's original implementation of Pascal used the same approach. By using bytecode, performance takes major hit. The designers of Java did an excellent job developing a byte code instruction set those workers well on today's most common computer architectures. And the codes have been designed to translate easily into actual machine instructions.

**Portable**

Unlike C and C++, they are no "implementation dependent" aspects of the specifications. The sizes of the primitive's data types are specified, as is the behavior of arithmetic on them.

For example, an int in Java is always a 32-bit integer. In C/C++, int can mean a 16-bit integer, a 32-bit integer, or any size the compiler vendor likes. The only restriction is that it must have at least as many bytes int and cannot have more bytes than a long int.

The libraries that are a part of the system define portable interfaces. For example, there is an abstract window class and implementations of it UNIX, Windows, and the Macintosh.
Interpreted

The Java interpreters can execute Java byte codes directly on any machine to which the interpreter has been ported. Since linking is a more incremental and lightweight process, the development process can be much more rapid and explanatory.

One problem is that the JDK is fairly slow at compiling your source code to the bytecode that will, ultimately, be interpreted in the current version.

High Performance

While the performance of interpreted byte code is usually more than adequate, there are situations higher performance is required. The byte code can be translated on fly into machine code for the particular CPU the application is running on.

Native code compilers for Java are not yet generally available. Instead there are just-in-time (JIT) compilers. These work by compiling the byte codes into native code once, caching the results, and then calling them again, if needed. This speeds up code once, catching the results, and calling them again, if needed. This speed up the loop tremendously since once has to do the interpretation only once. Although still slightly slower than a true native code compiler, just-in-time compilers can give you a 10-or even 20-fold speedup for some programs and will almost always be significantly faster than the Java Interpreter.

Multithreaded

In a number of ways, Java is more dynamic language than C or C++. It was designed to adapt to an evolving environment. Libraries can freely add new methods and instance variables without any effect on their clients.... In Java, finding out run time type information is straightforward.

This is an important feature in those situations where code needs to be added to a running program. A prime example is code that is downloaded from the Internet to run in browser.
6.1.2 Why Software Developers Choose Java?

Java has been tested, refined, extended, and proven by a dedicated community. And numbering more than 6.5 million developers, it's the largest and most active on the planet. With its versatility, efficiency, and portability, Java has become invaluable to developers by enabling them to:

- Write software on one platform and run it on virtually any other platform
- Create programs to run within a Web browser and Web services
- Develop server-side applications for online forums, stores, polls, HTML forms processing, and more
- Combine applications or services using the Java language to create highly customized applications or services
- Write powerful and efficient applications for mobile phones, remote processors, low-cost consumer products, and practically any other device with a digital heartbeat

6.2 About Java Server Pages (JSP):

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.

JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

Java Server Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with `<%` and end with `%`.

A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.
Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

Java Server Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- Java Server Pages are built on top of the Java Servlets API, so like Servlets; JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of J2EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

**JSP Processing:**
The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
- The JSP engine loads the JSP page from disk and converts it into servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be shown below in the following diagram:

Fig: 6.2.1 Jsp processing

6.3 About Swings:

Swing is a platform-independent, Model-View-Controller GUI framework for Java. It follows a single-threaded programming model, and possesses the following traits:
Platform independence

Swing is platform independent both in terms of its expression (Java) and its implementation (non-native universal rendering of widgets).

Extensibility

Swing is a highly partitioned architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations. In general, Swing users can extend the framework by extending existing (framework) classes and/or providing alternative implementations of core components.

Component-oriented

Swing is a component-based framework. The distinction between objects and components is a fairly subtle point: concisely, a component is a well-behaved object with a known/specifed characteristic pattern of behavior. Swing objects asynchronously fire events, have "bound" properties, and respond to a well-known set of commands (specific to the component.) Specifically, Swing components are Java Beans components, compliant with the Java Beans Component Architecture specifications.

