SHORTEST PATH IN A WIRELESS SENSOR NETWORK WITH MULTIPLE SENSOR FAILURES

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Sandeep Reddy Poreddy

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

October 2011

Fargo, North Dakota

North Dakota State University
Graduate School

Title

SHORTEST PATH IN A WIRELESS SENSOR

NETWORK WITH MULTIPLE SENSOR FAILURES

By

SANDEEP REDDY POREDDY

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Poreddy, Sandeep Reddy, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, October 2011. Shortest Path in a Wireless Sensor Network with Multiple Sensor Failures. Major Professor: Dr. Kendall Nygard.

This paper shows how a shortest path can be obtained in a wireless sensor network, between a source sensor and a destination sensor, in a hop-by-hop fashion, considering multiple sensor failures along the path of data transmission.

A wireless sensor network consists of a number of sensors spread across a geographical area; these sensors are distributed either randomly or systematically. Every sensor possesses the capability to communicate with other sensors, and each sensor possesses some level of intelligence to perform processing of the signals. Efficiency of a sensor network depends on the design of the network topology. The network must be continuously functional to perform the task; a major problem in wireless sensor networks is due to sensor failures. In order to remain functional in spite of sensor failures, it is required that an alternative shortest path is found to send and receive requests to fulfill the task. I addressed this problem by finding an alternative shortest path based on the hop count. I developed a web based application to simulate a network and find shortest paths in a network with multiple sensor failures. I performed an experimental analysis in finding the shortest path.

When a source sensor has data to transmit to a destination sensor, it broadcasts a RREQ (Route Request) to its immediate neighbors. A route to the source is created at every sensor when a RREQ is received. If the receiving sensor has not received this RREQ before and if it is not the destination, then it broadcasts the RREQ to its immediate neighboring sensors.

If the receiving sensor is the destination, it generates a RREP (Request Reply). The RREP is uni-cast to the source sensor in a hop-by-hop fashion and a shortest path is obtained. Broadcasting is stopped at the point where a destination sensor receives a RREQ and acknowledges by RREP. To obtain a new shortest path, one or more sensors in the previously obtained shortest path are failed and the algorithm continues from the point where it has broadcasted RREQ to the sensors in the network previously and a new shortest path is obtained.

ACKNOWLEDGEMENTS

I would take this opportunity to thank my advisor, Dr. Kendall Nygard, who has given me valuable support, encouragement and advice without which this work would not have been completed. I am thankful to the members of the committee, Dr. Wei Jin, Dr. Gursimran Walia and Dr. Benton Duncan, for their support and suggestions. I would also like to thank my good friends Mr. Shyamsundar Seshagiri and Mr. Srikar Pachva for their valuable support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Wireless sensor networks have always been the subject of interest due to their wide variety of potential applications. These networks were primarily researched for the military applications, and now have a wide industrial and consumer applicability through its remote interaction with the physical world.

1.1. Wireless Sensor Network

The building blocks of a wireless sensor networks are nodes (sensors). These nodes interact with other nodes collectively or cooperatively through links. Links provide a form of communication through which the nodes transmit and receive data. These nodes can be of varying sizes, depending on how the network is constructed; each sensor may have certain physical characteristics such as the range of transmission, battery power, computational speed and communications bandwidth etc. These factors govern the communicability of the node.

1.1.1. Applications of Wireless Sensor Networking

Currently, wireless sensor networks are used in a wide range of applications, primarily used for monitoring purposes. They are also used in other important applications such as environmental monitoring, traffic control application, weather checking, regularity checking of temperature, tank level monitoring, the daily life applications such as in agriculture, water level monitoring, green house monitoring, landfill monitoring etc.[1].

1.1.2. Pros and Cons of Wireless Sensor Networking

Wireless sensor networking have a variety of advantages: they operate and store a small amount of energy; they do not require cabling and can provide mobility. The major advantage of these networks is that they can operate effectively under jagged environmental conditions and they have large scalability to cover a wide geographical area. Some of its disadvantages come from its inefficient speed of communication and that are a bit costly to install and use.

1.2. Problem Statement

To develop and implement an algorithm to identify an alternative path in a wireless sensor network, considering the fact that multiple sensor failures can occur along the path of data transmission between the source sensor and the destination sensor.

1. A fixed geographical area with n x n dimensions, where n is the number of sensors.
2. A finite set of homogeneous sensors located at random geographical positions within n x n dimensions.
3. A source sensor and a destination sensor are chosen, the source sensor transmits data to the destination sensor.
4. Multiple sensor failures can occur along the path of transmission.
5. An alternative shortest path must be identified, if sensor failures occur along the path of transmission.
6. The performance of a network is measured in terms of Hop Count, CPU utilization, CPU Process Time, and Memory Consumption in finding a shortest path.

The size of the geographical area in which the sensors are deployed increases with an increase in the number of sensors. The size of the area is n x n, where; n is the number of sensors. Initially, a source and a destination sensor are chosen. A sensor network is formed by connecting the n sensors using distance matching. The maximum distance a sensor can transmit data is determined by its transmitting distance. A shortest path is found between the source and the destination senor. The performance metrics; Hop Count, CPU Process Time, CPU Processing % and Heap Memory are found for the shortest path using JConsole. An, alternate shortest path is found by failing the sensors in the previously obtained shortest path. The performance metrics are found for the new shortest path. The values of the current and the previously obtained performance metrics are compared and analyzed.

1.3. Outline

The remainder of the paper is constructed as follows: Chapter 2 talks about the background work. Chapter 3 describes the related work performed on wireless sensors to address the problem. Chapter 4 describes the algorithm used. Chapter 5 describes the methods used to solve the problem. Chapter 6 describes the working of the application. Chapter 7 describes the user operations on the application with screenshots. Chapter 8 shows the experimental analysis performed. Chapter 9 presents the conclusion and future work. Chapter 10 lists the references used.

2. BACKGROUND

The building blocks of a wireless sensor network are nodes (sensors). These nodes interact with other nodes collectively or cooperatively through links. Links provide a form of communication through which the nodes transmit and receive data [5]. These nodes usually communicate in a multi-hop fashion. The flow of data stops at special nodes called the base stations or sinks. A base station links one sensor network to another, like a gateway to propagate the data received for further processing. The base stations have special capabilities over normal nodes in that they involve certain physical characteristics such as the range of transmission, battery power, computational speed and communications bandwidth etc. These factors govern the communicability of the node.

The intricacy of the sensor networks is based on the energy consumption, which basically affects the communication between the sensor nodes. To overcome this issue, aggregation points may be introduced into the network. This lowers the accrued number of messages exchanged between the sensor nodes and thus results in lower energy consumption [8]. Aggregation point is a normal node, which arrogates data from neighboring nodes; it involves some kind of processing, and then transmits the filtered data to the next hop. Comparable to aggregation points is clustering [9]. Here, the sensor nodes are put together into clusters, with each cluster having a "cluster head" as the leader. All communication within a cluster must occur through the cluster head, which is then transmitted to a neighboring cluster head until the destination is reached (the base station). Another form of energy saving mechanism is setting the nodes in passive mode, if they are not needed and then bringing them to active mode by notifying them when needed.

2.1. Features of Sensor Network

The features of a sensor network depend on the features of the sensors that form the network. The features such as the range of transmission, battery power, computational speed and communications bandwidth define the characteristics of the sensor in a network.

2.1.1. Sensor Node Hardware Architecture

Figure 1 represents a Mica2 sensor node, currently the noted research staging at the moment. The essential components that form the sensor node are: a processing unit, which is an intelligent device with the capabilities to process data, a memory storage to store data, an antenna, a radio frequency (RF) transceiver that facilitates communication between nodes, a power unit that keeps the device operational, a power switch to turn the sensor on and off and connectors to interface with external devices.

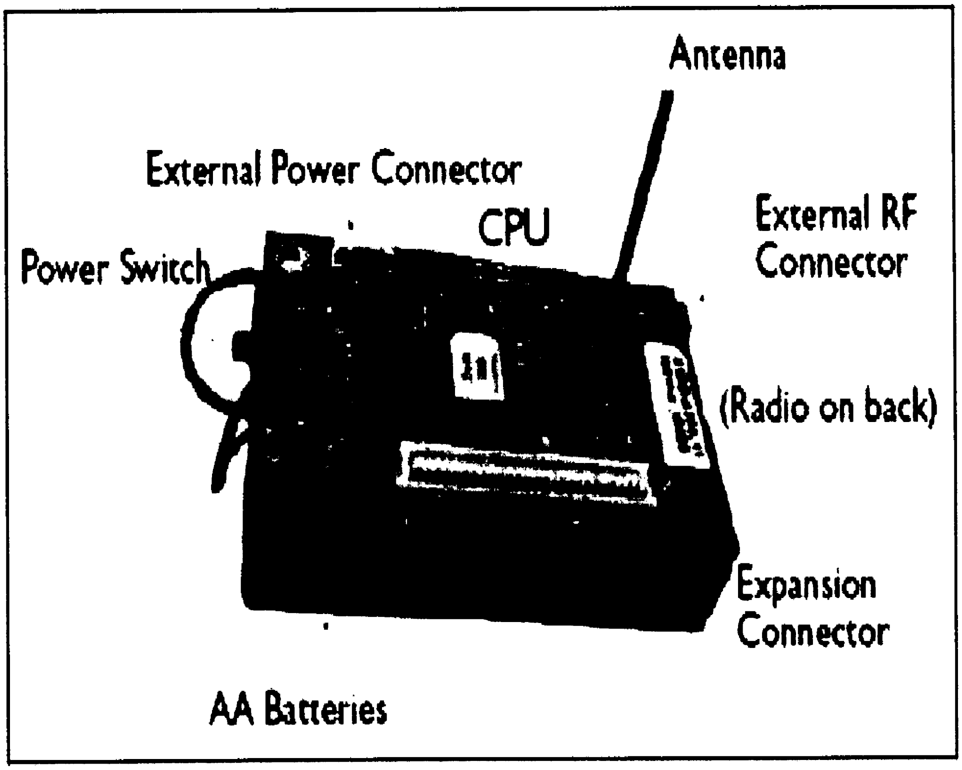


Figure 1: Mica2 Sensor [2]

2.2. Use of Sensor Networks

Wireless sensor networks have improved over the last few years due to amount of underlying research in a variety of areas of interest. The nodes sense environmental changes and report them to other nodes over flexible network architecture. Sensor nodes are great for deployment in hostile environments or over large geographical areas [10]. Wireless sensor networks are put into use in a variety of domains. The areas of domain operation include:

- **Environmental Observation**: Sensor networks can be used to monitor environmental changes. An example could be water pollution detection in a lake that is located near a factory that uses chemical substances. Sensor nodes could be randomly deployed in unknown and hostile areas and relay the exact origin of a pollutant to a centralized authority to take appropriate measures to limit the spreading of pollution. Other examples include forest fire detection, air pollution and rainfall observation in agriculture [2].
- **Military Monitoring:** The military uses sensor networks for battlefield surveillance; sensors can monitor vehicular traffic, track the position of the enemy or even safeguard the equipment of the side deploying sensors [2].
- **Building Monitoring**: Sensors can also be used in large buildings or factories monitoring climate changes. Thermostats and temperature sensor nodes are deployed all over the building's area. In addition, sensors could be used to monitor vibration that could damage the structure of a building [2].

3. RELATED WORK

The AODV Routing protocol uses an on-demand approach for finding routes, that is, a route is established only when it is required by a source node for transmitting data packets. It employs destination sequence numbers to identify the most recent path. In AODV, the source node and the intermediate nodes store the next-hop information corresponding to each flow for data packet transmission. In an on-demand routing protocol, the source node floods the *RouteRequest* packet in the network when a route is not available for the desired destination. It may obtain multiple routes to different destinations from a single *RouteRequest.* The major difference between AODV and other on-demand routing protocols is that it uses a *destination sequence number* (DestSeqNum) to determine an up-to-date path to the destination. A node updates its path information only if the *DestSeqNum* of the current packet received is greater than the last *DestSeqNum* stored at the node [3].

A RouteRequest carries the *source identifier* (SrcID), the *destination identifier* (DestID), the *source sequence number* (SrcSeqNum), the *destination sequence number* (DestSeqNum), the *broadcast identifier* (BcastID), and the *time to live* (TTL) field. DestSeqNum indicates the freshness of the route that is accepted by the source. When an intermediate node receives a RouteRequest, it either forwards it or prepares a RouteReply if it has a valid route to the destination. The validity of a route [4] at the intermediate node is determined by comparing the sequence number at the intermediate node with the destination sequence number in the RouteRequest packet. If a RouteRequest is received multiple times, which is indicated by the BcastID-SrcID pair, the duplicate copies are discarded. All intermediate nodes having valid routes to the destination, or the destination node itself, are allowed to send RouteReply packets to the source. Every intermediate node, while forwarding a RouteRequest, enters the previous node address and it's BcastID. A timer is used to delete this entry in case a RouteReply is not received before the timer expires. This helps in storing an active path at the intermediate node as AODV does not employ source routing of data packets [6]. When a node receives a RouteReply packet, information about the previous node from which the packet was received is also stored in order to forward the data packet to this next node as the next hop toward the destination. [3].

4. ALGORITHM

The algorithm below is developed to obtain the shortest path in a wireless sensor network with multiple sensor failures. A source sensor (SOURCE), destination sensor (DEST) transmission distance (TRANS_DIS) and number of sensors in the network (MAX_SEN) are supplied as input parameters. The source sensor broadcasts RREQ (Route Request) to all its neighboring sensors. All sensors within the range of sensor broadcasting are its neighbors. If a neighboring sensor is the destination sensor, then a RREP (Request Reply) is uni-casted back to the source sensor and the path is noted. The algorithm stops here if the destination is found. If the destination sensor is not found, the neighboring sensors which have received the RREQ broadcasts the RREQ to its immediate neighbors, if the neighboring sensor happens to be the destination, then a RREP is uni-casted back to the source sensor and the path is noted. This process proceeds until the destination sensor receives the request. On obtaining the shortest path, multiple sensors in the shortest path can be failed to obtain the new shortest path. Below is the algorithm for finding shortest path:

STEP 1: A source sensor (SOURCE), destination sensor (DEST), transmission distance (TRANS_DIS) and number of sensors (MAX_SEN) chosen.

STEP 2: A sensor network is formed connecting the MAX_SEN (refer section 5.2 for more information on network formation)

STEP 3: SOURCE broadcasts RREQ to its neighboring sensors

STEP 4: If neighboring sensor is DEST then BREAK

STEP 5: Unicast RPEP to SOURCE, note unicast path (SHORTEST_PATH)

STEP 6: ELSE CONTINUE; REPEAT STEPS 3 through 5 WHILE DEST reached

STEP 7: Fail sensors in the SHORTEST_PATH; exclude SOURCE and DEST

STEP 8: Update sensor network with failed sensors.

STEP 9: REPEAT STEPS 3 through 5 while DEST

Refer section 7 for implementation details.

5. METHOD

This section explains how a sensor network is formed, i.e. how the sensors are linked to one another in the network, data transmission between the source sensor and a destination sensor, via homogenous intermediate sensors. A routing table maintains the state of each sensor at any point of time in the network. This is followed by an example that illustrates how a shortest path is obtained between the source sensor and a destination sensor. An overview of the technologies and application code is listed in tables.

5.1. Abbreviation

MAX_SEN:	number of sensors entered by the user
SOURCE:	source sensor that wants to transmit data
DESTINATION:	destination to which the source transmits the data
TRANS_DIS:	maximum distance that a sensor is capable of transmitting data
N:	*MAX_SEN x MAX_SEN* dimensions

5.2. Forming a Dynamic Network of Sensors

The first step is to form a network connecting the MAX_SEN in space in *N x N* dimensions. The figure below represents the sensors in the network, initially before forming the network; the sensors are not linked to other sensors.

Figure 2 represents sensors named sensor1, sensor2, sensor3, sensor4, sensor5, and so on up to sensorMAX_SEN in space. These sensors must be connected to form a network. Linking the sensors involves a complex logic "distance matching". A network is formed, by connecting the sensors and a shortest path is found by choosing a source sensor and a destination sensor in the network. Alternate shortest path is found by failing sensors in the previously obtained shortest path.

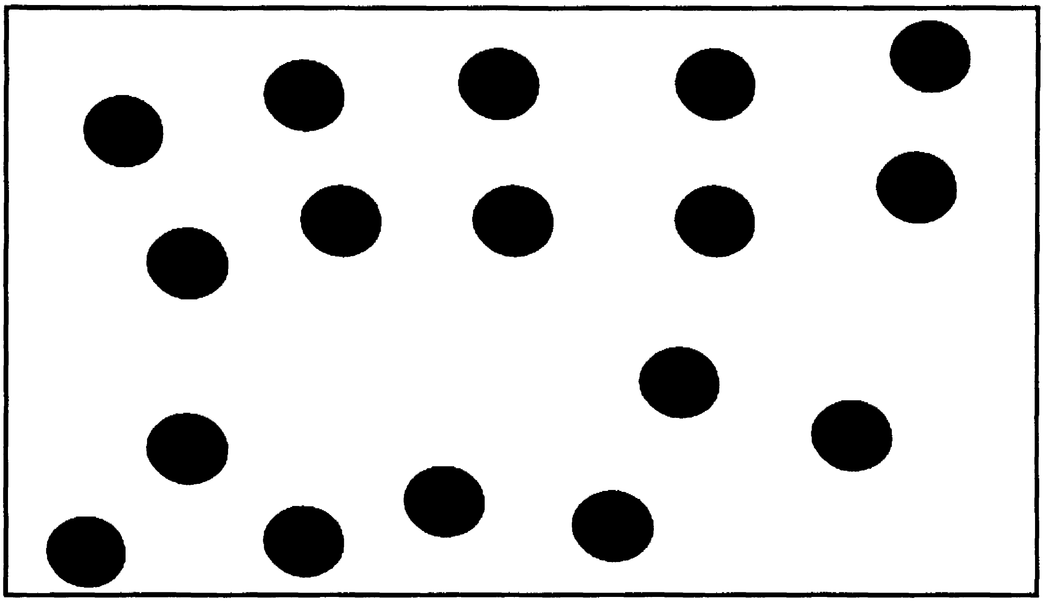


Figure 2: Sensors Distributed in Space Without Links

5.3. Distance Matching

The core idea behind distance matching is to examine a string for the existence of a given transmission distance pattern, say "18", in a random string "1234846357361**8**35" of digits, where the pattern "18" is the TRANS_DIS. The pattern stays standard for testing the presence of links in the application. The pattern is examined $MAX\_SEN - 1$ times for each sensor and each time on a random string of digits. Existence of the pattern in the string signifies the presence of a link between sensorX (checked for) and sensorY (checked with). Practically speaking, each sensor may be linked to MAX_SEN - 1 sensors, but not necessarily. The degree of a sensor depends on the intricacy of the pattern. E.g. a simple pattern "23" has a high probability of occurrence in a random string; the sensor in this case will have a high degree (greater number) of links. On the other hand a complex pattern "32324243435" has a low probability of occurrence in a random string. In this case a sensor will have a low degree (smaller number) of links. Occurrence or non occurrence of the pattern dictates the structure of the network. Figure 3 represents a sensor with a high degree and a low degree of links.

