PATH PLANNING UNDER FAILURES IN WIRELESS SENSOR NETWORKS

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

By

Nityananda Suresh Paturu Raghunatha Rao

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
COMPUTER SCIENCE

March 2012

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Path Planning Under Failures in Wireless Sensor Networks

By

Nityananda Suresh Paturu Raghunatha Rao

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

SUPERVISORY COMMITTEE:

Dr. Kendall E. Nygard
Chair

Dr. Simone Ludwig

Dr. Tariq King

Dr. Joseph Szmerekovsky

Approved:

1/29/13 — Date

Dr. Brian Slator — Department Chair

ABSTRACT

This paper explores how an all pair shortest path can be obtained in a wireless sensor network when sensors fail. Sensors are randomly deployed in a predefined geographical area, simulating the deployment of sensors from an airplane, and finding shortest path between all the sensors deployed based on distance. A major problem to address in wireless sensor networks is the impact of sensor failures on existing shortest paths in the network. An application is developed to simulate a network and find shortest paths affected by a sensor failure and find alternative shortest path. When a sensor fails, all the shortest paths and all the remaining sensors in the network are checked to see if the sensor failure has any impact on the network. Alternative shortest path is calculated for those paths affected by sensor failures.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my advisor, Dr. Kendall Nygard, who has given me immense support, motivation, and valuable advice that helped me to grow as a researcher and complete my paper. I am thankful to my committee members, Dr. Simone Ludwig, Dr. Tariq King, and Dr. Joseph Szmerekovsky for their consistent support. I am also thankful to my relatives and friends for their support throughout my lifetime.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

1. INTRODUCTION

Wireless sensor networks are of interest because of their use in the real world. They are an active research area in telecommunications, military applications, industrial applications, buildings, system automation, and utilities [1].

1.1. Wireless Sensor Network

The building blocks of a wireless sensor network are sensors. This network consists of distributed sensors that transmit their data to a base station [2]. Each sensor contains a microchip with limited processing power and limited memory, an energy source (usually a battery), a transmitter, and a receiver [1]. A sensor has characteristics such as a communication radius and a sensing radius. Each sensor is connected to one or more sensors based on the distance between them. Two sensors are called neighbors if the distance between them is less than or equal to the communication radius. A sensor fails if any of its parts don’t work. The central location is called the base station, and it has more computational power and communication resources.

1.1.1. Advantages and drawbacks of wireless sensor networks

Wireless sensor networks have many advantages. They are easy to use and provide mobility with less power consumption. They can operate under harsh environmental conditions, cover a wide geographical area, and gather information without monitoring on a daily basis. The cost of the network differs based on the size of the sensors, energy, memory, and computational ability.

1.1.2. Applications of wireless sensor networks

Wireless sensor networks are used in military applications and industrial applications for monitoring and control. They are also used in environmental applications for monitoring weather, traffic, waste water levels, area monitoring, and structural monitoring. Below are some other uses of wireless sensors networks:

a. Wireless sensor networks are used in multimedia surveillance to help systems find missing persons and terrorists, to protect against terror attacks, and to help law enforcement agencies monitor areas [3].
b. Multimedia sensors are used to record crimes, monitor traffic, and retrieve information, such as average speed and number of cars [3].
c. Sensors are used in health care to record a patient's body temperature, blood pressure, and pulse at particular time intervals [3].
d. Sensors are used in the industrial sector for quality control and automation [3].
e. Sensors are used in the military to survey the battlefield, target opposing forces, assess battle damage, detect nuclear or chemical attacks, and monitor friendly forces and ammunition [4].

2. LITERATURE REVIEW

The AODV Routing Protocol uses the destination sequence number with an on-demand approach. A route is discovered on demand when a sensor is ready to transmit information. The sensor uses hop-by-hop routing. When a source node is ready to transmit information, it floods a route request in the network for the destination. If the intermediate node receives a duplicate route request, it is discarded. If a valid route is available to the destination, then a route reply is generated by the intermediate node. Otherwise, the route request is sent to the node from which the route request is received. If the intermediate node itself is the destination, then it generates a route reply. While passing the route request to neighboring sensors, the sequence number, hop count, and next hop are updated [3].

When a source node needs to transmit information to a destination for which the path is not known, it needs to discover the path. All the nodes in the network have a node sequence number and a broadcast id. A route request packet is sent to the neighboring sensors by the source node, which consists of the source address and sequence number, broadcast id, and destination address and sequence number. The broadcast id increases with each broadcast of a new route request. If two route requests have the same source sequence and broadcast id, they are discarded from the intermediate nodes. If a valid route is available, the intermediate node checks if it has a route entry for the destination. The destination sequence number in the route request is compared with its own entry. If the destination sequence number is greater in route request, the intermediate node has to rebroadcast the route request. Otherwise the intermediate node generates a route reply back to the neighbor from

which it received the route request. The intermediate node updates its path information with the highest sequence number [4].