Customizable

Given the programmatic rendering model of the Swing framework, fine control over the details of rendering of a component is possible in Swing. As a general pattern, the visual representation of a Swing component is a composition of a standard set of elements, such as a "border", "inset", decorations, etc. Typically, users will programatically customize a standard Swing component (such as a JTable) by assigning specific Borders, Colors, Backgrounds, opacities, etc., as the properties of that component. The core component will then use this property (settings) to determine the appropriate renderers to use in painting its various aspects. However, it is also completely possible to create unique GUI controls with highly customized visual representation.
Configurable

Swing's heavy reliance on runtime mechanisms and indirect composition patterns allows it to respond at runtime to fundamental changes in its settings. For example, a Swing-based application can change its look and feel at runtime. Further, users can provide their own look and feel implementation, which allows for uniform changes in the look and feel of existing Swing applications without any programmatic change to the application code.

Lightweight UI

Swing's configurability is a result of a choice not to use the native host OS's GUI controls for displaying itself. Swing "paints" its controls programmatically through the use of Java 2D APIs, rather than calling into a native user interface toolkit. Thus, a Swing component does not have a corresponding native OS GUI component, and is free to render itself in any way that is possible with the underlying graphics APIs.

However, at its core every Swing component relies on an AWT container, since (Swing's) JComponent extends (AWT's) Container. This allows Swing to plug into the host OS's GUI management framework, including the crucial device/screen mappings and user interactions, such as key presses or mouse movements. Swing simply "transposes" its own (OS agnostic) semantics over the underlying (OS specific) components. So, for example, every Swing component paints its rendition on the graphic device in response to a call to component. Paint(), which is defined in (AWT) Container. But unlike AWT components, which delegated the painting to their OS-native "heavyweight" widget, Swing components are responsible for their own rendering.

This transposition and decoupling is not merely visual, and extends to Swing's management and application of its own OS-independent semantics for events fired within its component containment hierarchies. Generally speaking, the Swing Architecture delegates the task of mapping the various flavors of OS GUI semantics onto a simple, but generalized, pattern to the AWT container. Building on that generalized platform, it establishes its own rich and complex GUI semantics in the form of the JComponent model. A review of the source of
Container. Java and JComponent.java classes are recommended for further insights into the nature of the interface between Swing's lightweight components and AWT's heavyweight widgets.

**Loosely-Coupled/MVC**

The Swing library makes heavy use of the Model/View/Controller software design pattern, which conceptually decouples the data being viewed from the user interface controls through which it is viewed. Because of this, most Swing components have associated *models* (which are specified in terms of Java interfaces), and the programmer can use various default implementations or provide their own. The framework provides default implementations of model interfaces for all of its concrete components.

Typically, Swing component model objects are responsible for providing a concise interface defining events fired, and accessible properties for the (conceptual) data model for use by the associated JComponent. Given that the overall MVC pattern is a loosely-coupled collaborative object relationship pattern, the model provides the programmatic means for attaching event listeners to the data model object. Typically, these events are model centric (ex: a "row inserted" event in a table model) and are mapped by the JComponent specialization into a meaningful event for the GUI component.

For example, the JTable has a model calledTableModel that describes an interface for how a table would access tabular data. A default implementation of this operates on a two-dimensional array.

The view component of a Swing JComponent is the object used to graphically "represent" the conceptual GUI control. A distinction of Swing, as a GUI framework, is in its reliance on programmaticall-rendered GUI controls (as opposed to the use of the native host OS's GUI controls). This distinction is a source of complications when mixing AWT controls, which use native controls, with Swing controls in a GUI.

Finally, in terms of visual composition and management, Swing favors relative layouts (which specify the positional relationships between components) as opposed to absolute layouts (which specify the exact location and size of components). This bias towards "fluid" visual
ordering is due to its origins in the applet operating environment that framed the design and development of the original Java GUI toolkit. (Conceptually, this view of the layout management is quite similar to that which informs the rendering of HTML content in browsers, and addresses the same set of concerns that motivated the former.)