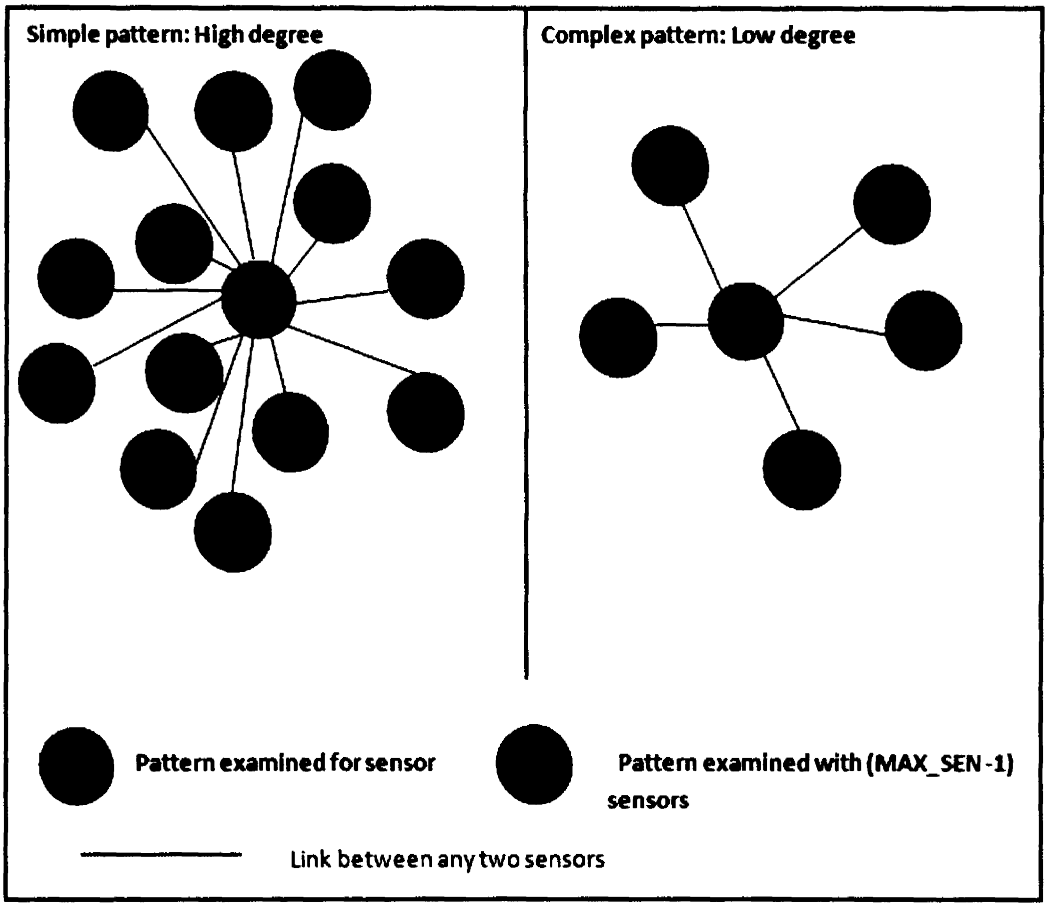

Simple pattern: High degree
Complex pattern: Low degree
Pattern examined for sensor
Pattern examined with (MAX_SEN -1) sensors
Link between any two sensors

Figure 3: Degree of Sensors

5.4. Split Networks

There may be a case, where the pattern examined, may not occur in a random string. This means that there is no link between the sensorP and sensorQ (where sensorP and sensorQ are sensors in MAX_SEN sensors). If this is the case with multiple sensors, there is a chance that more than one network may be formed (SPLIT networks). This scenario is overthrown by connecting random sensor(s) at the boundary in SPLIT_1 network to random sensor(s) (with a transmission distance less than or equal to the TRANS_DIS) in at the boundary SPLIT_2 network (where

SPLIT_1 and SPLIT_2 are two different networks). Figure 4 represents two split networks, and how these networks are connected.

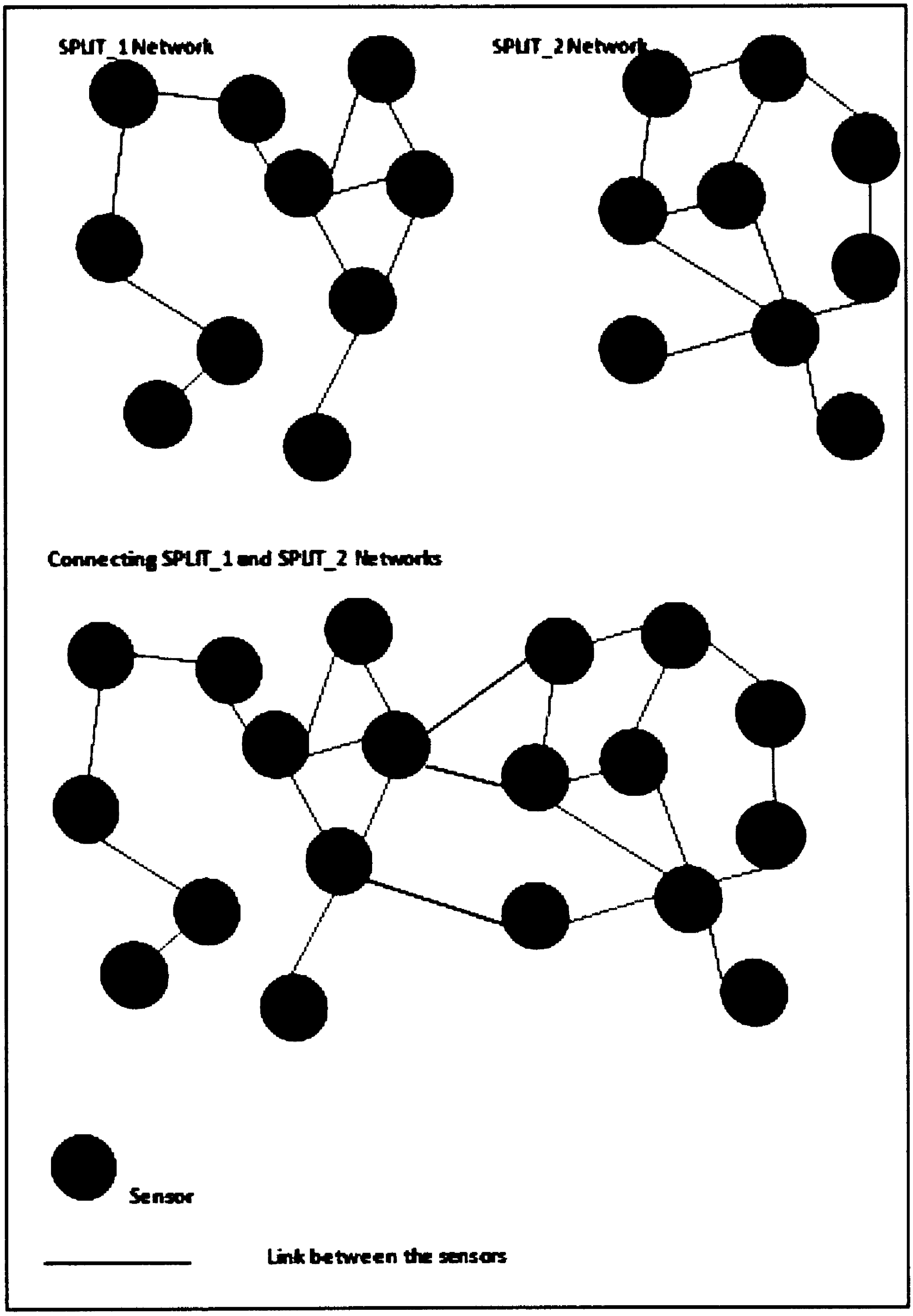
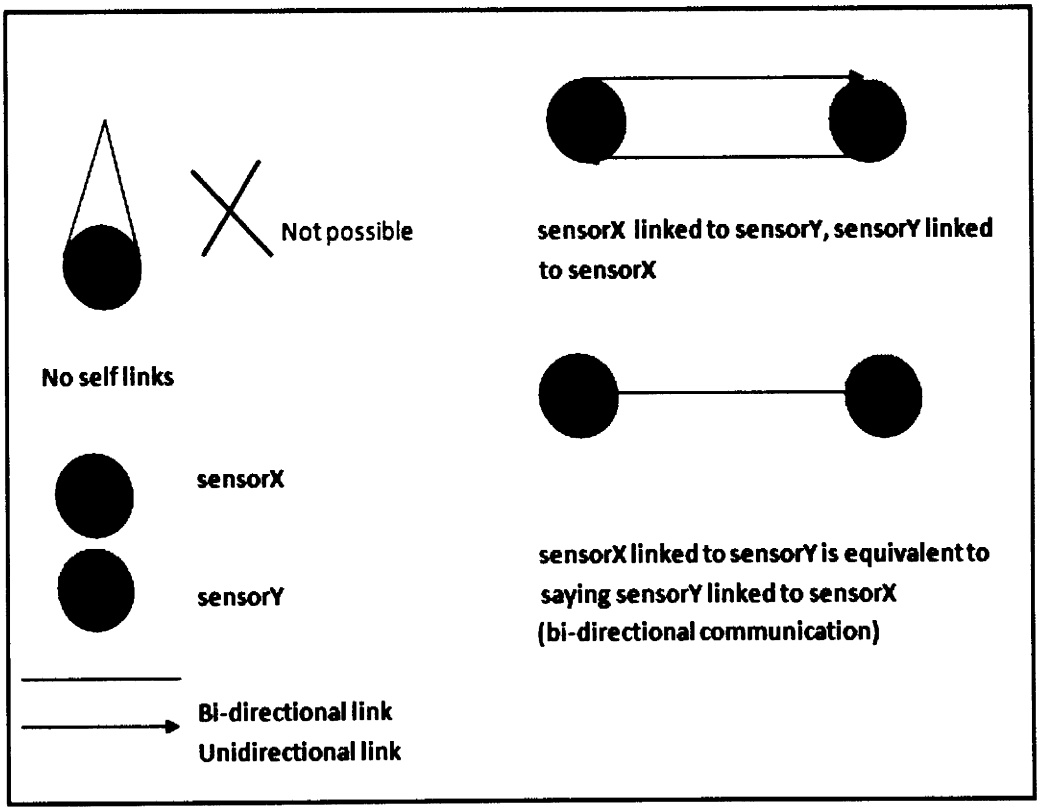


Figure 4: Split Networks

5.5. Sensor Self Link

Figure 5 shows that a sensor cannot be linked to itself and thus, no self link to a sensor can exist. If a sensorX has a link to a sensorY, it means that sensorY has a link to sensorX.



Not possible
No self links
sensorX linked to sensorY, sensorY linked to sensorX
sensorX
sensorY
sensorX linked to sensorY is equivalent to saying sensorY linked to sensorX (bi-directional communication)
Bi-directional link
Unidirectional link

Figure 5: Sensor Links

5.6. Method of Data Transmission

When a source sensor has data to transmit to a destination, it broadcasts a RREQ (Route Request) to its immediate neighbors. A route to the source is created at every sensor when a RREQ is received. If the receiving sensor has not received this RREQ before and if it is not the destination, then it broadcasts the RREQ to its immediate neighboring sensors. If the receiving sensor is the destination then it generates a RREP (Request Reply). The RREP is uni-cast to the source sensor in a hop by hop fashion and a shortest path is obtained. Broadcasting is stopped at the point where the destination sensor receives a RREQ, and acknowledges by RREP. To obtain a new shortest path one or more sensors in the shortest path are failed and the algorithm continues from the point it has broadcasted RREQ to the sensors previously in the network and a new shortest path is obtained. Figure 6 shows how a sensor broadcasts RREQ to its neighboring sensors before and after sensor failures.

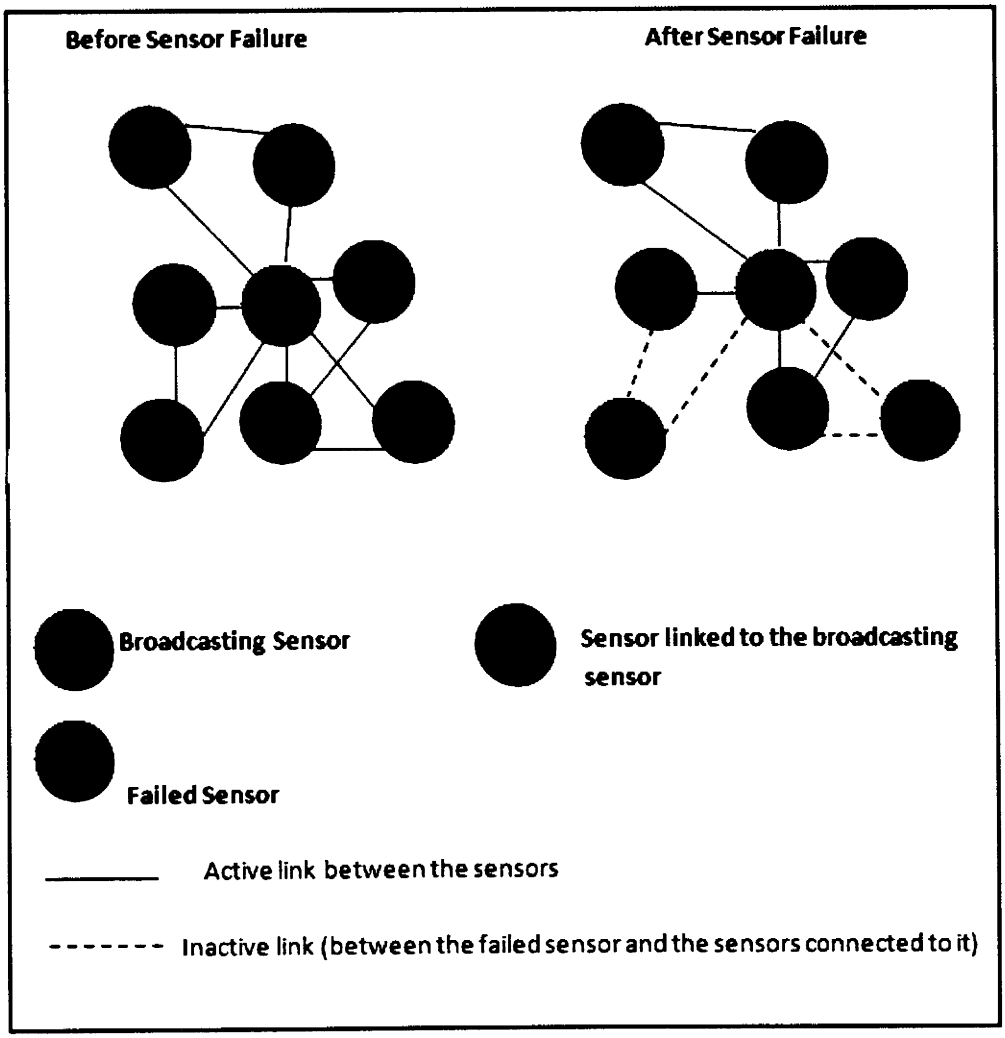
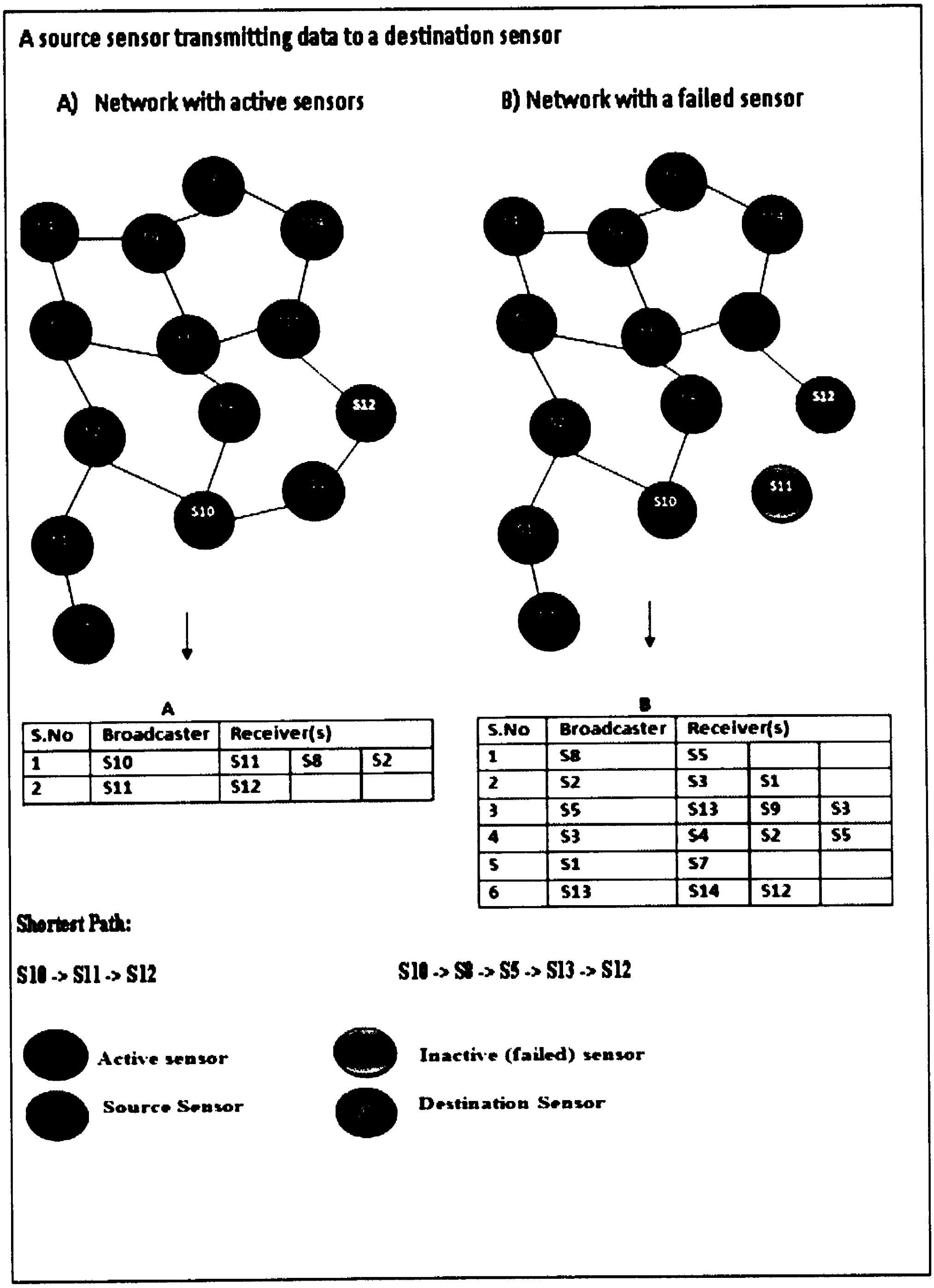

Before Sensor Failure
After Sensor Failure
Broadcasting Sensor
Sensor linked to the broadcasting sensor
Failed Sensor
Active link between the sensors
Inactive link (between the failed sensor and the sensors connected to it)

Figure 6: Data Transmission

This algorithm broadcasts only till the destination is found i.e. it only identifies the shortest path. All other paths to the destination are of no interest and speaking more specifically, the algorithm will not find them. This would be more effective and saves time when a source sensor has data to transmit to the destination. Figure 7 shows how an alternate shortest path is obtained in case of sensor failures.



A source sensor transmitting data to a destination sensor

A) Network with active sensors

B) Network with a failed sensor

A

S.No	Broadcaster	Receiver(s)		
1	S10	S11	S8	S2
2	S11	S12		

B

S.No	Broadcaster	Receiver(s)		
1	S8	S5		
2	S2	S3	S1	
3	S5	S13	S9	S3
4	S3	S4	S2	S5
5	S1	S7		
6	S13	S14	S12	

Shortest Path:

S10 -> S11 -> S12

S10 -> S8 -> S5 -> S13 -> S12

Figure 7: Source Sensor Transmitting Data to a Destination Sensor

5.6.1. Routing Table

All sensor data is maintained in a database. Table 1 represents a routing table for each sensor with detailed information will be seen on a page. The routing table lists the sensor ID, sensor name, its neighbors, state (active or inactive), route request, route reply, route error. Table 1: Routing Table for Sensor Named “sensor42”

Id	Link From	Link To	RREQ	RREP	Status
1	sensor42	sensor23	Yes	Yes	Active

5.6.2. Technologies Used

Table 2 lists the technologies used for developing the application for finding the shortest path in a wireless sensor network.

Table 2: Technologies Used

S.No	Technology	Version
1	Java	1.6.0_12
2	Servlets	2.5
3	JSP	2.2
4	JDBC	mysql-connector-java-5.1.8
5	Apache Tomcat	6.0.20
6	MSQL	5.1
7	Eclipse	Ganymede
8	Apache Logging	1.1.1
9	Project build management	Maven 2.2.1
10	XML	1.0
11	Microsoft Visual Web Developer 2010	Express Edition 2010
12	Microsoft Word	2007

5.6.3. Code

Table 3 lists the file type, number of files, lines of code and the total number of lines of code in the application.

Table 3: Code

S.No	File Type	Number of Files	Lines of Code	Total Number of Lines of Code
1	Java Interface	3	32	
	Java Servlets	4	247	
2	Java Concrete classes	3	1022	
3	Java Bean classes	3	195	
4	Java Business Utility classes	3	920	
5	JMX classes	4	268	
6	Java Server Pages	7	302 (only embedded java code)	
7	Maven pom.xml (configuration file)	1	68	
8	web.xml (configuration file)	1	58	
Total				3414

6. WORKING OF THE APPLICATION

This is a three tier web application. The user is provided a GUI (Graphical User Interface) made up of JSP (Java Server Pages). The subsections below list the required user inputs, error handling and working of the application.

6.1. Number of Sensors in the Network

It defines the number of sensors in the network. The user could define sensors starting from 2 to 65,535 (this is a limit because the underlying data type is int (4 Bytes)).

6.1.1. Source Sensor

It is the source sensor that wants to transmit data to the destination sensor. A source sensor is defined as a "sensor" + (digit, where 3 < digit < 65,535). Example: Sensor34. The user enters the source sensor name in the source text box

6.1.2. Destination Sensor

It is the destination to which the source transmits data. A destination sensor is defined as "sensor" + (digit, where 3 < digit < 65,535). Example: Sensor556. The user enters the destination sensor name in the destination text box.

6.1.3. Transmission Distance

It is the maximum distance that each sensor can transmit data. The user enters the transmission distance value in the text box supplied.

6.2. Error Handling

Error handling is done to ensure that the user enters only valid data such as checking the value of the number of sensors and ensuring that no field is left empty. The user can reset the fields by clicking the reset button. The data entered by the user is submitted on clicking the submit button.