Another algorithm that is used to find the shortest path between a source and destination is Dijkstra's algorithm, which computes the shortest paths by visiting all the nodes in a graph in increasing distance from a given source. It maintains a priority queue, which will have the path from source to destination. In each iteration step, a node with the least key value is entered into the queue. The key value of each node is calculated by adding the weight of the edge between the nodes in the queue and the remaining nodes. The least key value node is added to the queue. This process is repeated until the destination node is reached [7]. This algorithm gives the shortest distance, which is not the optimal shortest distance in the network, from the source to the destination. It takes more processing time to compute the shortest distance, as it goes through all the nodes in the network until it finds the destination.

A faster approach to find the shortest path is to add two more data structures to Dijkstra's algorithm [8]. A set of permanently labeled nodes and a set of temporarily labeled nodes are the two sets of data stored in the data structures. The minimum labeled distance for a node is identified from the temporarily labeled nodes set for each iteration in Dijkstra's algorithm. This takes O (n^2) steps. Fibonacci heap implementation [9] and Atomic Heap Implementation [10] are used to implement the priority queue in the algorithm to reduce run time [8].

A bidirectional search occurs, in which two Dijkstra's algorithms start in parallel, one running from the source and one running from the destination. The

algorithms stop when they meet each other [11]. To speed up the process of finding the shortest path between a source node and a destination node, the graphs are partitioned to pre-compute if an edge is part of the shortest path. Any partitioning technique can be used, but the number of partitions has to be decided, as Dijkstra's algorithm would give the shortest path. Rectangular partitioning, quad- tree partitioning, kd-tree partitioning, and Metis partitioning techniques are used to partition the graph [11].

3. PROBLEM STATEMENT

Given a geographical area with N x N dimensions, a wireless sensor network with a base station that can communicate with all the sensors in the network, and the shortest path calculated using Floyd's algorithm and a failed sensor, the shortest path affected by the failed sensor is found and an alternative shortest path is calculated. This path is the optimal shortest path after the sensor failure in the wireless sensor network. An overview of the algorithm is defined below:

1. A fixed number of homogenous sensors are deployed randomly in a fixed geographical area with N x N dimensions.
2. The base station is placed in a fixed location. The coordinates are calculated using the communication radius of the sensors.
3. All the sensors in the wireless sensor network are considered source and destination sensors.
4. The shortest path is calculated between all the sensors in the network using Floyd's algorithm.
5. A sensor is randomly selected in the network and is failed in the network.
6. Shortest paths that have the failed sensor in the path are found.
7. Alternative shortest paths are calculated for those paths affected by the failed sensor.

3.1. Creation of the Test Case Problem

Initially, the base station is placed at coordinates (X, Y) in the geographical area, which is at a distance D away from (0, 0), where D is equal to the communication radius. The sensors are deployed randomly in the geographical area. All the sensors, including the base station, are

connected to one another based on the distance between them. The sensors are connected to each other if the distance between the sensors is less than or equal to the communication radius. Shortest paths are found between all the sensors in the network. A sensor is picked randomly to fail and the alternate shortest path is found if the sensor is in any shortest path in the network. There are situations where a sensor failure might affect the connectivity of other sensors to the base station, as it might be the only link between them. The total shortest distance, total hop count, before and after failure of a sensor, and total number of sensors connected to the base station after the failure is obtained for analysis. Below is an example of a 5-sensor network generated and implemented using Floyd's shortest path algorithm by Robert W. Floyd [2]. Figure 3.1 below has a red triangle, which is the base station, and a red circle around it, which is the area inside in which the base station can communicate with the sensors. The distance matrix and link matrix of the sensor network is given below. The distance matrix is denoted by D and the link matrix is denoted by L. The distance matrix denotes the distance between the sensors and the link matrix denotes the direct links between the sensors. Initially, D [I, J] is the distance between sensor "I" and sensor "J" [2]. If there is no direct link between sensor "I" and sensor "J," then D [I, J} is ∞[2]. Similarly, L [I, J] is J if there is a direct link between sensor "I" and sensor "J." If there is no direct link between them, L [I, J] is -1. We start the process by considering all the sensors as intermediate sensors. They might be the link between two sensors that don't have a direct link.

$$D = \begin{array}{c} \\ B \\ S1 \\ S2 \\ S3 \\ S4 \end{array} \begin{array}{c} \begin{array}{ccccc} B & S1 & S2 & S3 & S4 \end{array} \\ \begin{bmatrix} 0.0 & 271.28 & \infty & \infty & 244.38 \\ 271.28 & 0.0 & \infty & 235.85 & 216.01 \\ \infty & \infty & 0.0 & 283.92 & \infty \\ \infty & 235.85 & 283.92 & 0.0 & \infty \\ 244.38 & 216.01 & \infty & \infty & 0.0 \end{bmatrix} \end{array} \quad L = \begin{bmatrix} B & 1 & -1 & -1 & 4 \\ B & 1 & -1 & 3 & 4 \\ -1 & -1 & 2 & 3 & -1 \\ -1 & 1 & 2 & 3 & -1 \\ B & 1 & -1 & -1 & 4 \end{bmatrix}$$