**Look and feel**

Swing allows one to specialize the look and feel of widgets, by modifying the default (via runtime parameters), deriving from an existing one, by creating one from scratch, or, beginning with J2SE 5.0, by using the skinnable synth Look and Feel (see [Synth Look and Feel](#)), which is configured with an XML property file. The look and feel can be changed at runtime, and early demonstrations of Swing frequently provided a way to do this.

### 6.4 Relationship to AWT

Since early versions of Java, a portion of the Abstract Window Toolkit (AWT) has provided platform-independent APIs for user interface components. In AWT, each component is rendered and controlled by a native peer component specific to the underlying windowing system.

By contrast, Swing components are often described as *lightweight* because they do not require allocation of native resources in the operating system's windowing toolkit. The AWT components are referred to as *heavyweight components*.

Much of the Swing API is generally a complementary extension of the AWT rather than a direct replacement. In fact, every Swing lightweight interface ultimately exists within an AWT heavyweight component because all of the top-level components in Swing (JApplet, JDialog, JFrame, and JWindow) extend an AWT top-level container. However, the use of both lightweight and heavyweight components within the same window is generally discouraged due to Z-order incompatibilities.

The core rendering functionality used by Swing to draw its lightweight components is provided by Java 2D, another part of JFC.
6.5 UML Diagrams (Unified Modeling Language)

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

- **User Model View:**
  - i. This view represents the system from the user’s perspective.
  - ii. The analysis representation describes a usage scenario from the end-users perspective.

- **Structural Model View:**
  - i. In this model the data and functionality are arrived from inside the system.
  - ii. This model view models the static structures.

- **Behavioral Model View:**
  It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

- **Implementation Model View:**
  In this the structural and behavioral as parts of the system are represented as they are to be built.

- **Environmental Model View:**
  In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

**Use case Diagram**

Use Case diagrams identify the functionality provided by the system (use cases), the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system. The primary goals of Use Case diagrams include:
Providing a high-level view of what the system does

Identifying the users ("actors") of the system

Determining areas needing human-computer interfaces

Use Cases extend beyond pictorial diagrams. In fact, text-based use case descriptions are often used to supplement diagrams, and explore use case functionality in more detail.

**Graphical Notation**

The basic components of Use Case diagrams are the Actor, the Use Case, and the Association.

**Actor**

An Actor, as mentioned, is a user of the system, and is depicted using a stick figure. The role of the user is written beneath the icon. Actors are not limited to humans. If a system communicates with another application, and expects input or delivers output, then that application can also be considered an actor.

**Use Case**

A Use Case is functionality provided by the system. Use Cases are depicted with an ellipse. The name of the use case is written within the ellipse.

**Association**

Associations are used to link Actors with Use Cases, and indicate that an Actor participates in the Use Case in some form. Associations are depicted by a line connecting the Actor and the Use Case.

**Class Diagram**

Class diagrams identify the class structure of a system, including the properties and methods of each class. Also depicted are the various relationships that can exist between classes, such as an inheritance relationship. The Class diagram is one of the most widely used diagrams from the UML specification. Part of the popularity of Class diagrams stems from the
fact that many CASE tools, such as Rational XDE, will auto-generate code in a variety of languages. These tools can synchronize models and code, reducing your workload, and can also generate Class diagrams from object-oriented code, for those "code-then-design" maintenance projects.

**Graphical Notation**

The elements on a Class diagram are classes and the relationships between them.

**Class**

Classes are the building blocks in object-oriented programming. A Class is depicted using a rectangle divided into three sections. The top section is the name of the Class. The middle section defines the properties of the Class. The bottom section lists the methods of the class.

**Association**

An Association is a generic relationship between two classes, and is modeled by a line connecting the two classes. This line can be qualified with the type of relationship, and can also feature multiplicity rules (e.g. one-to-one, one-to-many, many-to-many) or the relationship.