6.3. Working

The data entered by the user is submitted on clicking the submit button. The underlying servlet is SensorNetwork.java. This servlet retrieves the parameters from network.htm, performs the required data validation and dispatches it to action.htm. Behind the scenes, a network is formed; the shortest path is obtained on this network. Once the network is ready, the user is provided with the option to see how the sensors are linked in the network in the form of a table by clicking the 'Sensor Table' link. A sensor table is displayed listing all the sensors, their neighbors and their status. The user can fail sensors by clicking the 'Fail Sensors' link. This action generates the shortest path between the source and destination, with an option to enter the sensors to be failed in a text area box.

The relation between the sensors can be seen in the form of a table, on clicking the 'Broadcast Table' link. This table lists how the sensors are linked, which sensors have broadcasted, the receiving sensors, the request, and reply along with the status of the sensor.

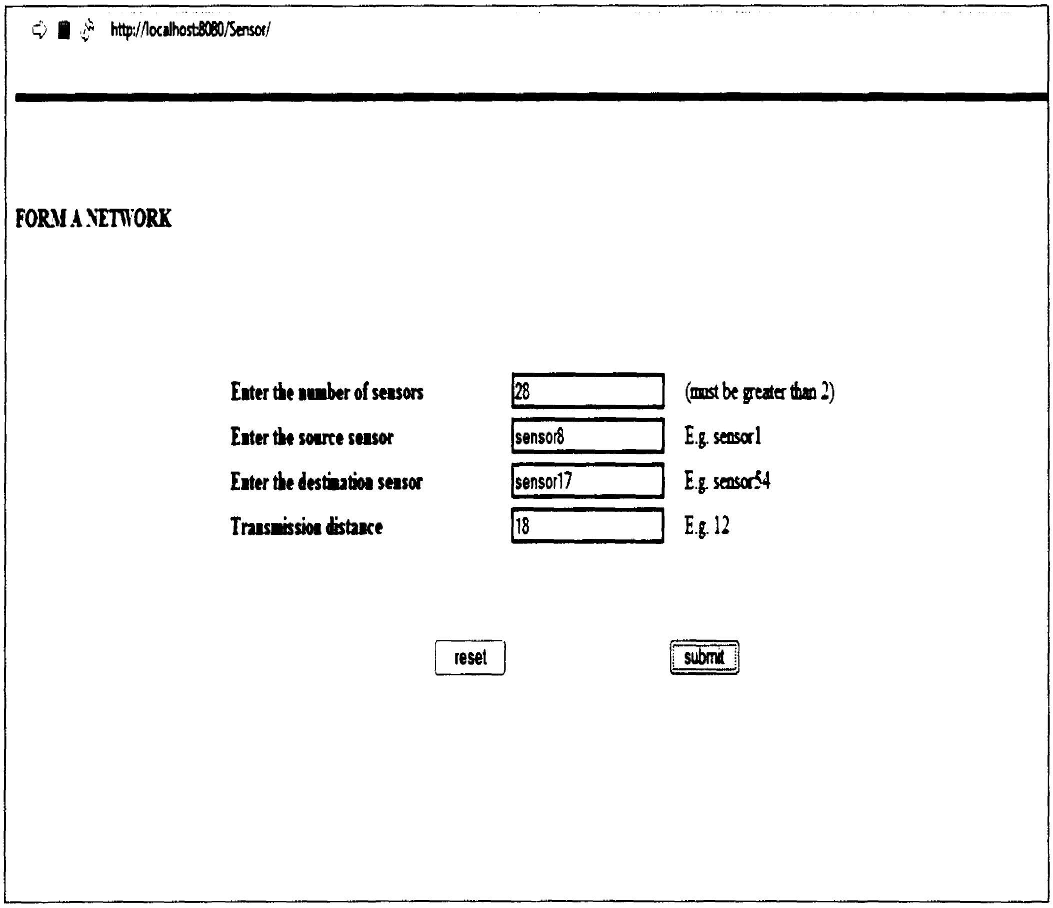
The sensors to be failed must be separated by a comma e.g. sensor45, sensor67, sensor11. On clicking submit, backend processing is done to remove connections to the failed sensors, and a table is displayed indicating the status of the failed sensors. On clicking 'After Fail' button, shortest path is obtained. On clicking the 'Fail Broadcast Table', link, a table displays the routing for the new path.

7. APPLICATION SCREENSHOTS

The section explains the sequential flow to be followed to find the shortest path before and after sensor failures in the application. It explains the web flow, with the actions that the user should perform to obtain the shortest path. It also explains how the performance metrics are obtained by monitoring the application using JConsole, and application of Confidence Interval on the performance metrics.

7.1. Landing Page

Figure 8 allows the user to enter the number of sensors in the network, the source sensor, destination sensor and the transmission distance. The shortest path is obtained bwtween the source sensor and the destination sensor.



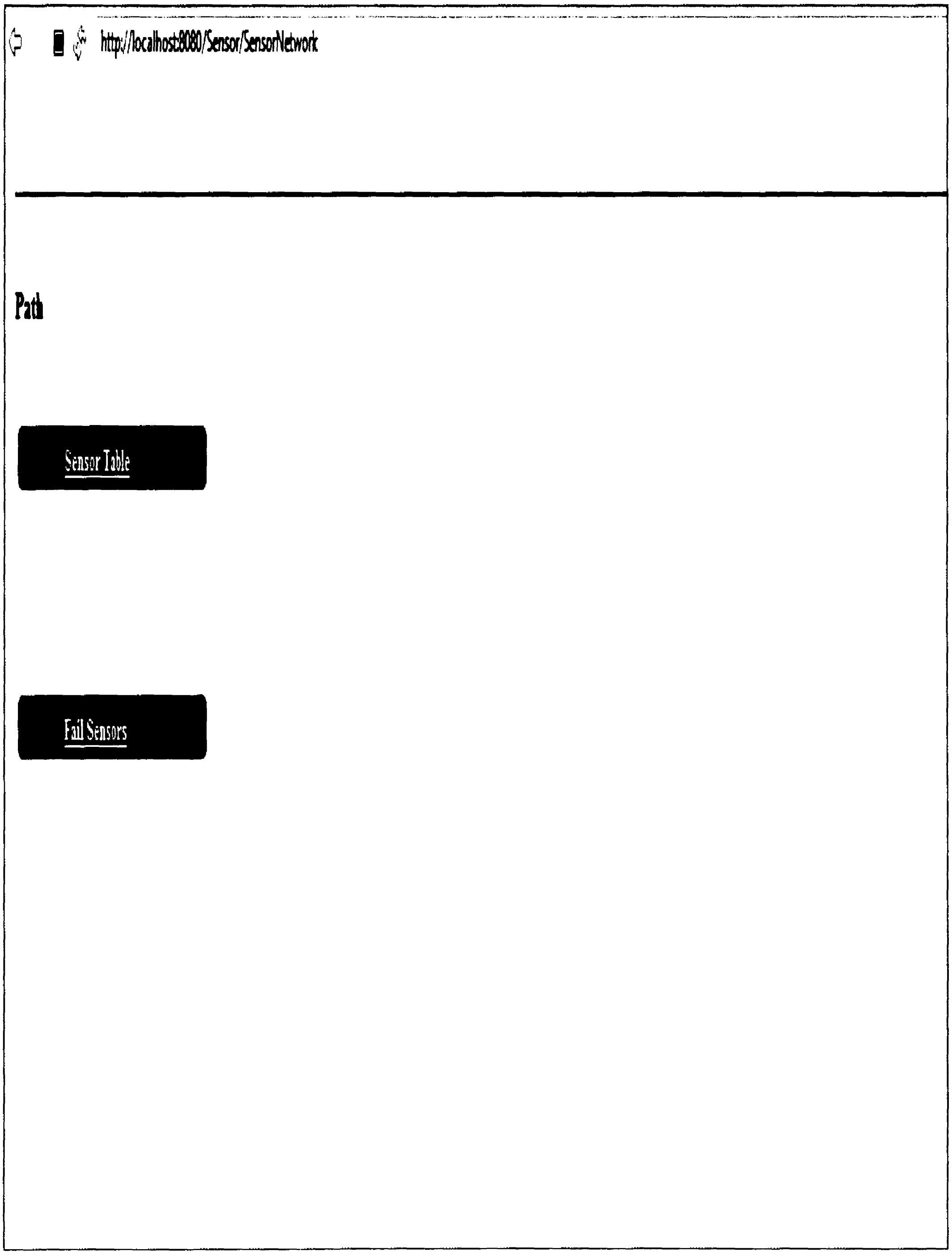
http://localhost:8080/Sensor/

FORM A NETWORK

Enter the number of sensors	28	(must be greater than 2)
Enter the source sensor	sensor8	E.g. sensor1
Enter the destination sensor	sensor17	E.g. sensor54
Transmission distance	18	E.g. 12

Figure 8: Landing Page

On clicking the submit button on the landing page, the user is dispatched to a new page. Figure 9 shows the relationship between the sensors on clicking the "Sensor Table", link, and the shortest path by clicking the "Fail Sensors" link.



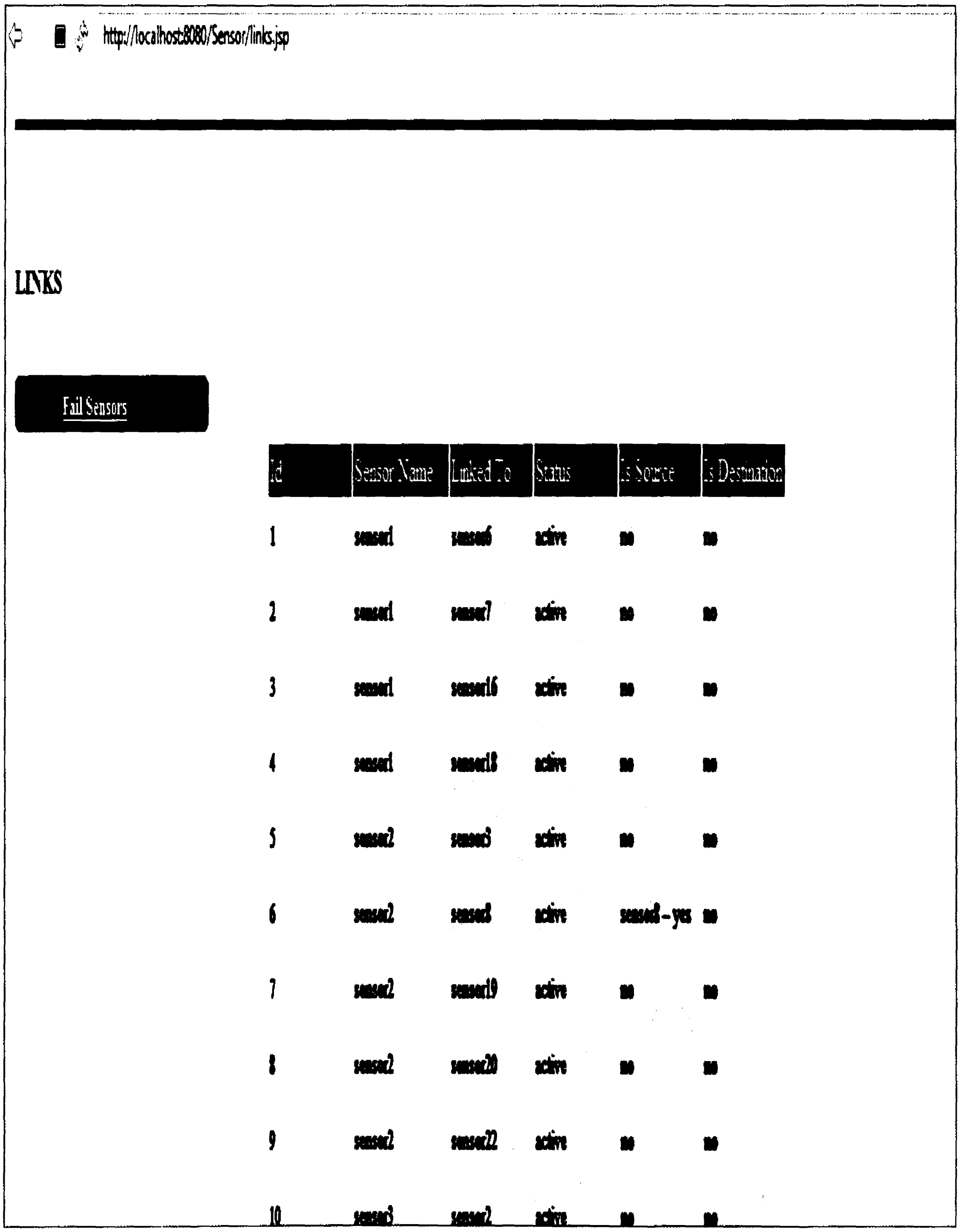
http://localhost:8080/Sensor/SensorNetwork

Path

Figure 9: View Sensor Table/Fail Sensors

7.1.1. Sensor Network

Figure 10 displays the structure of the network, i.e. how the sensors are linked to their neighbors, the status of the sensors and whether it is a source sensor or the destination sensor.



http://localhost:8080/Sensor/links.jsp

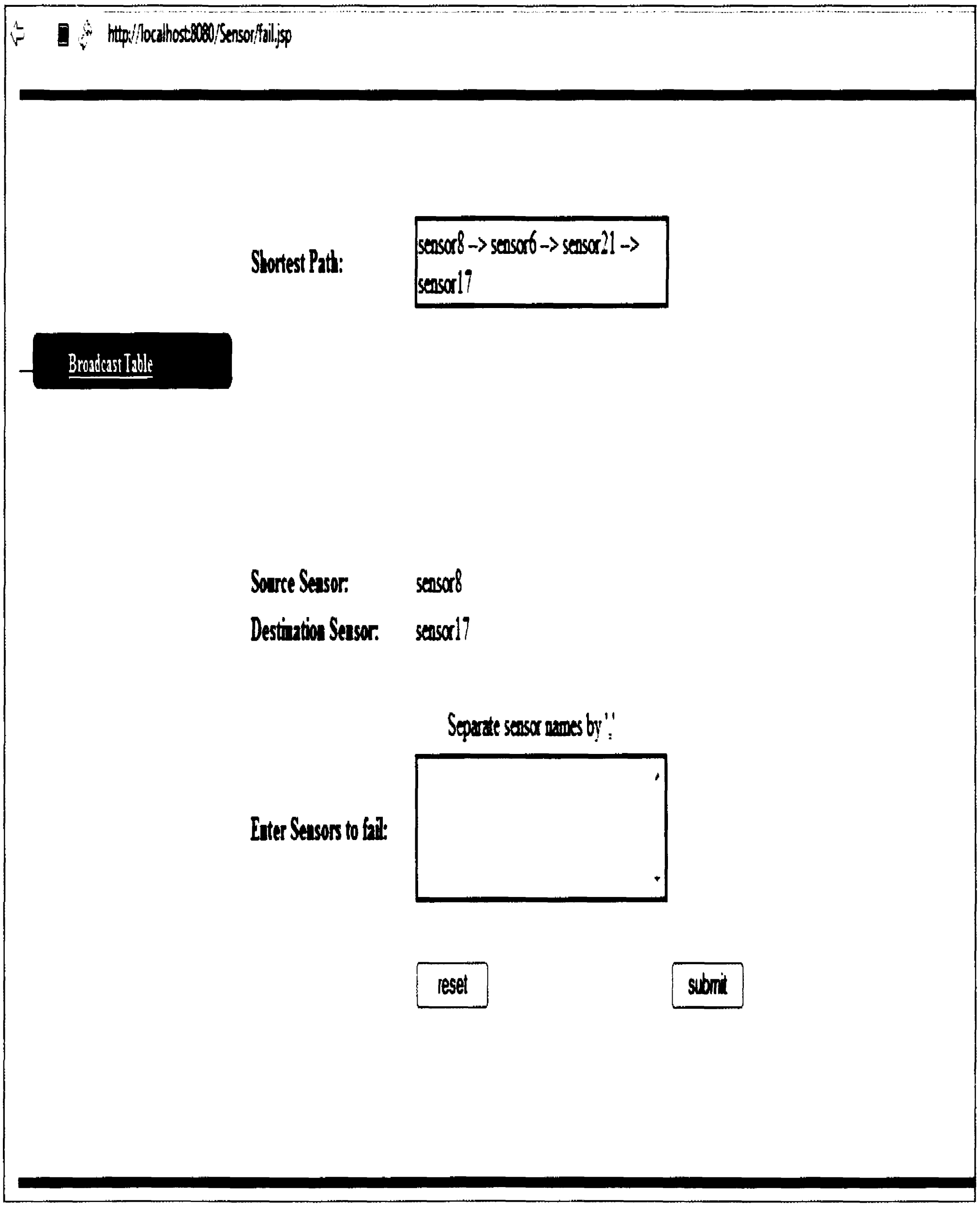
LINKS

Id	Sensor Name	Linked To	Status	Is Source	Is Destination
1	sensor1	sensor6	active	no	no
2	sensor1	sensor7	active	no	no
3	sensor1	sensor16	active	no	no
4	sensor1	sensor18	active	no	no
5	sensor2	sensor3	active	no	no
6	sensor2	sensor8	active	sensor8-yes	no
7	sensor2	sensor19	active	no	no
8	sensor2	sensor20	active	no	no
9	sensor2	sensor22	active	no	no
10	sensor3	sensor2	active	no	no

Figure 10: Sensor Table

7.1.2. Shortest Path Before Failing Sensors

On clicking the “Failure Sensors” link on the previous page, a shortest path is obtained and displayed on the page. Figure 11 allows the user to fail sensors to obtain a new shortest path. Here more than one sensor can be failed, by separating the sensors to be failed by a comma.


http://localhost:8080/Sensor/fail.jsp
Broadcast Table
Separate sensor names by ','
Enter Sensors to fail:
reset
submit

Shortest Path: sensor8 -> sensor6 -> sensor21 -> sensor17

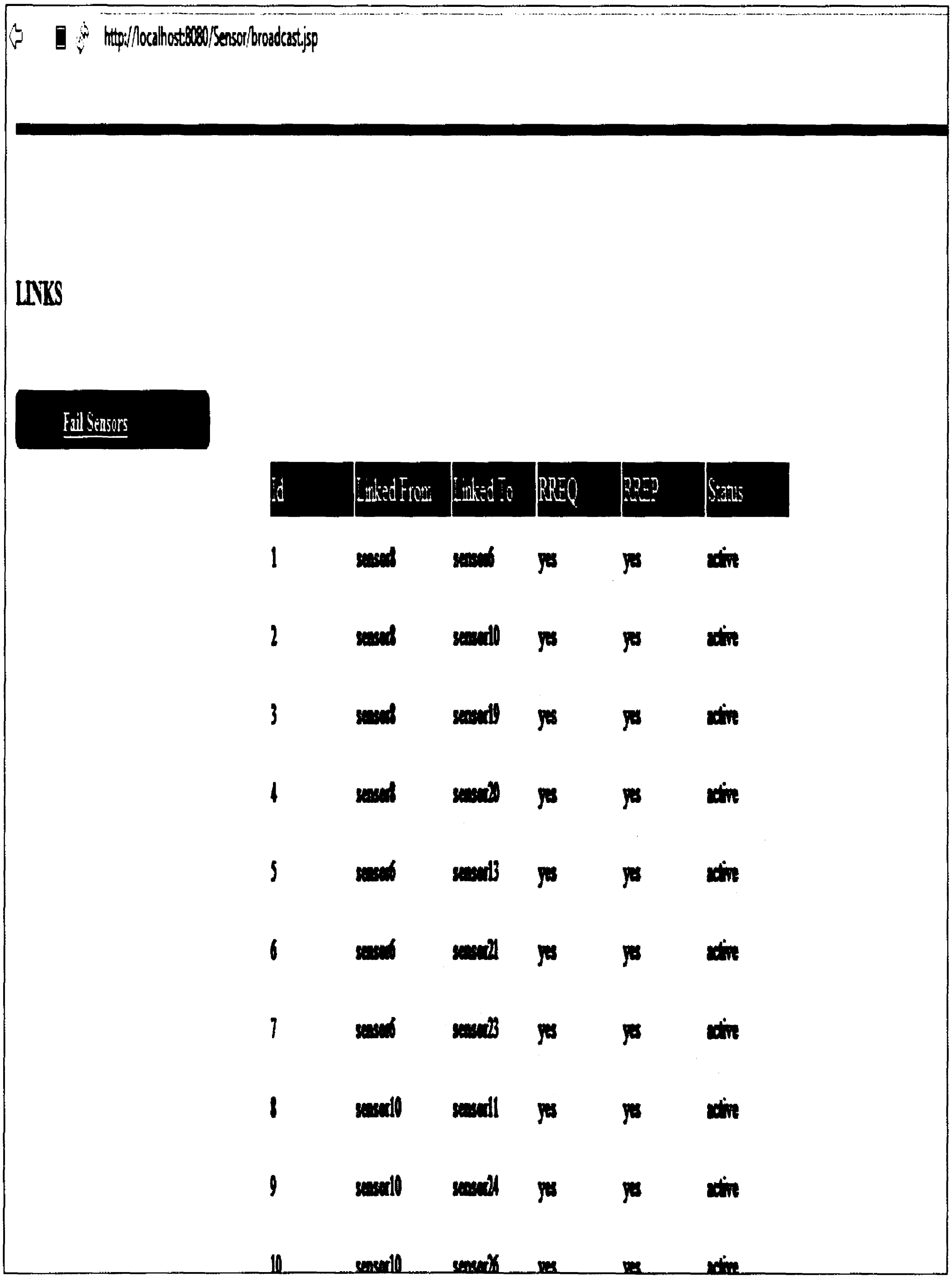
Source Sensor: sensor8

Destination Sensor: sensor17

Figure 11: Shortest Path Before Failing Sensors

7.1.3. Broadcast Table

Figure 12 displays what sensors have broadcasted, the RREQ, RREP and their status.



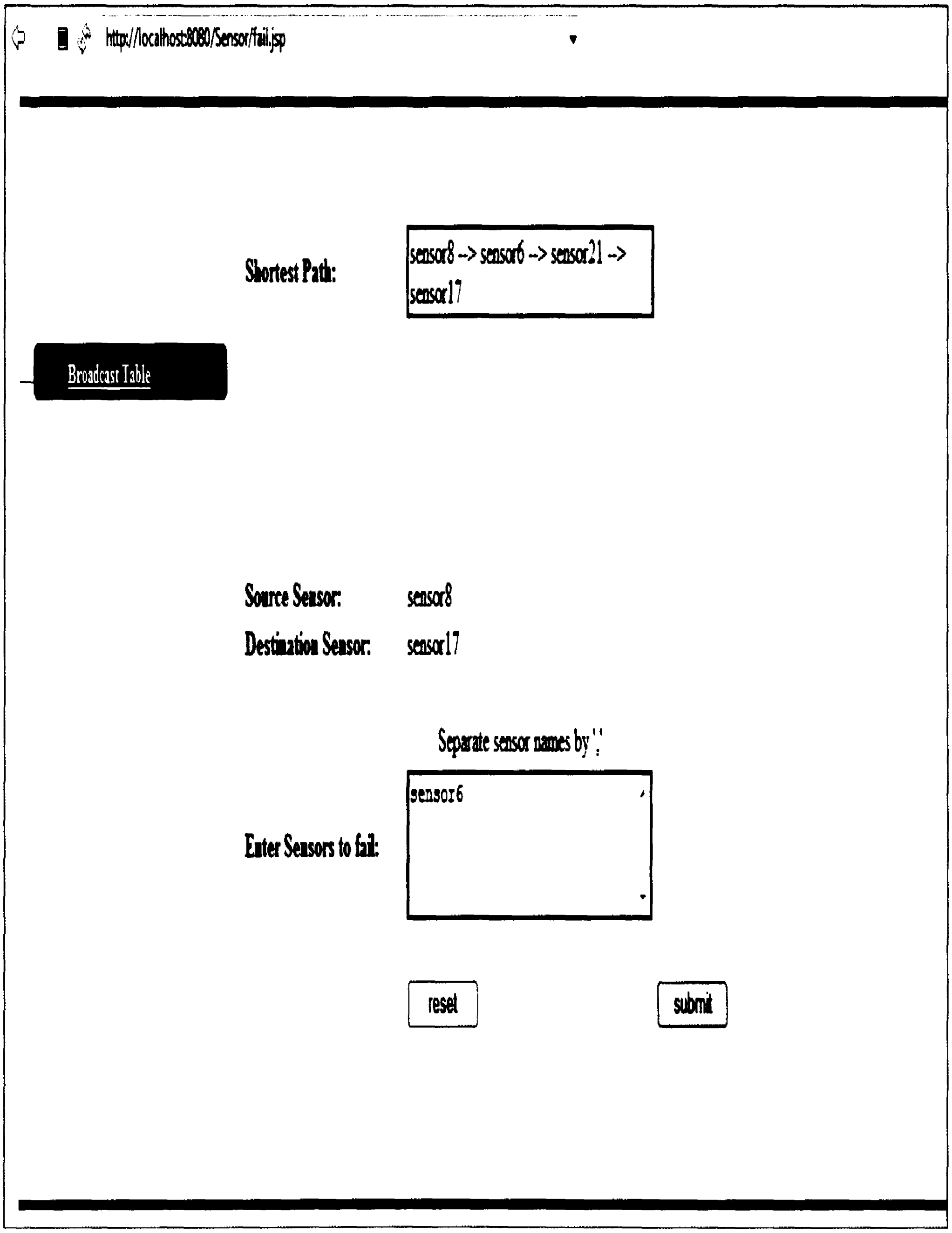
LINKS

Id	Linked From	Linked To	RREQ	RREP	Status
1	sensor8	sensor6	yes	yes	active
2	sensor8	sensor10	yes	yes	active
3	sensor8	sensor19	yes	yes	active
4	sensor8	sensor20	yes	yes	active
5	sensor6	sensor13	yes	yes	active
6	sensor6	sensor21	yes	yes	active
7	sensor6	sensor23	yes	yes	active
8	sensor10	sensor11	yes	yes	active
9	sensor10	sensor24	yes	yes	active
10	sensor10	sensor26	yes	yes	active

Figure 12: Broadcast Table

7.1.4. Fail Sensors

Figure 13 provides user the ability to fail sensors to obtain a new shortest path. Here more than one sensor can be failed by entering the sensors to be failed separated by a comma in the text area provided.


http://localhost:8080/Sensor/fail.jsp

Shortest Path: sensor8 --> sensor6 --> sensor21 --> sensor17

Source Sensor: sensor8

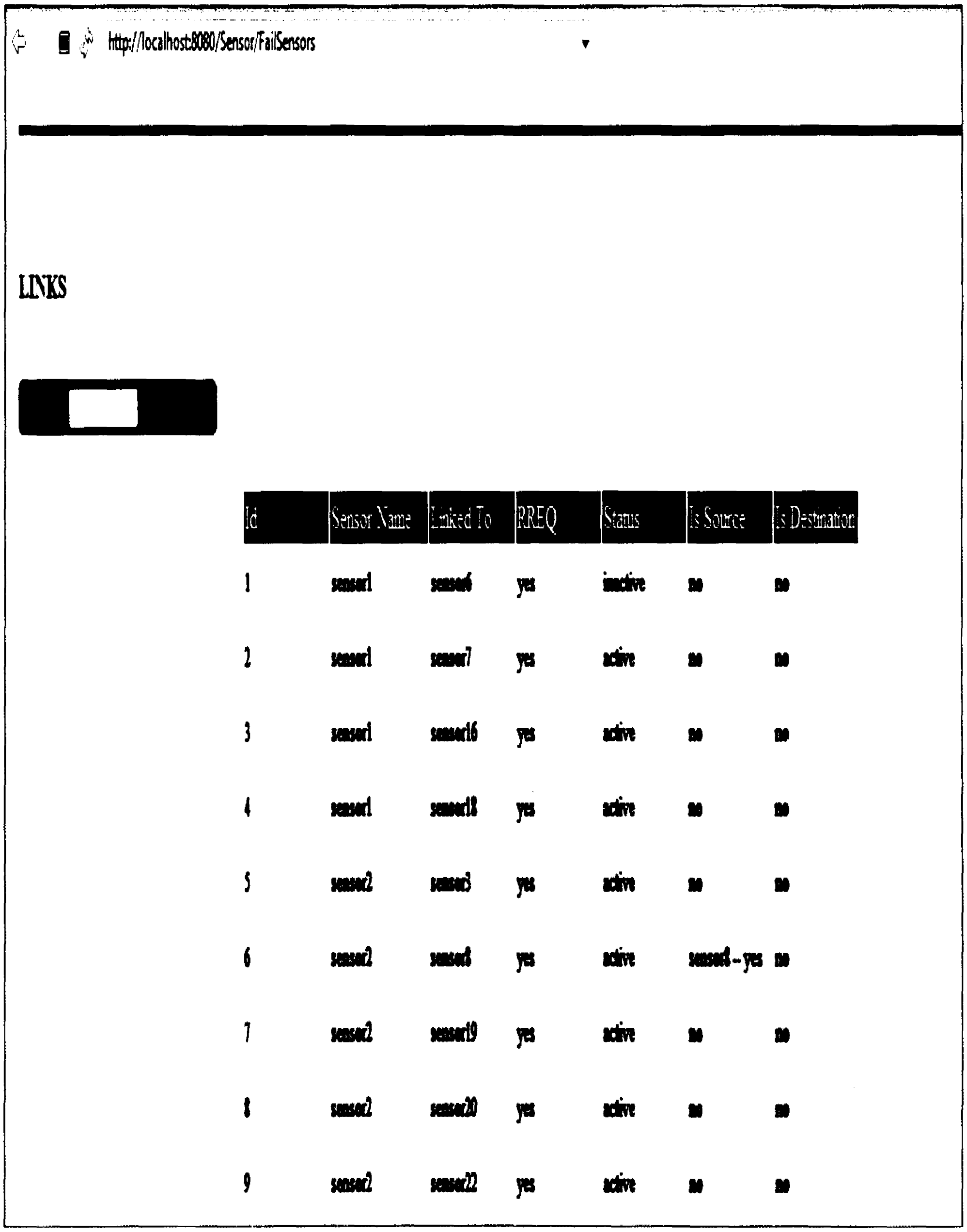
Destination Sensor: sensor17

Separate sensor names by ','

Enter Sensors to fail: sensor6

Figure 13: Fail Sensors

On clicking the submit button in the above page, Figure 14 is displayed listing the modifications made to the network. Here, all the links associated with a failed sensor are changed to inactive. This means that a failed sensor can neither receive a RREQ nor generate a RREP.



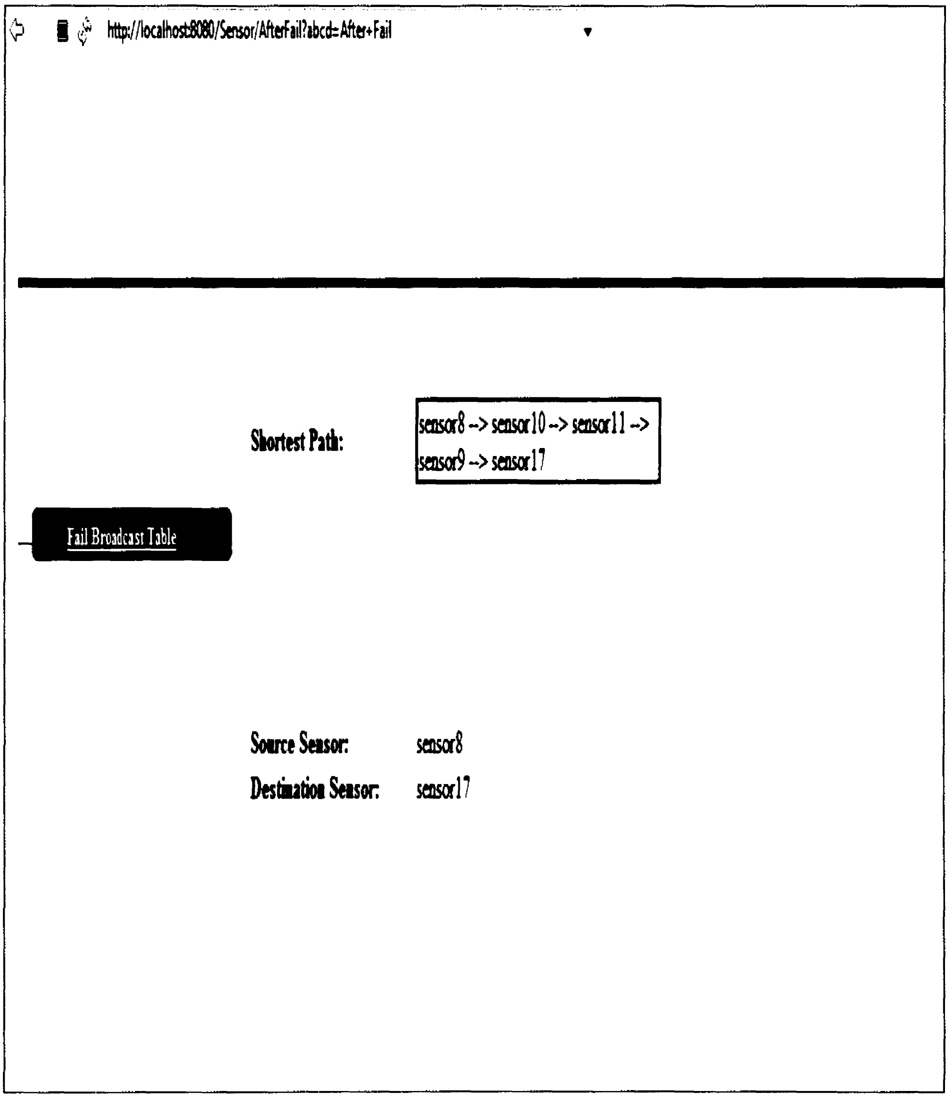
LINKS

Id	Sensor Name	Linked To	RREQ	Status	Is Source	Is Destination
1	sensor1	sensor6	yes	inactive	no	no
2	sensor1	sensor7	yes	active	no	no
3	sensor1	sensor16	yes	active	no	no
4	sensor1	sensor18	yes	active	no	no
5	sensor2	sensor3	yes	active	no	no
6	sensor2	sensor8	yes	active	sensor8 - yes	no
7	sensor2	sensor19	yes	active	no	no
8	sensor2	sensor20	yes	active	no	no
9	sensor2	sensor22	yes	active	no	no

Figure 14: Sensor Status for Failed Sensors

7.1.5. Shortest Path After Failing Sensors

On clicking the "After fail button" on the previous page, Figure 15 displays a new shortest path obtained (using the algorithm) on the screen.



http://localhost:8080/Sensor/AfterFail?abcd=After+Fail

Shortest Path: sensor8 --> sensor10 --> sensor11 --> sensor9 --> sensor17

Fail Broadcast Table

Source Sensor: sensor8
Destination Sensor: sensor17

Figure 15: New Shortest Path

On clicking the "Fail Broadcast Table" link on the previous page, a broadcast table is displayed. The failed sensors represent sensors that do not communicate with other sensors. This table shows what sensors have broadcasted to obtain the shortest path between the source sensor and the destination sensor.

7.2. Monitoring the Application

The application is monitored using JConsole. The CPU Processing Percentage, CPU Process Time, Heap Memory are monitored for each shortest path.

7.2.1. JConsole

JConsole is a JMX compliant tool for monitoring and managing an application by instrumentation of the Java virtual machine to provide information on performance and resource consumption of applications.

JConsole comes with JDK (for best results use JDK version >1.5). If the path for the JDK is set on the System Environment Variable "path", then JConsole can be started by simply typing JConsole on the command window. This opens up a JConsole window showing a list a processes running. Select the Shortest Path process and click connect. This opens up a new Java Monitoring & Management Console Window. This window displays a wide variety of information and graphs, of these only the CPU Processing Percentage, CPU Process Time, Heap Memory are monitored for each shortest path. These values are used for analysis purpose [4].

7.2.1.1. CPU Processing Percentage

It is the amount of CPU Processing Percentage used in finding the shortest path.

7.2.1.2. CPU Process Time

It is the total amount of CPU time (in seconds) consumed by the JVM in finding the shortest path.

7.2.1.3. Heap Memory

It is the runtime memory in Kilo Bytes (KB) used in finding the shortest path. It is the amount of memory consumed by the Java Virtual Machine (JVM), in finding the shortest path between the source senor and the destination sensor.

7.3. Performance Metrics

The performance metrics are used to measure the efficiency of the algorithm in finding the shortest path before and after failing the sensors.

7.3.1. Mean

The arithmetic mean is obtained by adding all values and dividing it by the total number of values.

$$\text{Mean} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Where, ***n*** is the number of values

x_i is the value

i iterates from 1 to *n*

7.3.2. Median

The median is middle value in a list of sorted values.

7.4. Confidence Interval

A confidence interval gives a gauged range of values which is likely to include an unknown population parameter, the estimated range being calculated from a given set of sample data.

Confidence intervals are constructed with a specific confidence level, such as 95% (depends on what the user wants). It means that on sampling the same population numerous times, and making interval estimates each time, the sampled intervals would fit into the true population parameter bordering in 95% of the cases.

Confidence intervals are constructed on the basis of a given dataset: *x* denotes the set of observations in the dataset, and *X* is used when considering the outcomes that might have been observed from the same population, where *X* is treated as a random variable whose observed outcome is $X=x$. A confidence interval for the given data set is defined as the interval ($u(x)$, $v(x)$). Suppose if the CI interval estimate on quantity is w. The interval ($u(x)$, $v(x)$) closest to what a confidence interval for w would be, relates to the properties of the set of random intervals given by ($u(X)$, $v(X)$) i.e. treating the end-points as random variables. This property is the coverage probability or the probability c that the random interval includes w. [15]

$$c = \Pr(u(X) < w < v(X)).$$

Here the endpoints $U = u(X)$ and $V = v(X)$ are statistics (i.e., observable random variables) which are derived from values in the dataset. The random interval is (U, V) [15].

7.4.1. Intervals for Random Outcomes

Consider an additional single-valued random variable Y which may or may not be statistically dependent on X. Then the rule for constructing the interval ($u(x)$, $v(x)$) provides a confidence interval for the as-yet-to-be observed value y of Y if

$$\Pr_{\theta,\varphi}(u(X) < Y < v(X)) = 1 - \alpha \text{ for all } (\theta, \varphi).$$

Here, $\Pr_{\theta,\varphi}$ is used to indicate the probability over the joint distribution of the random variables (X, Y) when this is characterized by parameters (θ, φ) [17].

7.4.2. Approximate Confidence Intervals

For non-standard applications it is sometimes not possible to find rules for constructing confidence intervals [15] that have exactly the required properties. But practically useful intervals can still be found. The coverage probability c (θ, φ) for a random interval is defined by [17].

$$\Pr_{\theta,\phi}(u(X) < \theta < v(X)) = c(\theta, \phi)$$

The rule for constructing the interval may be accepted as providing a confidence interval to an acceptable level of approximation [15].

$$c(\theta,\phi) \approxeq 1-\alpha \text{ for all } (\theta,\phi)$$

7.4.3. CI for the Performance Metrics

CI is calculated for the number of hops, CPU processing %, CPU process time, and heap memory. To get an impression of the expectation μ, it is sufficient to give an estimate. The appropriate estimator is the samples mean [15]:

$$\hat{\mu} = \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

Determining the endpoints by considering that the sample mean X from a normally distributed sample is also normally distributed, with the same expectation μ, but with standard error $\sigma/\sqrt{n}$ = 0.5. By standardizing, we get a random variable [16]

$$Z = \frac{\bar{X}-\mu}{\sigma/\sqrt{n}} = \frac{\bar{X}-\mu}{0.5}$$

[Depending on the parameter μ to be estimated, with a standard normal distribution independent of the parameter μ, it is possible to find numbers $-z$ and z, independent of μ, where Z lies in between with probability 1 – α, a measure of how confident we want to be. We take 1 – α = 0.95. So we have [16]:

$$P(-z \leq Z \leq z) = 1-\alpha = 0.95.$$

The number z follows from the cumulative distribution function [15].

This might be interpreted as: with probability 0.95 we will find a confidence interval in which we will meet the parameter μ between the stochastic endpoints. [15]

$$\bar{X} - 0.98$$

$$\bar{X} + 0.98.$$

Confidence intervals are constructed with a specific confidence level, such as 95% (depends on what the user wants). It means that on sampling the same population numerous times, and making interval estimates each time, the sampled intervals would fit into the true population parameter bordering in 95% of the cases.

Confidence intervals are constructed on the basis of a given dataset: x denotes the set of observations in the dataset, and X is used when considering the outcomes that might have been observed from the same population, where X is treated as a random variable whose observed outcome is $X = x$. A confidence interval for the given data set is defined as the interval ($u(x)$, $v(x)$). Suppose if the CI interval estimate on quantity is w. The interval ($u(x)$, $v(x)$) closest to what a confidence interval for w would be, relates to the properties of the set of random intervals given by ($u(X)$, $v(X)$) i.e. treating the end-points as random variables. This property is the coverage probability or the probability c that the random interval includes w. [15]

8. EXPERIMENTS AND RESULTS

This chapter provides an experimental analysis of the performance of the network in finding a shortest path, based on the Hop Count, CPU Process Time, CPU Processing % and Heap Memory monitored for each shortest path.

The CPU Process Time, CPU Processing % and Heap Memory are monitored for each shortest path using JConsole, while the number of hops in the shortest path is obtained using the algorithm in section 4. Refer sections 7.1.1.1, 7.1.1.2 and 7.1.1.3 for the definitions of CPU Processing %, CPU Process Time and Heap Memory.

The performance metrics provide accurate results in finding the shortest path. The number hops, represent the number of sensors the request has to pass through to reach the destination sensor. The CPU Processing %, gives the amount of CPU consumed in finding the shortest path. The CPU Processing % may vary depending on the amount of CPU available while finding the shortest path. The amount of memory consumed by the JVM varies, depending on the number of sensors the request has to pass through in subsequent iterations of the algorithm.