**Composition**

If a class cannot exist by itself, and instead must be a member of another class, then that class has a Composition relationship with the containing class. A Composition relationship is indicated by a line with a filled diamond.
Dependency
When a class uses another class, perhaps as a member variable or a parameter, and so "depends" on that class, a Dependency relationship is formed. A Dependency relationship is indicated by a dotted arrow.

Aggregation
Aggregations indicate a whole-part relationship, and are known as "has-a" relationships. An Aggregation relationship is indicated by a line with a hollow diamond.

Generalization
A Generalization relationship is the equivalent of an inheritance relationship in object-oriented terms (an "is-a" relationship). A Generalization relationship is indicated by an arrow with a hollow arrowhead pointing to the base, or "parent", class.

Sequence diagram
A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.
Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the unified activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

6.6 About RATIONAL ROSE (UML Tool)

DESIGN PRINCIPLES & METHODOLOGY:

That combines analysis and design. One reason for this blurring is the similarity of basic constructs (i.e., objects and classes) that are used in OOA and OOD. Through there is no agreement about what parts of the object-oriented development process belongs to analysis and what parts to design, there is some general agreement about the domains of the two activities.

OBJECTORIENTED ANALYSIS AND DESIGN

When Object orientation is used in analysis as well as design, the boundary between OOA and OOD is blurred. This is particularly true in methods former models the problem domain, leading to an understanding and specification of the problem, while the latter models the solution to the problem. That is, analysis deals with the problem domain, while design deals with the solution domain. However, in OOAD subsumed in the solution domain representation. That is, the solution domain representation, created by OOD, generally contains much of the representation created by OOA. The separating line is matter of perception, and different people have different views on it. The lack of clear separation between analysis and design can also be considered one of the strong points of the object-oriented approach the transition from analysis to design is “seamless”. This is also the main reason OOAD methods-where analysis and designs are both performed.

The main difference between OOA and OOD, due to the different domains of modeling, is in the type of objects that come out of the analysis and design process

FEATURES OF OOAD:

- It users Objects as building blocks of the application rather functions
- All objects can be represented graphically including the relation between them.
• All Key Participants in the system will be represented as actors and the actions done by them will be represented as use cases.

• A typical use case is nothing but a systematic flow of series of events which can be well described using sequence diagrams and each event can be described diagrammatically by Activity as well as state chart diagrams.

So the entire system can be well described using OOAD model, hence this model is chosen as SDLC model.

**RATIONAL ROSE ENTERPRISE EDITION 98**

Rational Rose facilitates object-oriented analysis and design, better known as OOAD. In fact, Rose is an acronym for Rational Object Oriented Software Engineering. The great thing about Rose is that it allows analysts, engineers, writers, and project managers to create, view, manipulate, and modify elements in a UML across the entire enterprise, using one tool and one language. The tool’s true value is that it exposes software development problems early on in the development life cycle, helping you manage everything from straightforward projects to more complex software solutions. Rose includes features that simplify the software development process:

**UML MODELING**

• Multilanguage development
• Component-based development
• Internet Web publisher
• Basic report generator
• Database schema generator

At the Project World Boston conference, Rational Corporation called an invaluable aid to any development effort because it unifies software development team modeling.

**Advantages of Rational Rose**

First, let’s explore why you’d want to use Rational Rose. The most immediate advantages are that it:
Facilitates Team Development:

Rose provides complete team support, allowing users (developers and analysts) to work with their own version of the model in their own workspace.

Used throughout the software development process:

From defining the user requirements to implementation, everyone working on the project understands a universal language. You can use Rose at any stage in the life-cycle process. It helps you uncover and prevent potentially serious mistakes downstream.

Makes it easier to manage model changes:

Any change you make to a Rational Rose model you make available to others by using a configuration management and version control (CMVC) system. This lets you integrate changes into the model, no matter where you are in the development stage. Rose typically uses add-in tools, such as Rational’s Clear Case and Microsoft’s Visual Source Cafe, for this purpose.

Saves on creating additional project documentation:

An advantage here is that you simply use the models created in Rose as a basis for design and development. I’ve seen many projects fail because of poor documentation practices.

Addresses bad legacy software:

You should consider using Rose when facing software that doesn’t fit user’s needs, since Rose lets you go back and correct flaws within the legacy application.

Rational Rose uses a GUI that includes a browser, a diagram window, a document window, a standard toolbar, and a diagram toolbar. You simply configure the Rose interface to suit your needs. Let’s face it—in today’s Web-based IT environment, it becomes necessary to design scalable architectures that are easily adapted to constantly changing business conditions. For those solutions that rely on technologies like Enterprise Java, Web, XML, or embedded technology, Rose accelerates implementation by automating proven architectural models for each solution. Rose also establishes a platform for automating architecture-based best Practice tailored to specific solution technologies.

One of the highlights of Rose is its add-in feature, which allows you to simply install programming languages to generate the necessary code. You can quite easily install C++,
PowerBuilder, Forte, Java, Visual Basic, Oracle 8/9, and XML as add-ins. Rose’s add-in feature also lets you install non language tools, such as the popular Microsoft Project. Once you’re working on a model, you can simply deactivate any other feature.

A key factor about Rose is that it allows users to tailor the application to specifically suit their requirements. Apart from providing a level of tractability throughout the software development life cycle, Rose has many features and functionality that users today want to see, such as ready-built frameworks for different types of models—a framework in this context being a set of predefined model elements that are needed to model a specific type of IT solution.

You access frameworks through a Framework wizard, which has a library of predefined frameworks. The Rational Unified Process (RUP) framework is one example in which Rose provides a model structure that follows RUP guidelines. The RUP is also tightly integrated with Rose, making it easy to use. Additionally, it’s no problem to create your own frameworks based on existing models. Rose lets you reuse common elements.

You can also use stereotypes to sub classify UML model elements. For example, you can assign the actor or interface stereotype to it, thereby defining your own stereotypes and assigning them to model elements. This lets you create stereotypes for use cases, classes, packages, relationships, operations, and attributes.

One additional aspect is that Rose facilitates forward and reverse engineering. Forward engineering typically involves generating code skeletons from Rose models. Reverse engineering allows to update or reuse existing code by pinpointing and isolating deviations from the original design.
Chapter 7

System Development

7.1 Modules of the project

7.1.1 Topology Construction

In this module, we construct a topology structure. Here I use mesh topology because of its unstructured nature. Topology is constructed by getting the names of the nodes and the connections among the nodes as input from the user. While getting each of the nodes, their associated port and ip address is also obtained. For successive nodes, the node to which it should be connected is also accepted from the user. While adding nodes, comparison will be done so that there would be no node duplication. Then I identify the source and the destinations.

7.1.2 Randomized Multipath Routing

We achieve randomized multipath routing that can overcome the Compromised Node attack Denial of Service attack. Here multiple paths are computed in a randomized way each time an information packet needs to be sent, such that the set of routes taken by various shares of different packets. As a result, a large number of routes can be potentially generated for each source and destination. To intercept different packets, the adversary has to compromise or jam all possible routes from the source to the destination, which is practically not possible.

7.1.3 Pure Random Propagation

Pure Random Propagation (PRP), shares are propagated based on one-hop neighborhood information. More specifically, a sensor node maintains a neighbor list, which contains the ids of all nodes within its transmission range. When a source node wants to send data to destination, it includes a TTL of initial value N in each share. It then randomly selects a neighbor for each share, and unicasts the share to that neighbor. After receiving the share, the neighbor first decrements the TTL. If the new TTL is greater than 0, the neighbor randomly picks a node from its neighbor list (this node cannot be the source node) and relays the share to it, and so on. When
the TTL reaches 0, the final node receiving this share stops the random propagation of this share, and starts routing it toward the sink using normal min-hop routing.