The CPU time increases since the RREQ has to pass through multiple sensors to reach the destination. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm. When the number of sensors through which the RREQ has to pass through decreases, the total heap memory consumed by the JVM in finding the shortest path also decreases. The total number of hops also decreases when the RREQ has to pass through less number of sensors. This process proceeds until the destination sensor receives the request. On obtaining the shortest path, multiple sensors in the shortest path can be failed to obtain the new shortest path. Performance metrics are obtained for each shortest path. The data obtained is analyzed in compared with results for the same sample number.

In the tables below, column one represents the source sensor, column two represents the destination sensor, column three represents the total number of hops in the shortest path between the source and destination sensors, column four represents the CPU Processing %, column five represents the CPU Process Time and column six represents the Heap Memory involved in identifying the shortest path.

Table 4 shows the experimental results for 100 sensors distributed in a geographical area, with no sensor failures.

Table 5 shows the experimental results for 100 sensors distributed in a geographical area with 1 sensor failure.

Table 6 shows the experimental results for 100 sensors distributed in a geographical area with 2 sensor failures.

Table 7 shows the experimental results for 500 sensors distributed in a geographical area with no sensor failures.

Table 8 shows the experimental results for 500 sensors distributed in a geographical area with 3 sensor failures.

Table 9 shows the experimental results for 5000 sensors distributed in a geographical area with no sensor failures.

Table 10 shows the experimental results for 5000 sensors distributed in a geographical area with 5 sensor failures.

Table 11 shows the experimental results for 10000 sensors distributed in a geographical area with no sensor failures.

Table 12 shows the experimental results for 10000 sensors distributed in a geographical area with 10 sensor failures.

Table 4: Experimental Results for 100 Sensors with no Sensor Failure

S.No	Source Sensor	Destination Sensor	No of Hops	CPU processing %	CPU Process Time	Heap Memory Kbytes
1	sensor1	sensor100	15	0.25	0.873	2197
2	sensor4	sensor100	28	0.21	0.951	1810
3	sensor5	sensor100	21	0.23	0.922	1973
4	sensor8	sensor100	19	0.22	1.076	1746
5	sensor8	sensor100	34	0.26	0.876	1656
6	sensor11	sensor100	27	0.27	0.976	2100
7	sensor79	sensor100	41	0.24	0.879	1963
8	sensor18	sensor100	23	0.19	0.819	1533
9	sensor21	sensor100	24	0.12	0.761	1389
10	sensor32	sensor100	34	0.16	0.755	1745
11	sensor72	sensor100	54	0.26	0.856	1981
12	sensor73	sensor100	43	0.23	0.943	1961
13	sensor36	sensor100	23	0.25	0.876	1841
14	sensor39	sensor100	31	0.31	1.211	1746
15	sensor80	sensor100	42	0.25	0.911	1989
Total			459	3.45	13.6845	27630
Mean			30.6	0.23	0.91233	1842
Median			28	0.24	0.879	1841
Standard Deviation			10.7291	0.04645	0.1157	214.4151
Confidence Interval			5.43	0.02	0.06	108.51
Range			25.17 to 36.03	0.21 to 0.25	0.85 to 0.97	1733.49 to 1950.51

It can be seen that the CPU Processing %, Heap Memory and the hops vary with sample number when no sensors are failed. Since, each time, the algorithm is run on a newly formed network, the number of hops, CPU Processing %, Heap Memory vary, depending on the density of the network. Greater number hops, signifies a densely populated sensor network for a network with the same sample number. The number of hops increases in a densely populated network.

Table 5: Experimental Results for 100 Sensors with 1 Sensor Failure

S.No	Source Sensor	Destination Sensor	No. of Hops	CPU processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	sensor1	sensor100	33	0.23	0.876	1897
2	sensor4	sensor100	34	0.11	0.679	1712
3	sensor5	sensor100	41	0.26	0.887	1972
4	sensor8	sensor100	24	0.12	0.676	1788
5	sensor8	sensor100	54	0.09	0.581	1689
6	sensor11	sensor100	34	0.24	0.835	1886
7	sensor79	sensor100	56	0.14	0.516	1635
8	sensor18	sensor100	45	0.15	0.621	1661
9	sensor21	sensor100	39	0.11	0.53	1588
10	sensor32	sensor100	63	0.06	0.542	1790
11	sensor72	sensor100	59	0.16	0.646	1782
12	sensor73	sensor100	34	0.03	0.379	1594
13	sensor36	sensor100	43	0.02	0.332	1545
14	sensor39	sensor100	36	0.21	0.826	1872
15	sensor80	sensor100	49	0.23	0.876	1890
Total			644	2.16	9.8	26301
Mean			42.9333	0.144	0.65	1753.4
Median			41	0.14	0.64	1782
Standard Deviation			11.2088	0.0771	0.1791	132.7413
Confidence Interval			5.67	0.04	0.09	67.18
Range			37.26 to 48.6	0.1 to 0.18	0.56 to 0.74	1686.22 to 1820.58

Comparing the results in Table 4 and Table 5, it is seen that the number of Hops, the CPU Processing % and CPU Processing Time decreases with a sensor failure. With less number of sensors in the network, the algorithm has sent a RREQ to a number of sensors in the network. When a sensor is failed, the algorithm, starts finding the shortest path by disabling the failed sensor. Since, each sensor broadcasts only once, the numbers of Hops, CPU Processing %, CPU Process Time and Heap Memory decrease in this case.

Figure 16 shows the graphical representation of the Number of Hops vs. Sample Number. It can be seen that the number of Hops decreases with sample number on a sensor failure. Since, only one sensor is failed, the algorithm takes less time to find an alternative shortest path by broadcasting RREQ from the sensors, to its neighbors from the point where it has previously stopped.

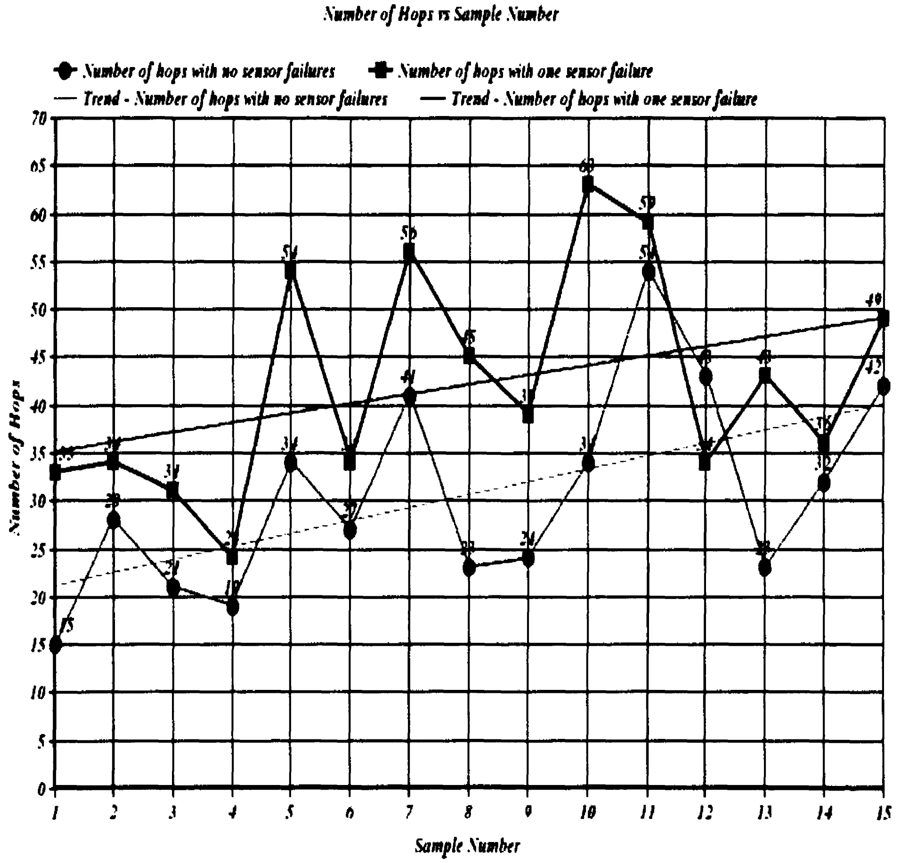


Figure 16: Graph of Number of Hops vs. Sample Number for 100 Sensors with 0 and 1 Sensor Failure

Figure 17 shows the graphical representation of the CPU Processing % vs. Sample Number. It can be seen that the CPU Processing % decreases with Sample Number on a sensor failure.

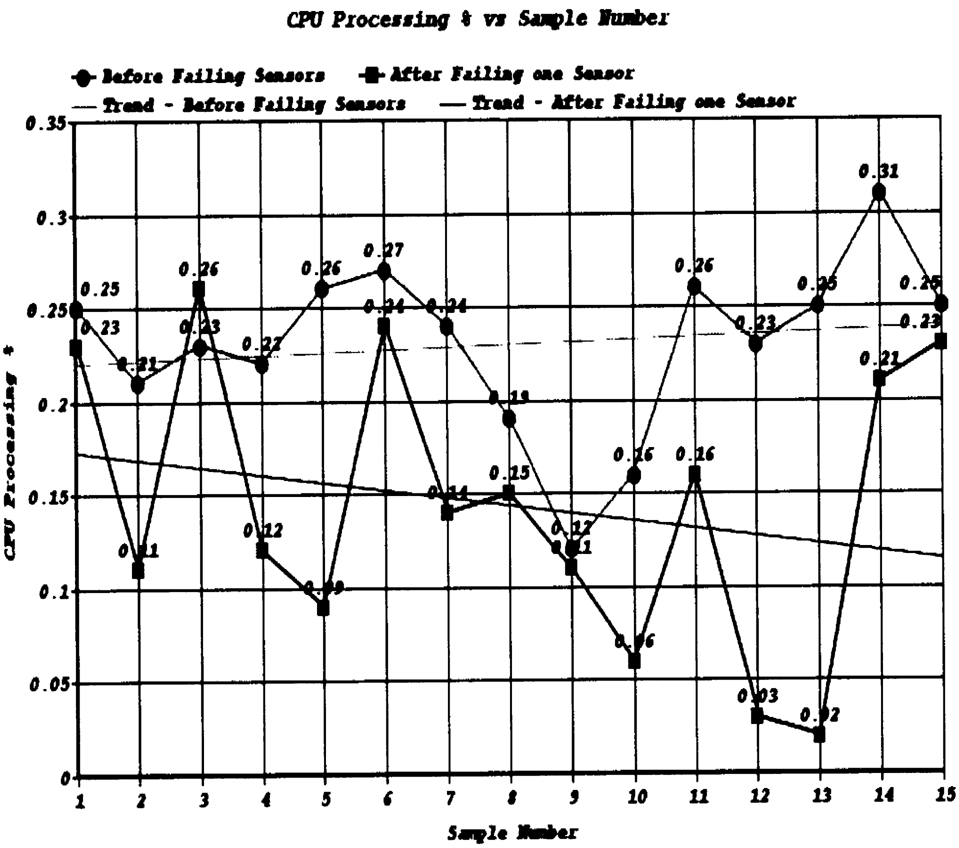


Figure 17: Graph of CPU Processing % vs. Sample Number for 100 Sensors with 0 and 1 Sensor Failure

It can be seen that the CPU Processing % decreases with sample number on a sensor failure. With a single sensor failure, the algorithm takes less time to find an alternative shortest path by broadcasting RREQ from the sensors, to its neighbors. This results in a decrease in the amount of CPU Processing % used. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm. The number of hops is this case is likely to decrease with a decrease in the amount of heap memory consumed.

Comparing the results in table 5 and 6, it is seen that the number of Hops, the CPU Processing %, the CPU Process Time and the Heap Memory increases with two sensor failures.

Table 6: Experimental Results for 100 Sensors with 2 Sensor Failures

S.No	Source Sensor	Destination Sensor	No of Hops	CPU processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	sensor1	sensor100	45	0.31	0.889	1887
2	sensor4	sensor100	56	0.24	0.971	1934
3	sensor5	sensor100	49	0.48	0.987	1991
4	sensor8	sensor100	35	0.17	0.996	1895
5	sensor8	sensor100	56	0.34	0.991	1974
6	sensor11	sensor100	61	0.64	0.897	1988
7	sensor79	sensor100	63	0.31	0.982	1964
8	sensor18	sensor100	52	0.64	0.941	1841
9	sensor21	sensor100	42	0.59	0.899	1982
10	sensor32	sensor100	68	0.23	0.993	1986
11	sensor72	sensor100	66	0.45	0.842	1993
12	sensor73	sensor100	40	0.29	0.986	1987
13	sensor36	sensor100	48	0.56	0.932	1982
14	sensor39	sensor100	40	0.45	0.986	1984
15	sensor80	sensor100	58	0.68	0.899	1976
Total			799	5.7	14.19105	29364
Mean			51.9333	0.38	0.94607	1957.6
Median			52	0.34	0.971	1982
Standard Deviation			10.2572	0.1625	0.0497	46.7176
Confidence Interval			5.19	0.08	0.03	23.64
Range			46.74 to 57.12	0.3 to 0.46	0.92 to 0.98	1933.96 to 1981.24

With two failed sensors, the algorithm takes more time to find an alternative shortest path by broadcasting RREQ from the sensors, to its neighbors. This results in increase in the number of Hops, the CPU Processing %, the CPU Process Time and the Heap Memory.

It is concluded from the above results, the number of Hops, CPU Processing % and CPU Process Time increases with two sensor failures. The total heap memory increases as observed in the results of Table 5 and Table 6 due to the amount of memory consumed while finding an alternative shortest path with two sensor failures.

Figure 18 shows the graphical representation of the Number of Hops vs. Sample Number. It can be seen that the number of Hops increases with Sample Number on two sensor failures.

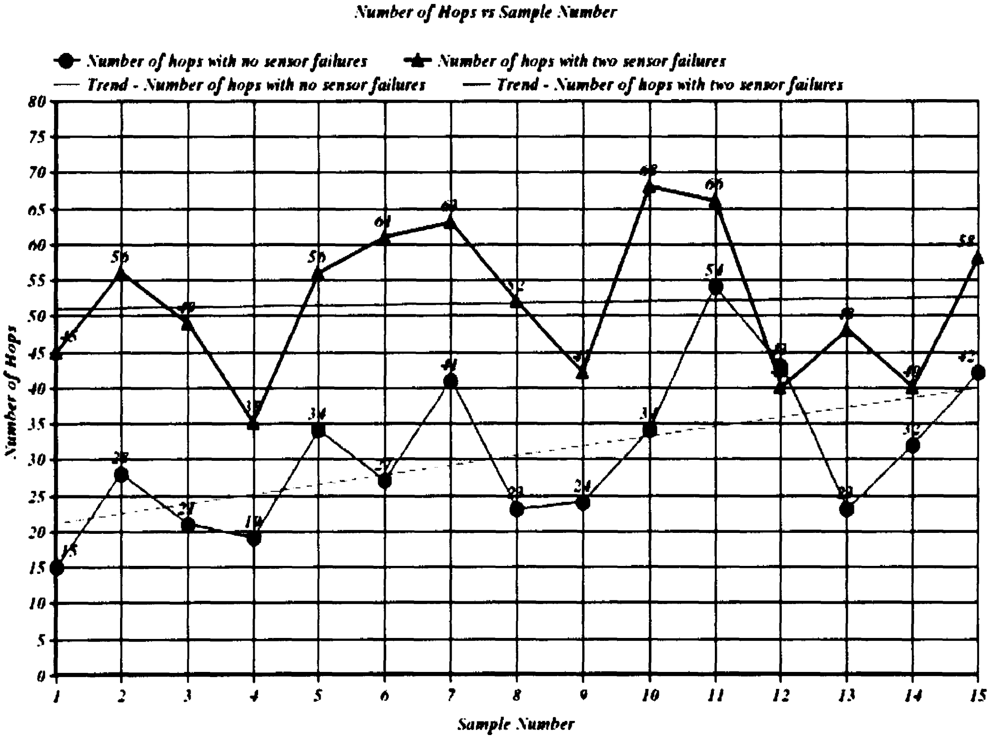


Figure 18: Graph of Number of Hops vs. Sample Number for 100 Sensors with 0 and 2 Sensor Failures

It can be seen that the number of Hops increases with sample number on two sensor failures. With two failed sensors; the algorithm traverses through the sensors to find an alternative shortest path by broadcasting RREQ from the sensors, to reach the destination. This results in increased Hop count.

Figure 19 shows the graphical representation of the Number of Hops vs. Sample Number. It can be seen that the number of Hops increases with Sample Number on two sensor failures.

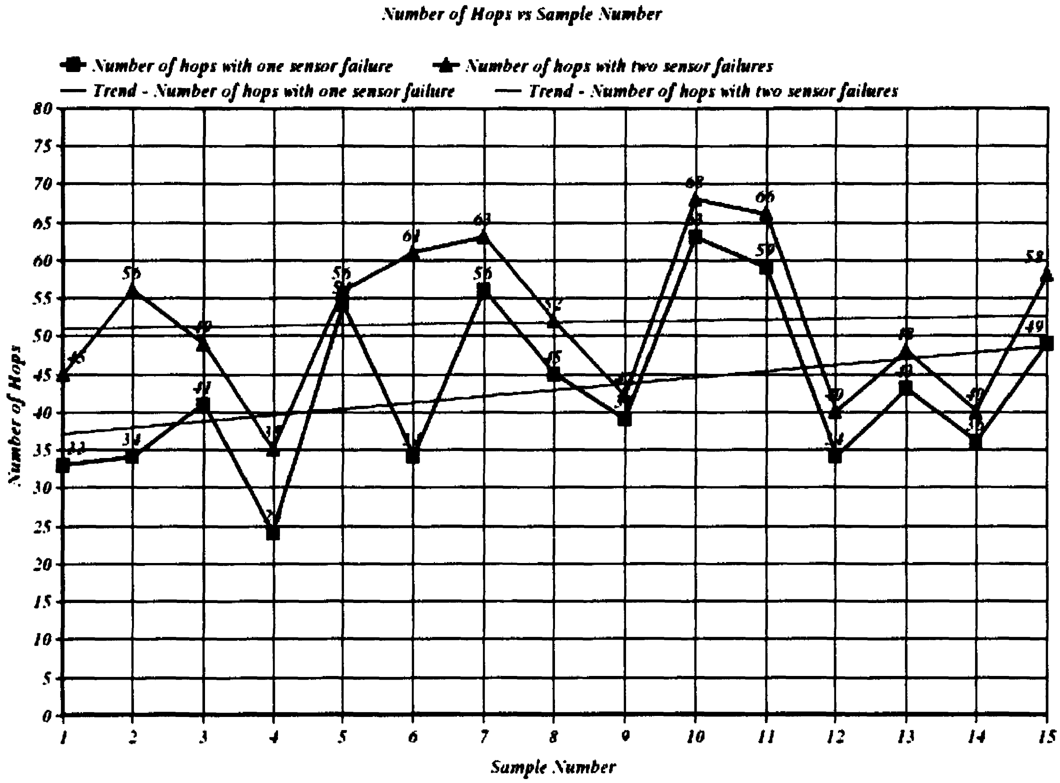


Figure 19: Graph of Number of Hops vs. Sample Number for 100 Sensors with 1 and 2 Sensor Failures

It can be seen that the number of Hops increases with sample number on two sensor failures. With two failed sensors, the algorithm traverses through more sensors to find an alternative shortest path by broadcasting RREQ from the sensors, to its neighbors. The number of Hops increases since the RREQ has to pass through multiple sensors to reach the destination.

The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during the subsequent iteration of the algorithm.

Figure 20 shows the graphical representation of the Number of Hops vs. Sample Number. It can be seen that the number of Hops increases with Sample Number on two sensor failures. The number of hops is this case is likely to decrease with a decrease in the amount of heap memory consumed.

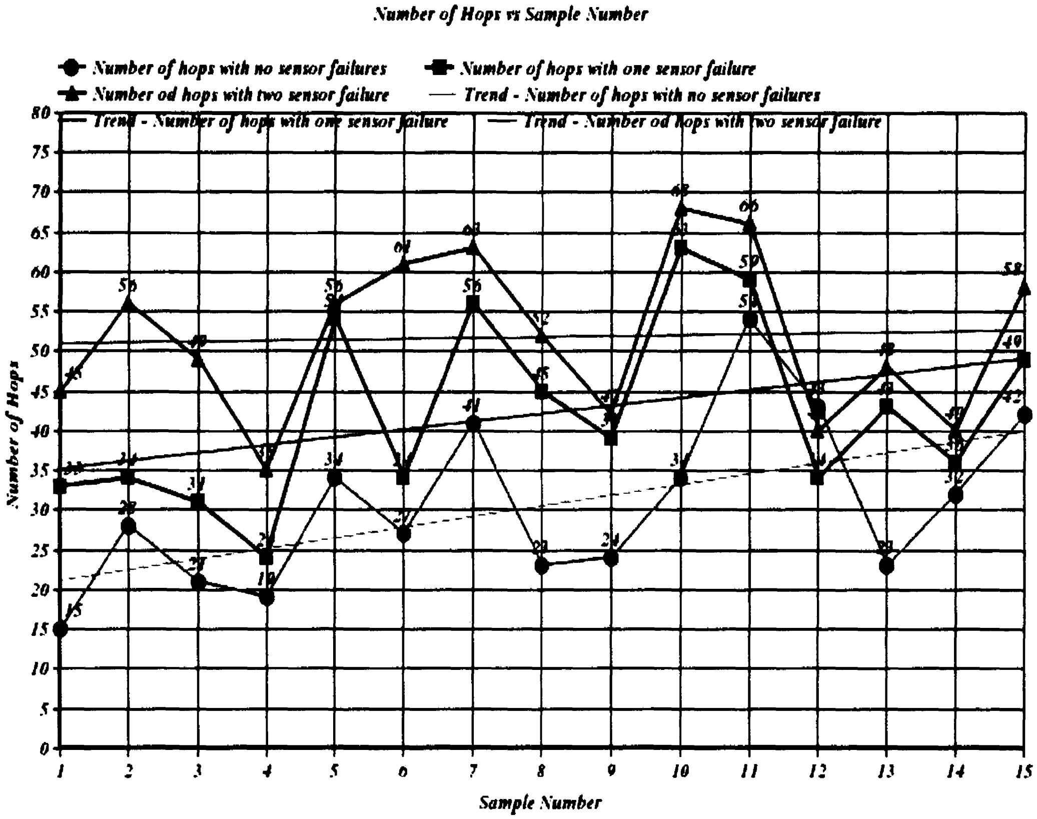


Figure 20: Graph of Number of Hops vs. Sample Number for 100 Sensors with 0, 1 and 2 Sensor Failures

It can be seen that the number of Hops increases with sample number on two sensor failures. With two failed sensors, the algorithm traverses through more sensors to find an alternative shortest path by broadcasting RREQ from the sensors, to its neighbors. The number of Hops increases since the RREQ has to pass through multiple sensors to reach the destination.

The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm.

Figure 21 shows the graphical representation of the CPU Processing % vs. Sample Number. It can be seen that the CPU Processing % increases with Sample Number on two sensor failures. The number of hops also increases in this case.

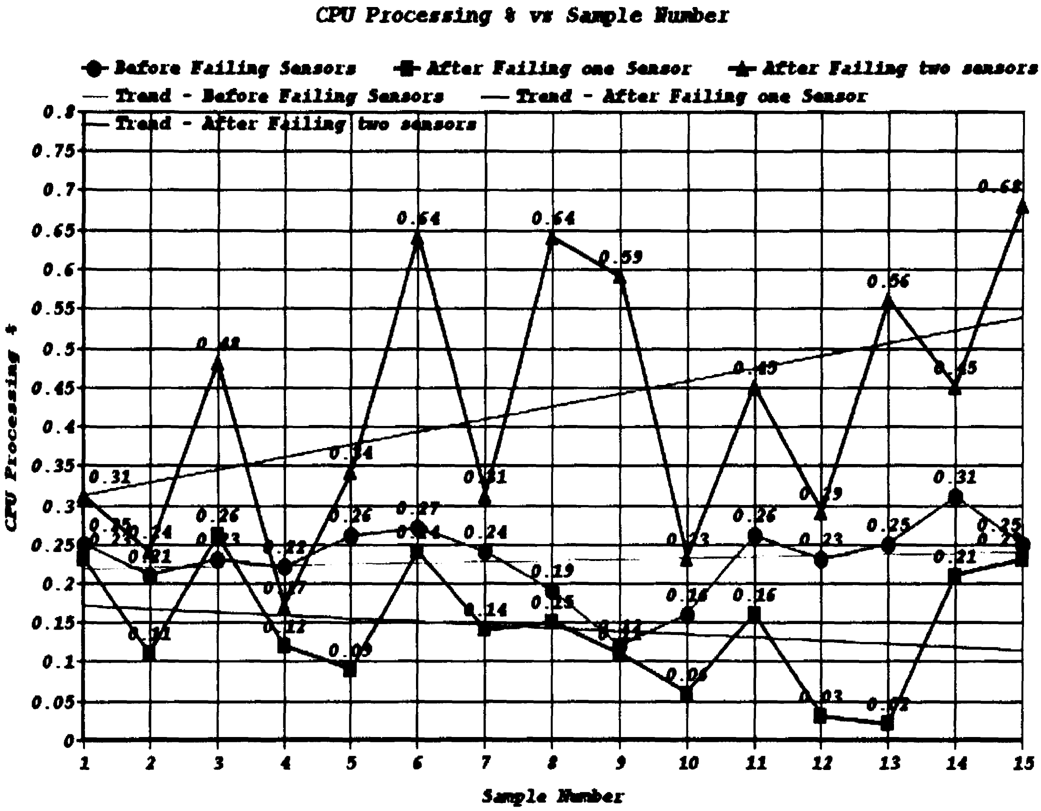


Figure 21: Graph of CPU Processing % vs. Sample Number for 100 Sensors with 0, 1 and 2 Sensor Failures

It can be seen that the CPU Processing % increases with sample number on two sensor failures. With two failed sensors, the algorithm traverses through more number of sensors to find an alternative shortest path, by broadcasting RREQ from the sensors, to its neighbors. The CPU Processing % increases since the RREQ has to pass through multiple sensors to reach the destination. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm.

Figure 22 shows the graphical representation of the CPU Process Time vs. Sample Number. It can be seen that the CPU Process Time increases with Sample Number on two sensor failures.

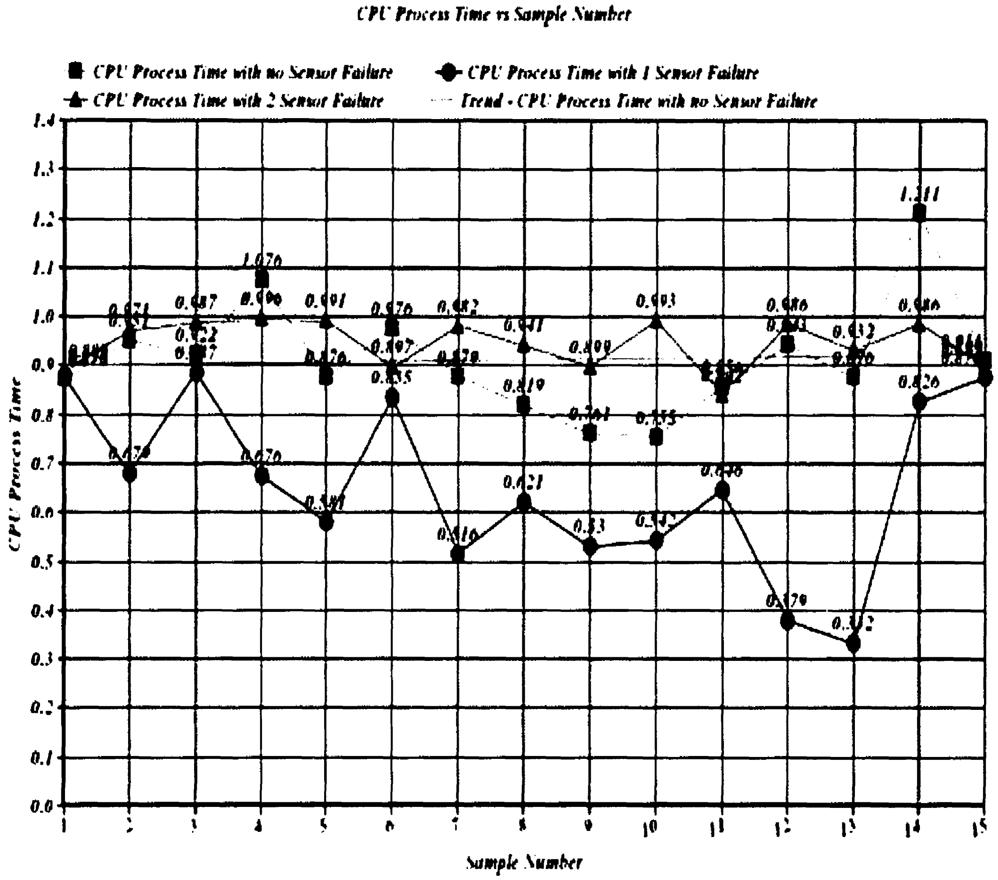


Figure 22: Graph of CPU Process Time vs. Sample Number for 100 Sensors with 0, 1 and 2 Sensor Failures

It can be seen that the CPU Process Time increases with sample number on two sensor failures. With two sensor failures, the algorithm has to traverse through more number of sensors to find an alternative shortest path. The CPU Process Time increases since the CPU consumes more time to send a RREQ to multiple sensors to reach the destination. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm.

Figure 23 shows the graphical representation of the Heap Memory vs. Sample Number. It can be seen that the Heap Memory increases with Sample Number on two sensor failures.

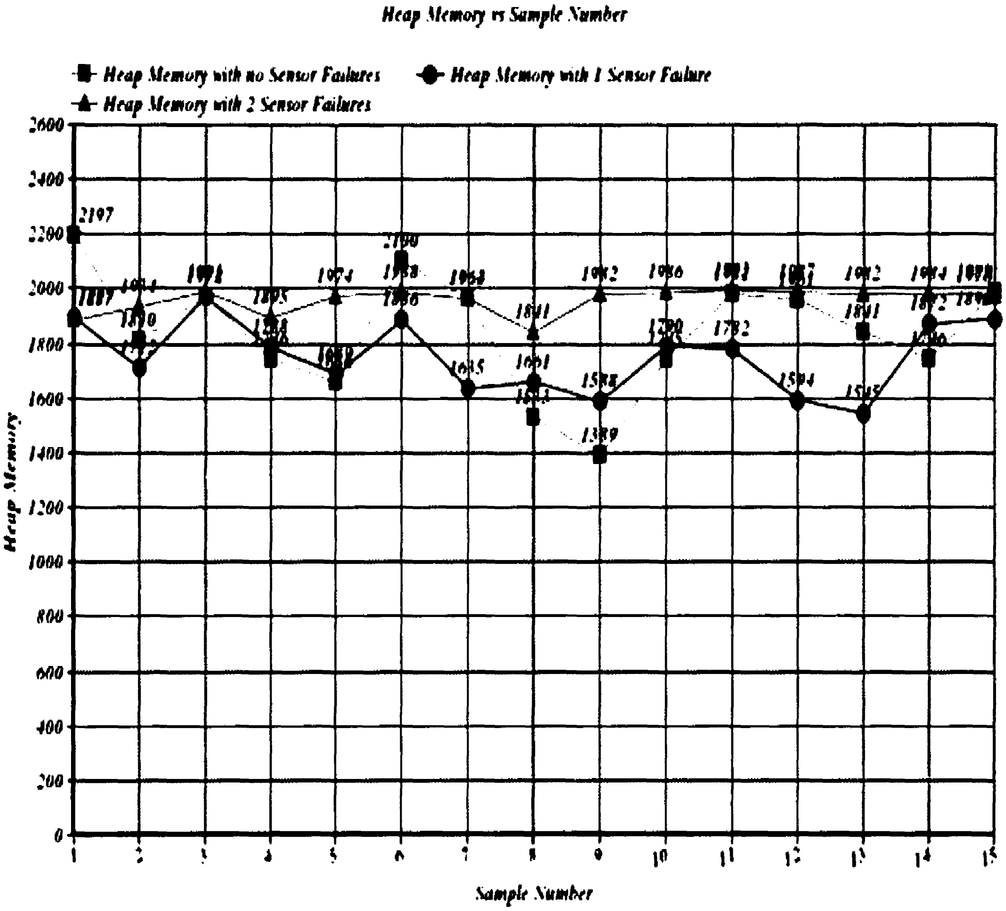


Figure 23: Graph of Heap Memory vs. Sample Number for 100 Sensors with 0, 1 and 2 Sensor Failures

It can be seen that the Heap Memory increases with sample number on two sensor failures. With two sensor failures, the algorithm has to traverse through more number of sensors to find an alternative shortest path. The Heap Memory increases since the CPU consumes more memory to send a RREQ to multiple sensors to reach the destination.

The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm to reach the destination sensor. The number of hops is this case is likely to decrease with a decrease in the amount of heap memory consumed.

Table 7: Experimental Results for 500 Sensors with no Sensor failure

S.No	Source Sensor	Destination Sensor	No of Hops	CPU processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	Sensor11	Sensor500	73	0.48	1.212	2111
2	Sensor43	sensor500	78	0.42	1.216	2214
3	sensor59	Sensor500	82	0.41	1.222	2421
4	sensor81	Sensor500	72	0.4	1.098	1976
5	Sensor91	Sensor500	89	0.48	1.245	2931
6	sensor110	Sensor500	80	0.45	1.221	2511
7	Sensor10	Sensor500	76	0.42	1.108	2033
8	Sensor190	Sensor500	65	0.4	1.189	2285
9	sensor210	Sensor500	69	0.43	1.12	2219
10	Sensor225	Sensor500	92	0.45	1.321	2687
11	Sensor250	Sensor500	89	0.48	1.301	2671
12	Sensor255	Sensor500	94	0.45	1.298	2634
13	Sensor300	Sensor500	86	0.41	1.311	2224
14	Sensor231	Sensor500	56	0.4	1.198	1957
15	Sensor56	Sensor500	102	0.48	1.331	2620
Total			1203	6.5599	18.391	35494
Mean			80.2	0.4373	1.22607	2366.26
Median			80	0.45	1.221	2285
Standard Deviation			12.1902	0.0317	0.0767	297.8989
Confidence Interval			6.17	0.02	0.04	150.75
Range			74.03 to 86.37	0.42 to 0.46	1.19 to 1.27	2215.45 to 2516.95

It can be seen that the CPU Processing %, Heap Memory and the number of hops vary with sample number when no sensors are failed. Since, each time, the algorithm is run on a newly formed network, the number of hops, CPU Processing %, Heap Memory vary, depending on the density of the network. Greater number hops, signifies a densely populated sensor network for a network with the same sample number. The number of hops is this case is likely to decrease with a decrease in the amount of heap memory consumed.

Table 8: Experimental Results for 500 Sensors with 3 Sensor Failures

S.No	Source Sensor	Destination Sensor	No of Hops	CPU Processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	Sensor11	Sensor500	105	0.45	1.232	2074
2	Sensor43	sensor500	112	0.41	1.316	2145
3	sensor59	Sensor500	132	0.44	1.2	2242
4	sensor81	Sensor500	110	0.43	1.289	2563
5	Sensor91	Sensor500	148	0.51	1.345	3091
6	sensor110	Sensor500	122	0.46	1.271	2521
7	Sensor10	Sensor500	129	0.52	1.328	3123
8	Sensor190	Sensor500	137	0.4	1.289	2432
9	sensor210	Sensor500	176	0.53	1.32	2222
10	Sensor225	Sensor500	187	0.35	1.181	2095
11	Sensor250	Sensor500	115	0.4	1.291	2671
12	Sensor255	Sensor500	175	0.45	1.245	2335
13	Sensor300	Sensor500	97	0.43	1.241	2241
14	Sensor231	Sensor500	76	0.42	1.232	2053
15	Sensor56	Sensor500	173	0.45	1.236	2420
Total			1994	6.66	19.0159	36228
Mean			132.3333	0.444	1.2677	2415.2
Median			129	0.44	1.271	2335
Standard Deviation			32.8752	0.0477	0.0485	336.3361
Confidence Interval			16.64	0.02	0.03	170.21
Range			101.29 to 149.57	0.42 to 0.46	1.25 to 1.29	2244.99 to 2585.41

Comparing the results in Table 7 and Table 8, it is seen that the total number of Hops, CPU Processing %, the CPU Process Time and Heap Memory increases with 3 sensor failures. It is concluded from the above results in table 7 and 8 that as the number of failed sensors increases, the number of Hops, CPU Processing %, CPU Process Time and the Heap Memory increases.

Figure 24 shows the graphical representation of the Number of Hops vs. Sample Number. It can be seen that the Number of Hops increases with Sample Number on three sensor failures.

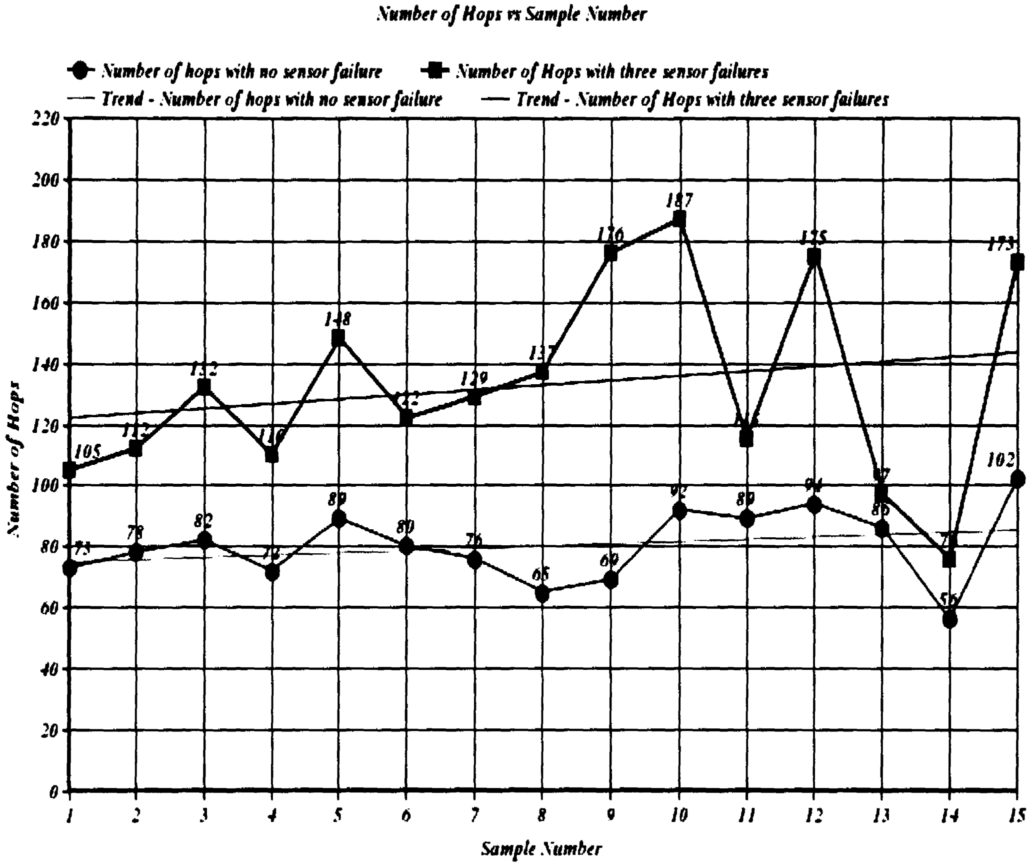


Figure 24: Graph of Number of Hops vs. Sample Number for 500 Sensors with 3 Sensor Failures

It can be seen that the number of Hops increases with sample number on three sensor failures. The algorithm traverses through more number of sensors to find an alternative shortest path with three sensor failures. The number of Hops increases, since finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm to reach the destination sensor

Figure 25 shows the graphical representation of the CPU Processing % vs. Sample Number. It can be seen that the CPU Processing % increases with Sample Number on three sensor failures.

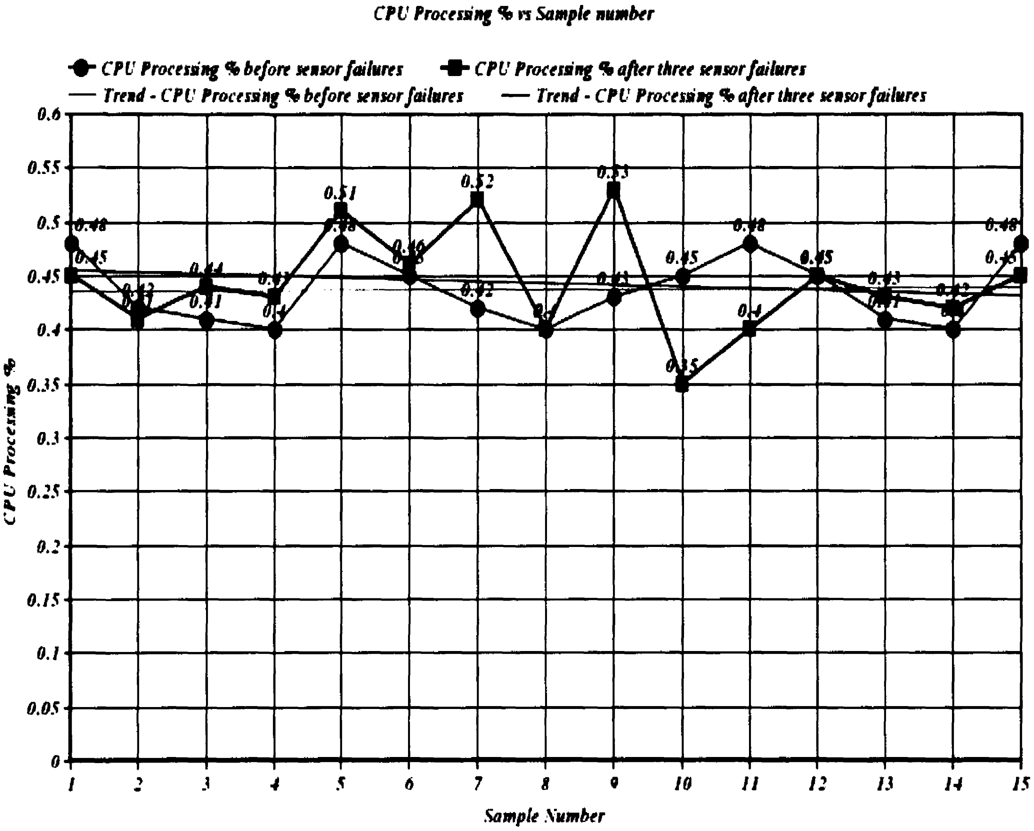


Figure 25: Graph of CPU Processing % vs. Sample Number for 500 Sensors with 3 Sensor Failures

It can be seen that the CPU Processing % increases with sample number on three sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path. Finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found, which results in an increase in CPU Processing % consumption. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm.