7.1.4 **SECURED DELIVERY OF PACKETS**

In this module we can maintain the routing table; here we add one more column to maintain the packet delivery ratio. In this one we can maintain how many packets are transmitted over each path. It will be useful for to identify any path can handle number packets. I can stop transmission some amount of time period over that path. So the hacker cannot identify in which path the message is transmitted and also we can easily transmit the data securely. To reduce unnecessary retransmissions and improve energy efficiency, the Gossiping algorithm was proposed as a form of controlled flooding, whereby a node retransmits packets according to a pre-assigned probability. It is well known that the Gossiping algorithm has a percolation behavior, in that for a given retransmission probability, either very few nodes receive the packet, or almost all nodes receive it.

7.2 **code for the module Topology construction**

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.net.*;

public class TopologyCreation extends JFrame {

    private JLabel jLabel1;
    private JLabel jLabel2;
```
private JLabel jLabel3;
private JLabel jLabel6;
private JTextField jTextField1;
private JComboBox jComboBox1;
private JComboBox jComboBox2;
private JButton jButton1;
private JButton jButton2;
private JPanel contentPane;
private Vector allNodes;

public TopologyCreation()
{
    super();

    allNodes = getNodeNames();

    System.out.println("All Nodes are"+allNodes);

    initializeComponent();

    this.setVisible(true);
}

private void initializeComponent()
{

    jLabel1 = new JLabel();
    jLabel2 = new JLabel();
    jLabel3 = new JLabel();

    jButton1 = new JButton();
    jButton2 = new JButton();

    contentPane = new JPanel();
jLabel6 = new JLabel();

jTextField1 = new JTextField();

jComboBox1 = new JComboBox( allNodes );

jComboBox2 = new JComboBox( allNodes );

jButton1 = new JButton();

jButton2 = new JButton();

contentPane = (JPanel)this.getContentPane();

jLabel1.setText(" ");

jLabel2.setText(" Nodes");

jLabel3.setText("Nodes");

jLabel6.setText("Weight");

jTextField1.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        jTextField1_actionPerformed(e);

    }

});

jComboBox1.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        jComboBox1_actionPerformed(e);

    }

});
jComboBox2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jComboBox2_actionPerformed(e);
    }
});

jButton1.setText("Enter");

jButton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButton1_actionPerformed(e);
    }
});

jButton2.setText("Cancel");

jButton2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButton2_actionPerformed(e);
    }
});
contentPane.setLayout(null);
addComponent(contentPane, jLabel1, 0, 1, 535, 111);
addComponent(contentPane, jLabel2, 74, 144, 71, 23);
addComponent(contentPane, jLabel3, 77, 194, 78, 23);
addComponent(contentPane, jLabel6, 78, 245, 68, 21);
addComponent(contentPane, jTextField1, 180, 244, 100, 22);
addComponent(contentPane, jComboBox1, 175, 142, 100, 22);
addComponent(contentPane, jComboBox2, 178, 192, 100, 22);
addComponent(contentPane, jButton1, 95, 301, 83, 28);
addComponent(contentPane, jButton2, 197, 300, 83, 28);
this.setTitle("TopologyCreation");
this.setLocation(new Point(0, 0));
this.setSize(new Dimension(536, 402));
this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
private void addComponent(Container container, Component c, int x, int y, int width, int height) {
    c.setBounds(x, y, width, height);
    container.add(c);
}
private void jTextField1_actionPerformed(ActionEvent e)
{
    System.out.println("\n\njTextField1_actionPerformed(ActionEvent e) called.");
}

private void jComboBox1_actionPerformed(ActionEvent e)
{
    System.out.println("\n\njComboBox1_actionPerformed(ActionEvent e) called.");
    Object o = jComboBox1.getSelectedItem();
    System.out.println("\n\n>>" + ((o==null)? "null" : o.toString()) + " is selected.");
}

private void jComboBox2_actionPerformed(ActionEvent e)
{
    System.out.println("\n\njComboBox2_actionPerformed(ActionEvent e) called.");
    Object o = jComboBox2.getSelectedItem();
    System.out.println("\n\n>>" + ((o==null)? "null" : o.toString()) + " is selected.");
}

private void jButton1_actionPerformed(ActionEvent e)
{
    updateNodeInfo();
}

private void jButton2_actionPerformed(ActionEvent e)


    System.out.println("\nImageButton2_actionPerformed(ActionEvent e) called.");

    }

public Vector getNodeNames()