Figure 26 shows the graphical representation of the CPU Process Time vs. Sample Number. It can be seen that the CPU Process Time varies with Sample Number on three sensor failures.

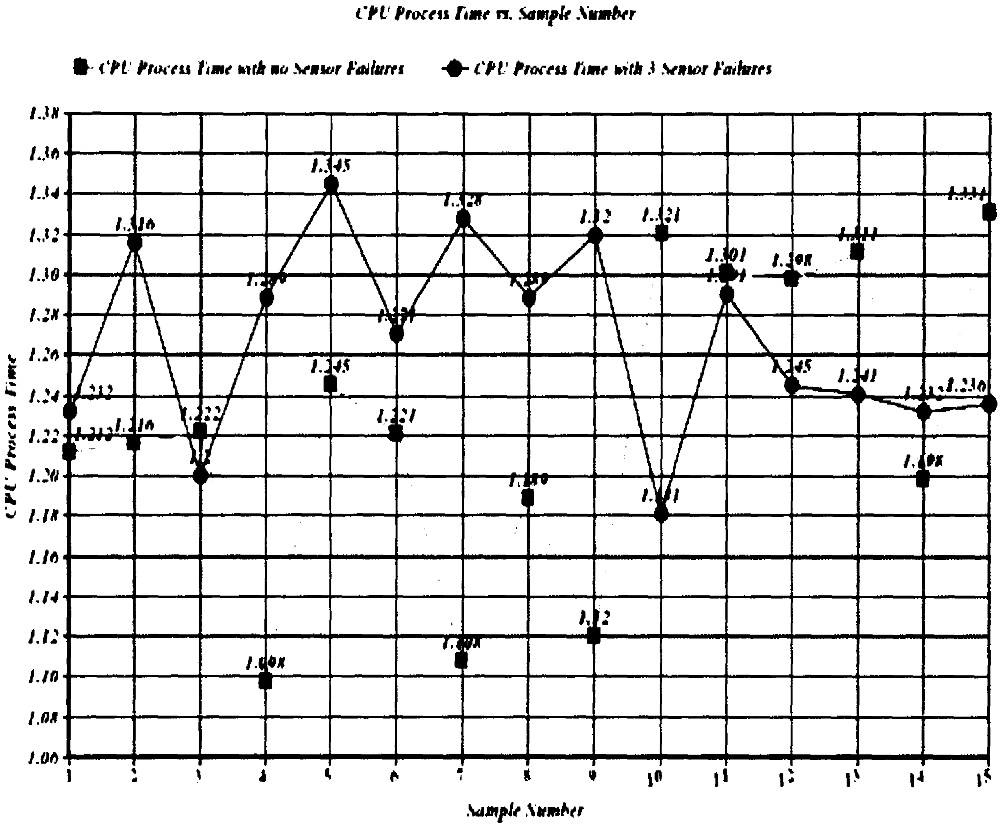


Figure 26: Graph of CPU Process Time vs. Sample Number for 500 Sensors with 3 Sensor Failures

It can be seen that the CPU Process Time increases with sample number on three sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path. Finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found which results in an increase in CPU Process Time consumption.

The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm to reach the destination sensor. Figure 27 shows the graphical representation of the Heap Memory vs. Sample Number. It can be seen that the Heap Memory varies with Sample Number on three sensor failures.

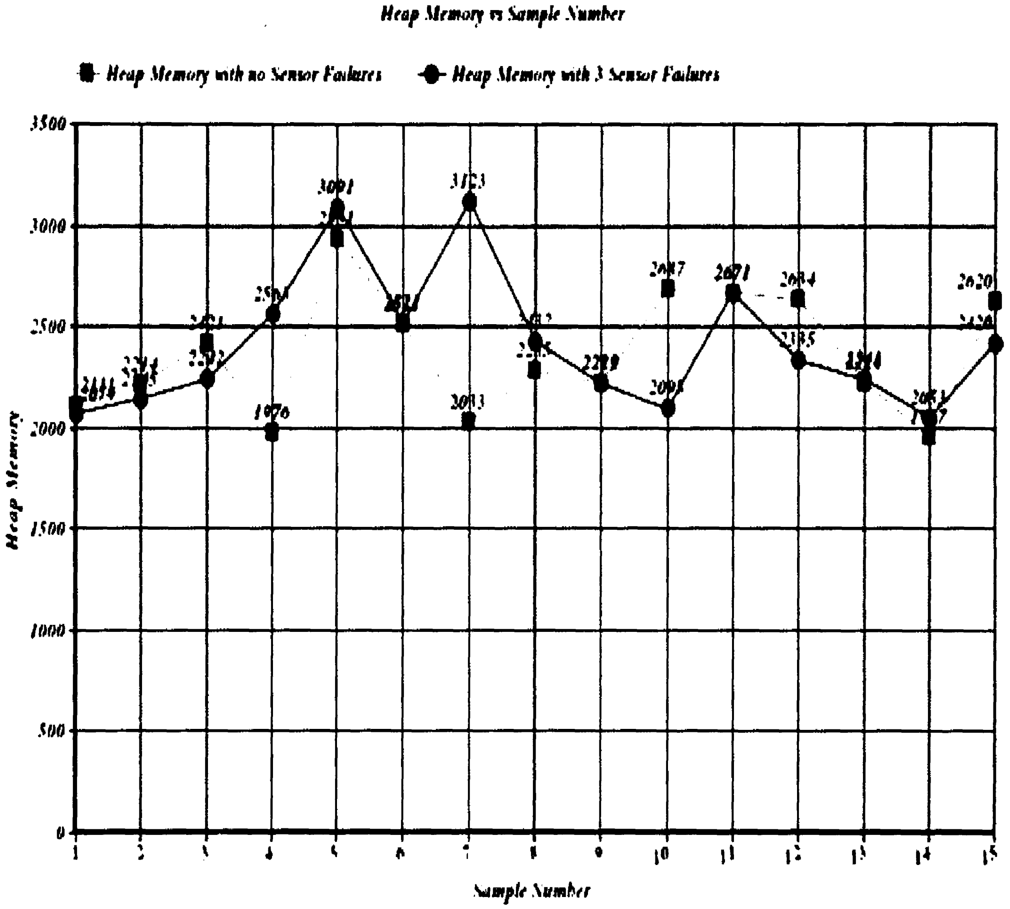


Figure 27: Graph of Heap Memory vs. Sample Number for 500 Sensors with 3 Sensor Failures

It can be seen that the Heap Memory increases with sample number on three sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path. Finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found which results in an increase in Heap Memory consumption. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm to reach the destination sensor. This result in a decrease in the amount of memory consumed.

Table 9: Experimental Results for 5000 Sensors with no Sensor Failure

S.No	Source Sensor	Destination Sensor	No of Hops	CPU Processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	Sensor11	Sensor5000	453	0.51	1.132	2574
2	Sensor43	sensor5000	588	0.54	1.216	2745
3	sensor59	Sensor5000	454	0.52	1.22	2342
4	sensor81	Sensor5000	888	0.2	1.089	2963
5	Sensor91	Sensor5000	963	0.54	1.445	3397
6	sensor110	Sensor5000	1001	0.66	1.471	3529
7	Sensor10	Sensor5000	1211	0.57	1.228	4123
8	Sensor190	Sensor5000	767	0.47	1.229	3212
9	sensor210	Sensor5000	875	0.63	1.46	3822
10	Sensor225	Sensor5000	740	0.55	1.282	3597
11	Sensor250	Sensor5000	873	0.52	1.311	3471
12	Sensor255	Sensor5000	911	0.68	1.343	3635
13	Sensor300	Sensor5000	511	0.37	1.142	2241
14	Sensor231	Sensor5000	234	0.55	1.232	2353
15	Sensor56	Sensor5000	431	0.58	1.286	2820
Total			10900	7.89	19.086	46824
Mean			726.6666	0.526	1.2724	3121.6
Median			767	0.54	1.232	3212
Standard Deviation			29.396	0.1174	0.1174	590.811
Confidence Interval			136.33	0.06	0.06	298.99
Range			590.33 to 862.99	0.47 to 0.59	1.21 to 1.33	2822.61 to 3420.59

It can be seen that the CPU Processing %, Heap Memory and the number of hops vary with sample number when no sensors are failed. Since, each time, the algorithm is run on a newly formed network, the number of hops, CPU Processing %, Heap Memory vary, depending on the density of the network. Greater number hops, signifies a densely populated sensor network for a network with the same sample number. The number of hops is this case is likely to decrease with a decrease in the amount of heap memory consumed.

Table 10: Experimental Results for 5000 Sensors with 5 Sensor Failures

S.No	Source Sensor	Destination Sensor	No of Hops	CPU Processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	Sensor11	Sensor5000	673	0.53	1.072	2672
2	Sensor43	sensor5000	787	0.64	1.411	2835
3	sensor59	Sensor5000	675	0.42	1.31	2322
4	sensor81	Sensor5000	911	0.6	1.189	2813
5	Sensor91	Sensor5000	1024	0.67	1.411	3098
6	sensor110	Sensor5000	1343	0.71	1.49	3331
7	Sensor10	Sensor5000	1872	0.73	1.518	4012
8	Sensor190	Sensor5000	876	0.57	1.2	3552
9	sensor210	Sensor5000	1221	0.53	1.413	3986
10	Sensor225	Sensor5000	1076	0.6	1.222	3322
11	Sensor250	Sensor5000	987	0.58	1.34	3459
12	Sensor255	Sensor5000	1201	0.59	1.45	3658
13	Sensor300	Sensor5000	554	0.49	1.141	2012
14	Sensor231	Sensor5000	472	0.41	1.312	2871
15	Sensor56	Sensor5000	987	0.59	1.196	2561
Total			146590	8.6659	19.675	46504
Mean			977.267	0.5773	1.3116	3100.27
Median			987	0.59	1.312	3098
Standard Deviation			351.032	0.0924	0.1362	587.471
Confidence Interval			177.65	0.05	0.07	297.3
Range			977.61 to 1154.91	0.53 to 0.63	0.24 to 1.38	2802.9 to 3397.5

Comparing the results in Table 9 and Table 10, it is seen that the number of Hops, CPU processing % and CPU process time increases with 5 sensor failures. It is concluded form the above results in table 9 and 10 that as the number of failed sensors increases, the number of Hops, CPU Processing % and CPU Process Time increases. Figure 28 shows the graphical representation of the Number of Hops vs. Sample Number. It can be seen that the Number of Hops increases with Sample Number on five sensor failures. This result in a decrease in the mount of memory consumed.

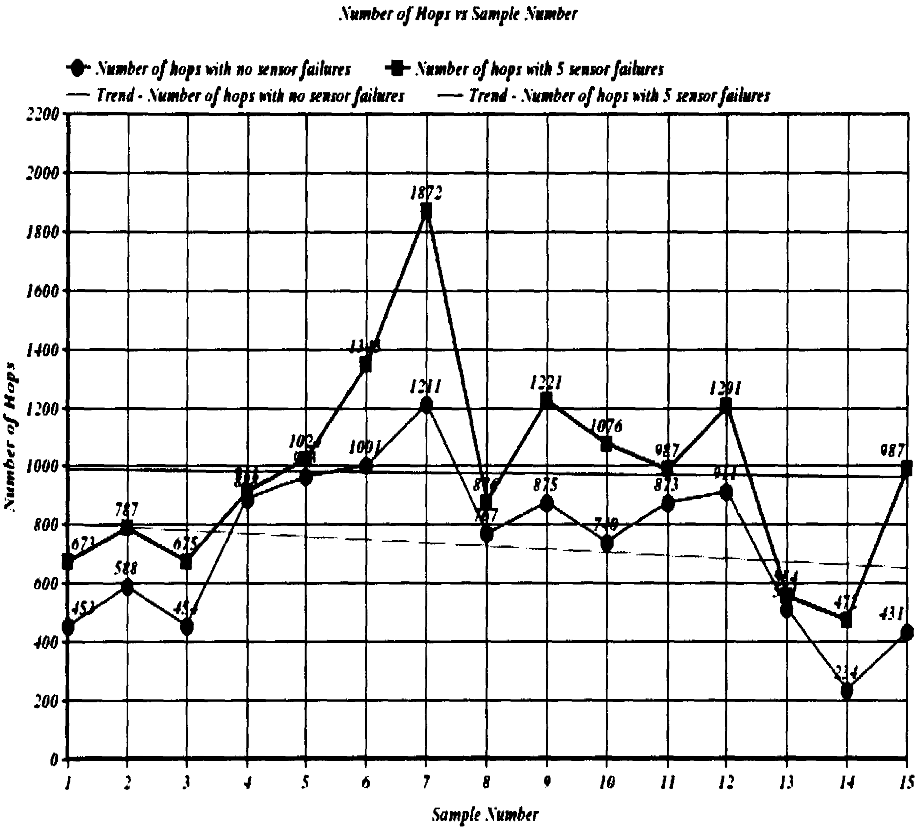


Figure 28: Graph of Number of Hops vs. Sample Number for 5000 Sensors with 5 Sensor Failures

It can be seen that the number of Hops increases with sample number on five sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path. The number of hops increases, since finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found.

Figure 29 shows the graphical representation of the CPU Processing % vs. Sample Number. It can be seen that the CPU Processing % increases with Sample Number on five sensor failures. The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm to reach the destination sensor.

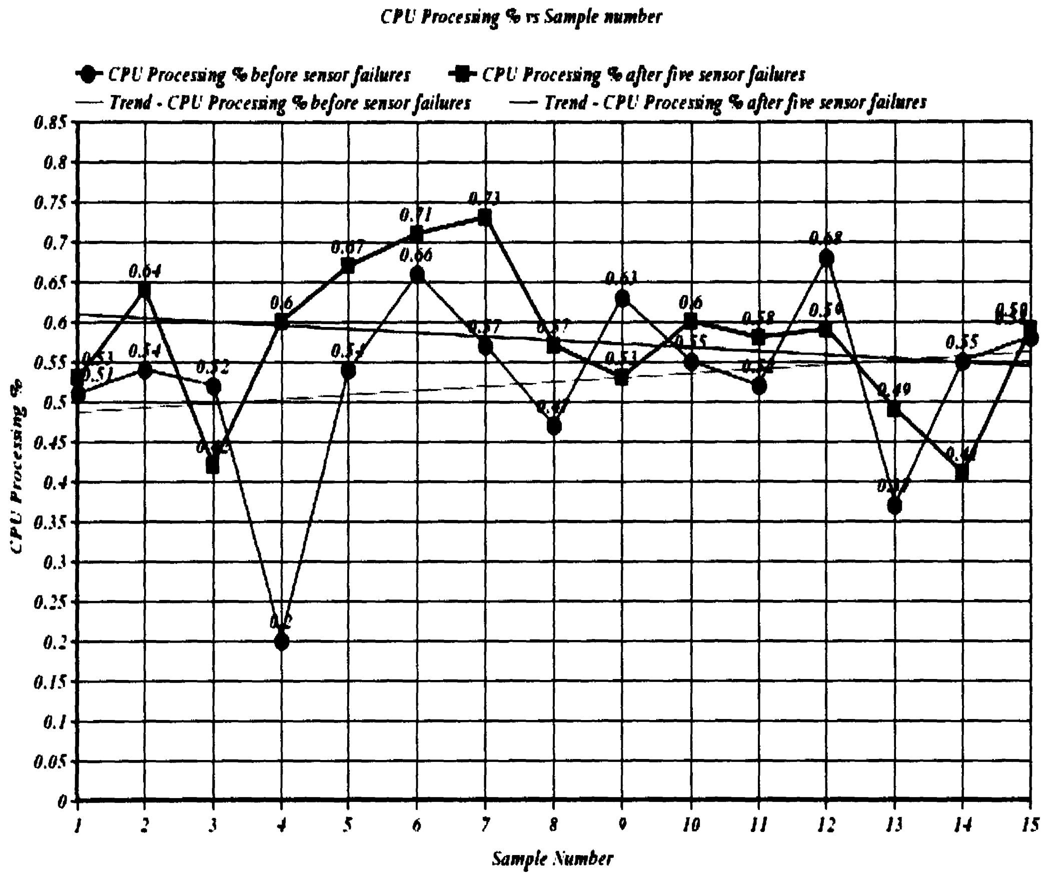


Figure 29: Graph of CPU Processing % vs. Sample Number for 5000 Sensors with 5 Sensor Failures

It can be seen that the CPU Processing % increases with sample number on five sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path. Finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found, which results in an increase in CPU Processing % consumption. Figure 30 shows the graphical representation of the CPU Process Time vs. Sample Number. It can be seen that the CPU Process Time varies with Sample Number on five sensor failures.

The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm to reach the destination sensor.

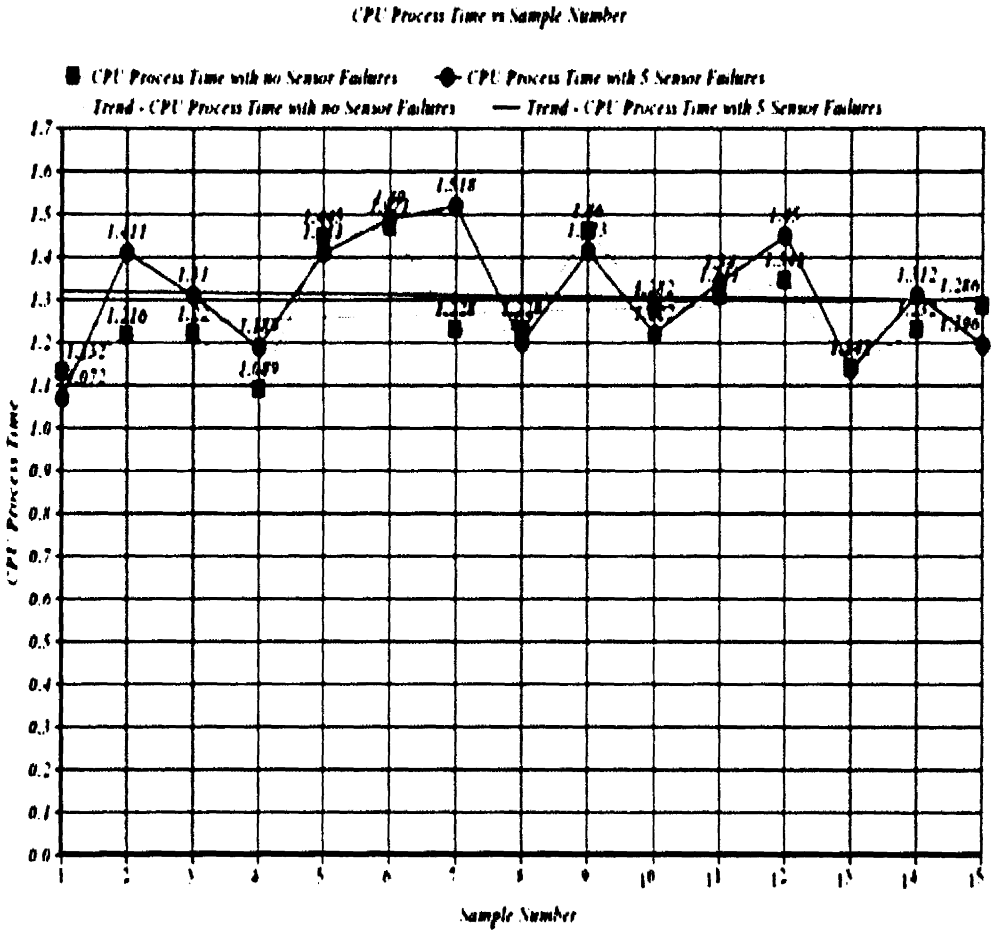


Figure 30: Graph of CPU Process Time vs. Sample Number for 5000 Sensors with 5 Sensor Failures

It can be seen that the CPU Process Time increases with sample number on five sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path. Finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found, which results in an increase in CPU Process Time consumption.

Figure 31 shows the graphical representation of the Heap Memory vs. Sample Number. It can be seen that the Heap memory varies depending on the number of failed sensors. The number of hops is this case is likely to decrease with a decrease in the amount of heap memory consumed.