    {

Vector<String> nodeList = new Vector<String>();

    try
    {
    DB.DBConnection obj = new DB.DBConnection();
    Connection con = obj.getConnection();
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("select * from NodeInfo");
    while(rs.next())
    {
        nodeList.addElement(rs.getString(1));
    }
    }
    }
}

    catch (Exception e)
    {
        e.printStackTrace();
    }
return nodeList;
}

public void updateNodeInfo()
{
    Socket soc;
    ObjectInputStream dis;
    ObjectOutputStream dos;
    try
    {
        String startingNode = jComboBox1.getSelectedItem().toString().trim();
        String endingNode = jComboBox2.getSelectedItem().toString().trim();
        String edgeWeight = jTextField1.getText().trim();
        String nodeDetails = startingNode+"#"+endingNode+"#"+edgeWeight;
        if ( true )
        {
            soc = new Socket("localhost",2345);
            dos = new ObjectOutputStream(soc.getOutputStream());
            dos.writeObject("insert");
            dos.writeObject( nodeDetails );
            dis = new ObjectInputStream(soc.getInputStream());
            String reply = (String)dis.readObject();
if( reply.equals("inserted"))
{
    JOptionPane.showMessageDialog(this,"Inserted Successfully");
}
else
{
    JOptionPane.showMessageDialog(this,"This Link Already Available");
}
}

catch (Exception e)
{
    e.printStackTrace();
}
}
Chapter 8

EXPERIMENTAL RESULTS

Screens:

Fig 8.1 Input form
Fig 8.2 GetNodeInfo Form
Fig 8.2 GetNodeInfo Form
Fig 8.2 GetNodeInfo Form
Fig 8.3 Topology Creation
Fig 8.3 Topology Creation
Fig 8.4 Login Form
Fig 8.5 Node Name
Fig 8.6 Possible Routes
Fig 8.7 Dynamic Routing for Sending a Data
Fig 8.8 Dynamic Routing
Fig 8.9 View Message
Fig 8.10 Message Transmission
Fig 8.11 Output Form
Chapter 9

Testing

Testing is a process, which reveals errors in the program. It is the major quality measure employed during software development. During software development, the program is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as it is expected to perform.

9.1 Testing Concepts

Software testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for well planned through testing.

9.1.1 Testing procedures for the project is done in the following sequence

- System testing is done for checking the server name of the machines being connected between the customer and executive.
- The product information provided by the company to the executive is tested against the validation with the centralized data store.
- System testing is also done for checking the executive availability to connect to the server.
- The server name authentication is checked and availability to the customer
- Proper communication chat line viability is tested and made the chat system function properly.
- Mail functions are tested against the user concurrency and customer mail date validate.

9.2 Testing Objectives

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say,

- Testing is a process of executing a program with the intent of finding an error.
A successful test is one that uncovers an as yet undiscovered error.

A good test case is one that has a high probability of finding error, if it exists.

The tests are inadequate to detect possibly present errors.

The software more or less confirms to the quality and reliable standards.

9.3 Types of Testing

9.3.1 Unit Testing:

Unit testing focuses verification effort on the smallest unit of software design module. Important of tests and uncovered errors is limited by the constrained scope established for unit testing. In unit testing, each module is tested separately. In this project web page controls are evaluated by unit testing.

9.3.2 Integration Testing:

Integration testing is a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing. The objective is to take unit tested modules and build a program structure that has been dictated by design.

There are two strategies in integration testing.

- Top-down Integration
- Bottom-up Integration

In **Top-down integration** modules are integrated by moving downward through the control hierarchy, beginning with the main control module. **Bottom-up integration** testing begins construction and testing with atomic modules. That is, modules at the lowest levels in the program structure. This project follows bottom-up integration.

9.3.3 System Testing

Once individual testing is completed, modules are assembled and integrated to perform as a system. Then the top down testing begins from upper level to lower level module testing was carried out to check whether the entire system is performing satisfactory. This is the test done for whole system integration after completion of the module testing.
9.3.4 Acceptance Testing

When the user fined no major problems with its accuracy, the system passes through a final acceptance test. This test confirms that the system meets the original goals, objectives and requirements established during analysis. The responsibility for requirements it is finally acceptable and ready for operation.

9.3.5 User Interface Testing

The data that is given by the user should lie on the particular range. When the user given wrong data, the system will give error message. According to this the program was developed. The value of the data should fall on the given range of value.

The following are some of the testing strategies that are carried out during the testing period:

Specification testing

Executing the specification starting what the program should do and how it should it perform under various conditions. Test cases for various situations and combination of condition in all the modules were tested.

Module testing

To locate errors, focus is given on modules, independent of other modules. This is known as module testing. This enables us to detect error and correct it without affecting any other modules. Whenever the program was not executing the required function, it was corrected to get required results. Thus all the modules are tested from bottom to up starting with the smallest and lowest level modules and proceedings to next level. Individual like purchase order rising was done successfully for connection to the next modules.

Test data and output

For all types of test dummy data were keyed in and after the testing was over live data were keyed in from the source documents and concerned department people were followed to key in and work on the system.
9.3.6 Validation Testing:

Validation is the process of evaluating software at the end of the software development process to determine compliance with the requirements, various validations. In this project,Textbox is the input box for getting the user's inputs. If the user does not enter the correct data type mismatch data type error will be occurred.

9.4 Levels of Testing

In order to uncover the errors present in different phases we have the concept of levels of testing. The basic levels of testing are

![Levels of Testing Diagram]

Fig .8.4.1 Levels of Testing
## 9.5 Test case

<table>
<thead>
<tr>
<th>Test case id</th>
<th>Description</th>
<th>Input value</th>
<th>Expected value</th>
<th>Actual value</th>
<th>Result</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Request for Number of Nodes</td>
<td>Integer value</td>
<td>Accepted value</td>
<td>Accepted value</td>
<td>Pass</td>
<td>Valid input</td>
</tr>
<tr>
<td>2</td>
<td>Minimum threshold number of nodes</td>
<td>Any integer value</td>
<td>Accepted value</td>
<td>Accepted value</td>
<td>Pass</td>
<td>Valid input</td>
</tr>
<tr>
<td>3</td>
<td>Maximum threshold number of nodes</td>
<td>Unlimited value</td>
<td>Accepted value</td>
<td>Accepted value</td>
<td>pass</td>
<td>Valid input</td>
</tr>
<tr>
<td>4</td>
<td>Request for number of nodes</td>
<td>Character value</td>
<td>Accepted value</td>
<td>Not accepted value</td>
<td>Fail</td>
<td>Invalid input</td>
</tr>
</tbody>
</table>

Table 8.5 Test cases for execution
Chapter 10

CONCLUSION AND FUTURE WORK

Our simulation results have shown the effectiveness of randomized dispersive routing in combating CN and DOS attacks. By appropriately setting the secret sharing and propagation parameters, the packet interception probability can easily be reduced by the proposed algorithms to as low as $10^{-3}$, which is at least one order of magnitude smaller than approaches that use deterministic node-disjoint multi-path routing. At the same time, we have also verified that this improved security performance comes at a reasonable cost of energy.

To enhance the security features instead of using random path selection, using a method called odd & even path selection method, also making use of cryptographic algorithm, in order to get efficient packets.
Bibliography

REFERENCES


TEXT BOOKS

1. Java complete reference by Herblet Shield
2. Object Oriented Analysis & Design by Grady Booch

WEB REFERENCES

- http://java.sun.com
- http://www.sourceforgegde.com
- http://www.networkcomputing.com/
- http://www.java2s.com/