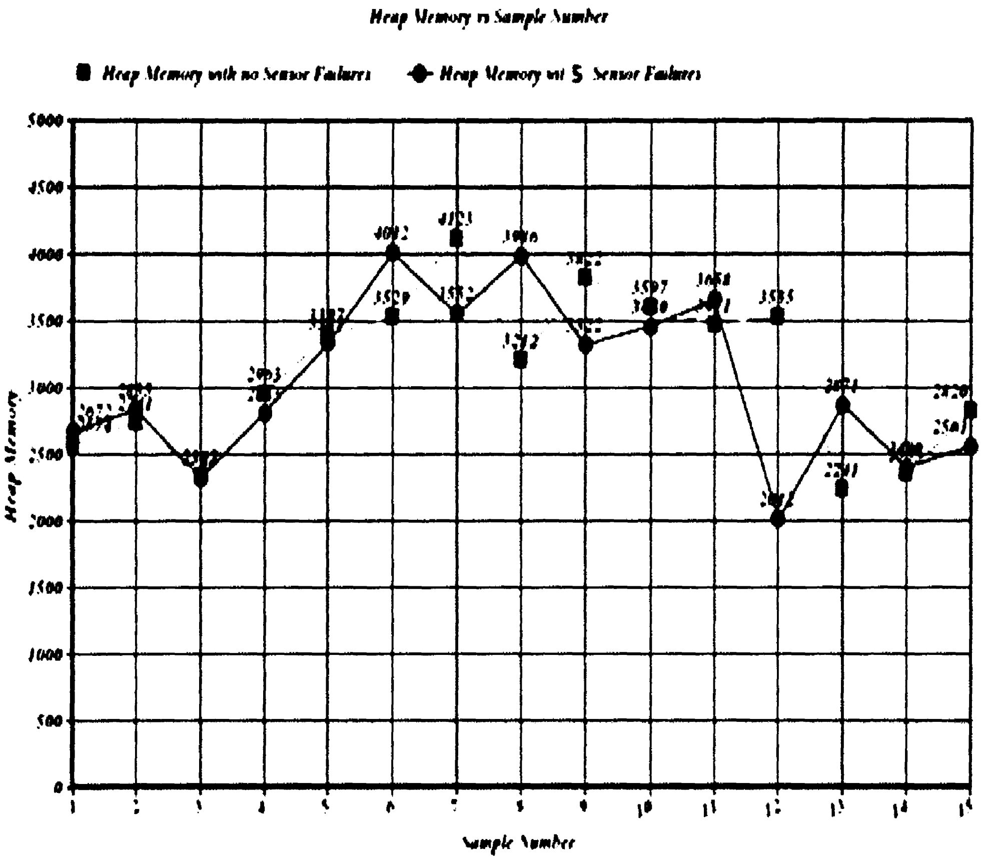


Figure 31: Graph of Heap Memory vs. Sample Number for 5000 Sensors with 5 Sensor Failures

It can be seen that the Heap Memory on average increases with sample number on five sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path. The Heap Memory increases, since finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found, which results in an increase in average Heap Memory consumption.

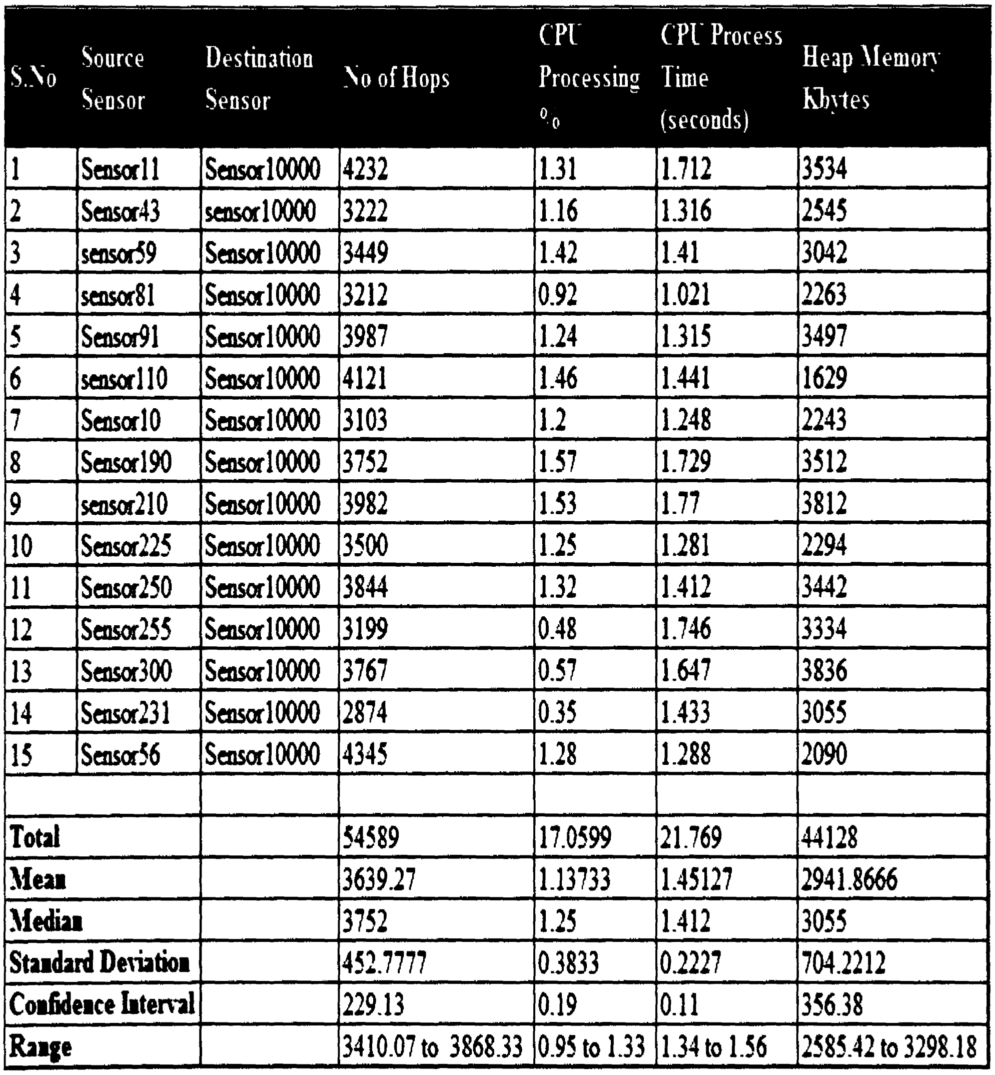
The next shortest path, might consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm to reach the destination sensor. These results in a decrease in the amount of memory consumed.

Table 11: Experimental Results for 10000 Sensors with no Sensor Failure

S.No	Source Sensor	Destination Sensor	No of Hops	CPU Processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	Sensor11	Sensor10000	3430	1.21	1.632	3574
2	Sensor43	Sensor10000	2422	1.14	1.416	2945
3	sensor59	Sensor10000	2532	1.12	1.42	2942
4	sensor81	Sensor10000	1678	0.72	1.089	1963
5	Sensor91	Sensor10000	3411	1.14	1.245	3097
6	sensor110	Sensor10000	3098	1.16	1.371	3629
7	Sensor10	Sensor10000	2987	1.2	1.428	3123
8	Sensor190	Sensor10000	2975	1.17	1.229	2912
9	sensor210	Sensor10000	2453	1.03	1.06	2822
10	Sensor225	Sensor10000	3109	1.55	1.682	3797
11	Sensor250	Sensor10000	3587	1.12	1.311	3271
12	Sensor255	Sensor10000	2311	0.68	1.243	2935
13	Sensor300	Sensor10000	2867	0.87	1.242	2832
14	Sensor231	Sensor10000	1765	0.65	1.032	1355
15	Sensor56	Sensor10000	3718	1.58	1.686	3890
Total			42343	16.3399	20.086	45087
Mean			2822.87	1.0893	1.33907	3005.8
Median			2975	1.14	1.311	2945
Standard Deviation			620.091	0.2734	0.2104	660.6036
Confidence Interval			313.81	0.14	0.11	334.31
Range			2508.99 to 3136.61	0.95 to 1.23	1.23 to 1.45	2671.49 to 3340.11

It can be seen that the CPU Processing %, Heap Memory and the number of hops vary with sample number when no sensors are failed. Since, each time, the algorithm is run on a newly formed network, the number of hops, CPU Processing %, Heap Memory vary, depending on the density of the network. Greater number hops, signifies a densely populated sensor network for a network with the same sample number.

Table 12: Experimental Results for 10000 Sensors with 10 Sensor Failures



S.No	Source Sensor	Destination Sensor	No of Hops	CPU Processing %	CPU Process Time (seconds)	Heap Memory Kbytes
1	Sensor11	Sensor10000	4232	1.31	1.712	3534
2	Sensor43	sensor10000	3222	1.16	1.316	2545
3	sensor59	Sensor10000	3449	1.42	1.41	3042
4	sensor81	Sensor10000	3212	0.92	1.021	2263
5	Sensor91	Sensor10000	3987	1.24	1.315	3497
6	sensor110	Sensor10000	4121	1.46	1.441	1629
7	Sensor10	Sensor10000	3103	1.2	1.248	2243
8	Sensor190	Sensor10000	3752	1.57	1.729	3512
9	sensor210	Sensor10000	3982	1.53	1.77	3812
10	Sensor225	Sensor10000	3500	1.25	1.281	2294
11	Sensor250	Sensor10000	3844	1.32	1.412	3442
12	Sensor255	Sensor10000	3199	0.48	1.746	3334
13	Sensor300	Sensor10000	3767	0.57	1.647	3836
14	Sensor231	Sensor10000	2874	0.35	1.433	3055
15	Sensor56	Sensor10000	4345	1.28	1.288	2090
Total			54589	17.0599	21.769	44128
Mean			3639.27	1.13733	1.45127	2941.8666
Median			3752	1.25	1.412	3055
Standard Deviation			452.7777	0.3833	0.2227	704.2212
Confidence Interval			229.13	0.19	0.11	356.38
Range			3410.07 to 3868.33	0.95 to 1.33	1.34 to 1.56	2585.42 to 3298.18

Comparing the results in Table 11 and Table 12, it is seen that the number of Hops, CPU Processing % and CPU Process Time increases with 10 sensor failures. It is concluded from the above results in Table 11 and Table 12 that as the number of failed sensors increases, the number of Hops, CPU processing % and CPU Process Time increases.

Figure 32 shows the graphical representation of the Number of Hops vs. Sample Number. The number of hops is this case is likely to decrease with a decrease in the amount of heap memory consumed. It can be seen that the Number of Hops increases with Sample Number on ten sensor failures.

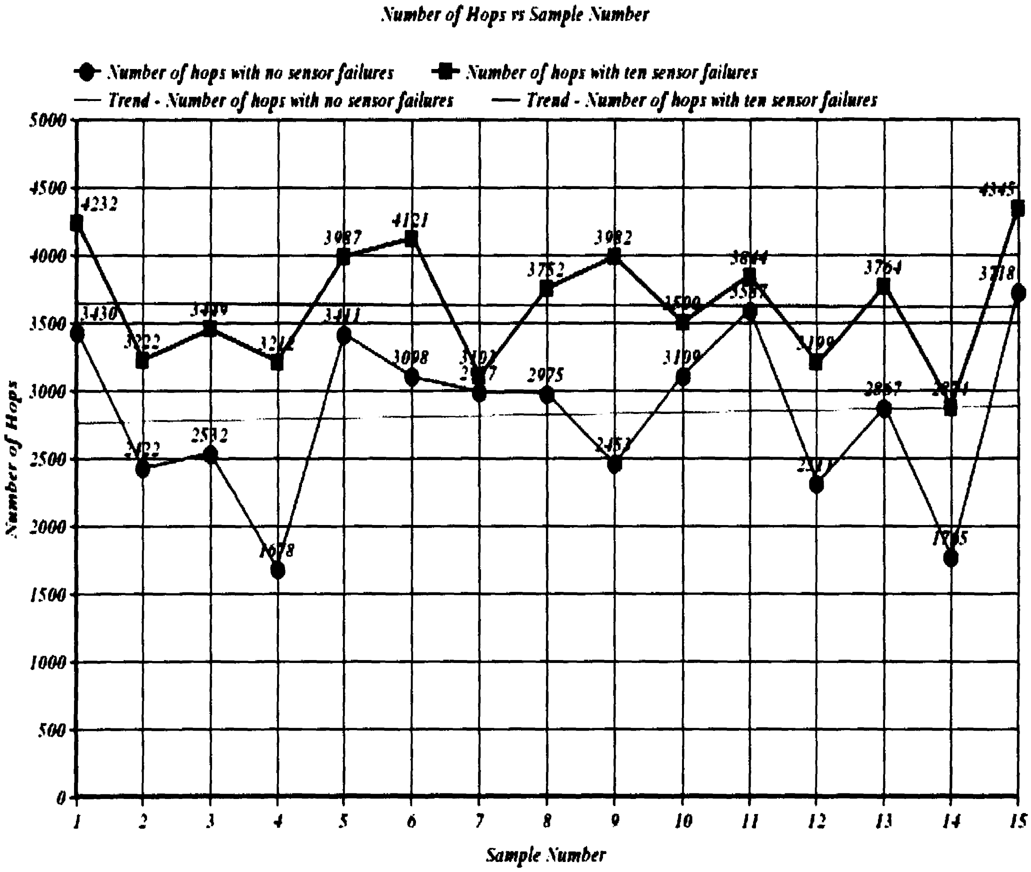


Figure 32: Graph of Number of Hops vs. Sample Number for 10000 Sensors with 10 Sensor Failures

It can be seen that the number of Hops increases with sample number on ten sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path on ten sensor failures. The number of hops increases, since finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found. The next shortest path, may consume less time depending on the number of sensors the RREQ has to pass through during subsequent iteration of the algorithm. The number of sensors that the request had to pass through defines the hop count.

Figure 33 shows the graphical representation of the CPU Processing % vs. Sample Number. It can be seen that the CPU Processing % increases with Sample Number on ten sensor failures.

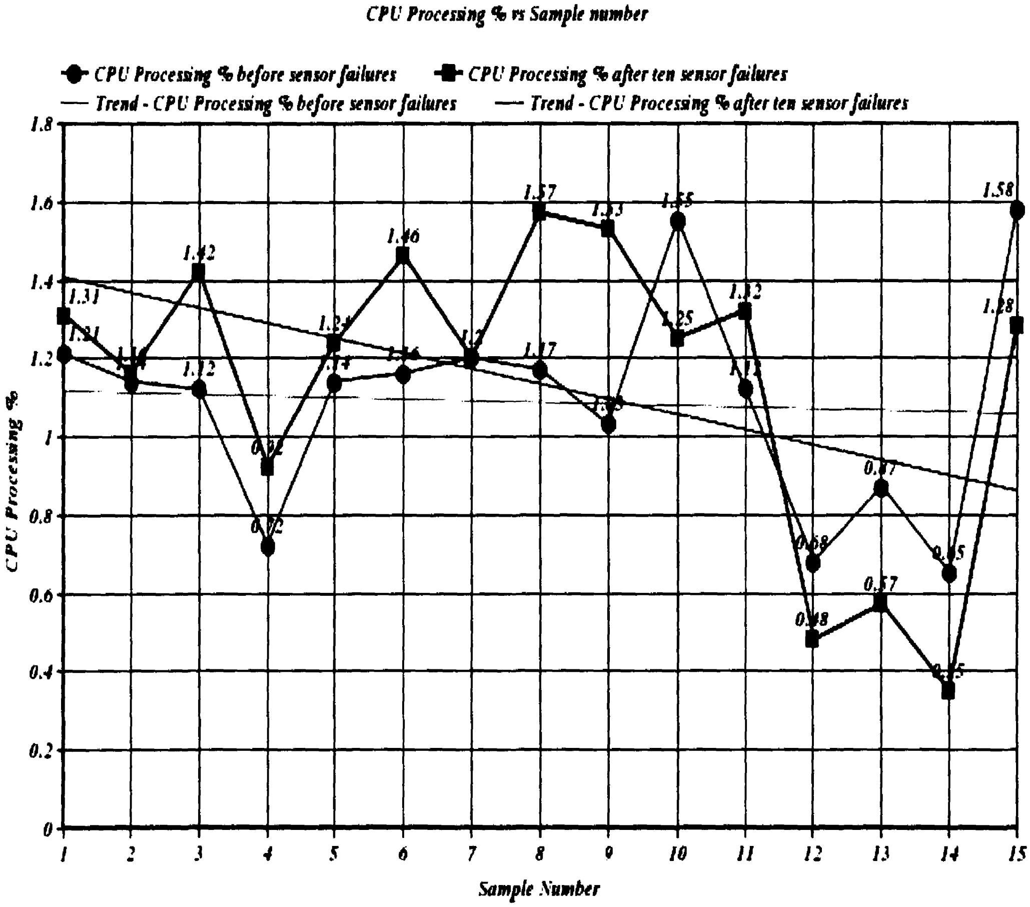


Figure 33: Graph of CPU Processing % vs. Sample Number for 10000 Sensors with 10 Sensor Failures

It can be seen that the CPU Processing % increases with sample number on ten sensor failures. The algorithm has to traverse through more number of sensors to find an alternative shortest path on ten sensor failures. Finding an alternative shortest path involves broadcasting RREQ to neighboring sensors until a new shortest path is found which results in an increase in the CPU Processing % consumed.

9. CONCLUSION AND FUTURE WORK

From the research and experiments performed, the following can be concluded.

1. Developed a new web application to explore the algorithm to find the shortest path on multiple sensor failures in a wireless sensor network.
2. Established that a shortest path can be found in a hop by hop fashion.
3. Developed a network can be developed based on the transmission distance of homogenous network.
4. Compared the CPU Processing Percentage and CPU Process Time and the Heap Memory used to identify the shortest path before and after multiple sensor failures.
5. The total CPU Processing Percentage and CPU Process Time and the Heap Memory increases with sensor failures.
6. The hop count in the shortest path increases as the number of failed sensors increases.
7. Results were consistent for 100, 500, 1000, 5000 and 10000 sensors.

The web application can be extended to form multiple clusters and obtain a shortest path. The code of the application is flexible to support multiple clusters with varying transmission distance with heterogeneous sensors, with minor changes.

Other techniques could be used to compare the efficiency of the research work by comparing the experimental results obtained with different methods such as Voronoi approach [12] based on genetic algorithms [11]. Dynamic sensor paths can be implemented using Google Charting.

10. REFERENCES

[1] "Remote Monitoring in Agricultural Greenhouse Using Wireless Sensor and Short Message Service (SMS)" by *Ian D. Chakeres and Elizabeth M. Belding-Royer*

[2] "Wireless Sensor Networks". http://webhosting.devshed.com/c/a/Web-Hosting-Articles/Wireless-Sensor-Networks-pt-1-Introduction/1/

[3] Adhoc On-Demand Distance Vector Routing http://en.wikipedia.org/wiki/Ad_hoc_ On-Demand_Distance_Vector_Routing

[4] "AODV Routing Protocol Implementation Design" by *Ian D. Chakeres and Elizabeth M. Belding-Royer*

[5] "Ad Hoc On Demand Distance Vector (AODV) Routing". IETF RFC 3561 by *Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir Das.*

[6], "Ad hoc on-demand distance vector (aodv) routing", by *C. Perkins, E. Belding-Royer, and S. Das.*

[7] "Ad Hoc On Demand Distance Vector (AODV) Routing. Draft-ietf-manet- aodv-02.txt, Nov. 1998" by *C. E. Perkins and E. M. Roye*r.

[8] "On-Demand MultiPath Distance Vector Routing in Ad Hoc Networks". Department of Electrical & Computer Engineering and Computer Science, University of Cincinnati by *Mahesh K Marina and Samir R Das,*

[9] "Performance of Multi Path Routing for On-demand Protocols in Mobile Ad Hoc Networks. ACM/Kluwer Mobile Networks and Applications" by *A Nasipuri, R Castaneda, and S R Das.*

[10] "Routing in ZigBee: benefits from exploiting the IEEE 802.15.4 association tree, 2003" *by Francesca Cuomo, Sara Della Luna, Ugo Monaco, and Tommaso Melodia.*

[11] "Genetic Algorithm Finding the Shortest Path in Networks". Department of Computer Science and Engineering, University of Nevada, *by Bilal Gonen.*

[12] "Voronoi Diagrams 6.838 Computational Geometry", by *Allen Miu.*

[13] "ACRR: Ad-hoc On-Demand Distance Vector Routing with Controlled Route Requests". Mumbai University, India by Jayesh *Kataria, P.S. Dhekne and Sugata Sanyal.*

[14] "Using JConsole for monitoring a Java process using Oracle JDK 1.5". http://downloadllnw.oracle.com/javase/1.5.0/docs/guide/management/jconsole.html

[15] "Confidence Interval". http://en.wikipedia.org/wiki/Confidence_interval

[16] "Confidence Intervals for Probabilities of Default" by Samuel Hanson and Til Schuermann.

[17] "Likelihood Ratio Based Confidence Intervals in Survival Analysis" by S.A. Murphy.

[18] "Charting Tool". http://www.onlinecharttool.com/graph.php

[19] "Google Graphs". http://code.google.com/apis/chart/interactive/docs/quick_start.html