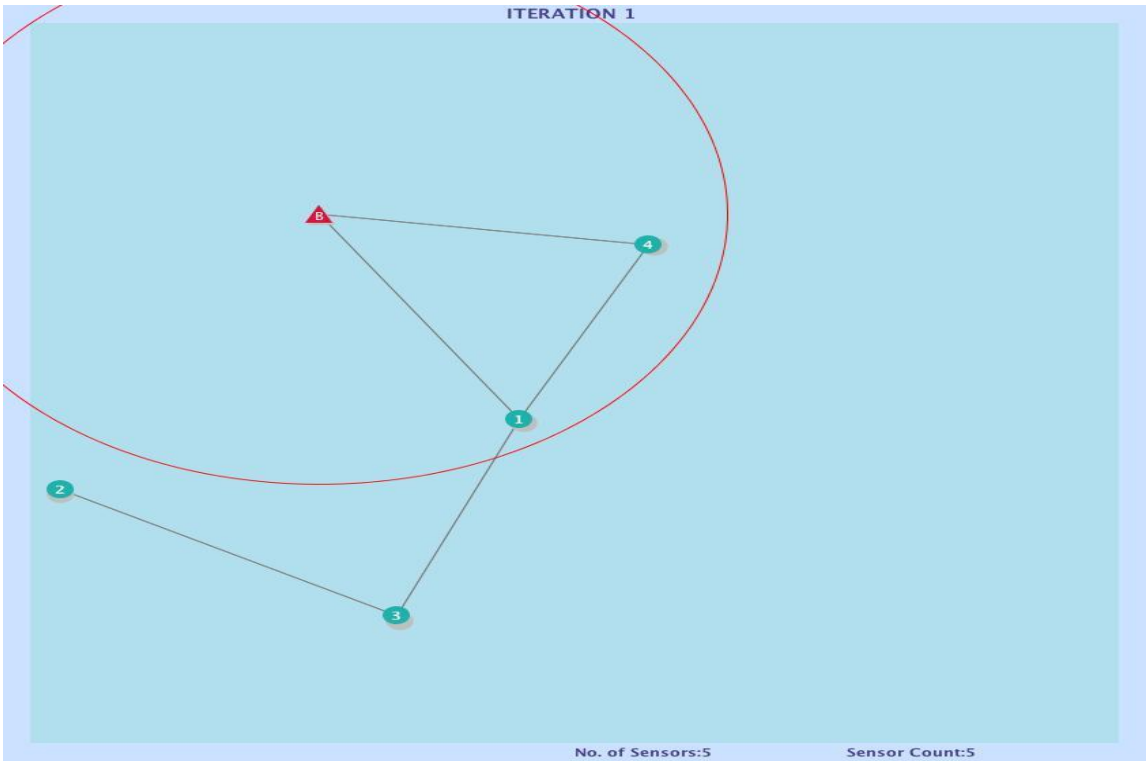

Figure 3.1. Random 5-Sensor Network.

First we use sensor “B” as the intermediate sensor to see if it can create a shortest path between sensors that don’t have a link. Similarly, we take sensor “1” as the intermediate sensor, and continue until we check all the sensors in the network. If any of these sensors create a path between two other sensors, we check if the distance of the path between the two sensors created by the intermediate sensor is less than the existing path’s distance. If it is, then we update the distance matrix with the least distance and the link matrix with the name of the intermediate sensor. The resulting matrix when we run Floyd’s algorithm [2] is:

$$D = \begin{array}{c} \\ B \\ S1 \\ S2 \\ S3 \\ S4 \end{array} \begin{bmatrix} B & S1 & S2 & S3 & S4 \\ 0.0 & 271.28 & 791.05 & 507.13 & 244.38 \\ 271.28 & 0.0 & 519.77 & 235.85 & 216.01 \\ 791.05 & 519.77 & 0.0 & 283.92 & 735.78 \\ 507.13 & 235.85 & 283.92 & 0.0 & 451.86 \\ 244.38 & 216.01 & 735.78 & 451.86 & 0.0 \end{bmatrix} \quad L = \begin{bmatrix} B & 1 & 3 & 1 & 4 \\ B & 1 & 3 & 3 & 4 \\ 3 & 3 & 2 & 3 & 3 \\ 1 & 1 & 2 & 3 & 1 \\ B & 1 & 3 & 1 & 4 \end{bmatrix}$$

Sensors are prone to fail. When a failure occurs then the shortest path might be affected. We need to find all the paths that are affected because of the failure and find the alternative shortest path. If sensor “1” fails, there will be an impact on the network. In Figure 3.2, the sensors shaded in black are not reached by the base station, so the base station cannot communicate with those two sensors. The failure of sensor “1” affected the communication between sensors “2” and “3” with the base station and sensor “4.” The algorithm described in the paper solves the following:

1. Finds the sensors in the network not reached by the base station after each sensor failure.
2. Finds the shortest paths that are affected by the sensor failure.
3. Finds alternative shortest paths if any shortest path between two sensors is affected by the node failure.

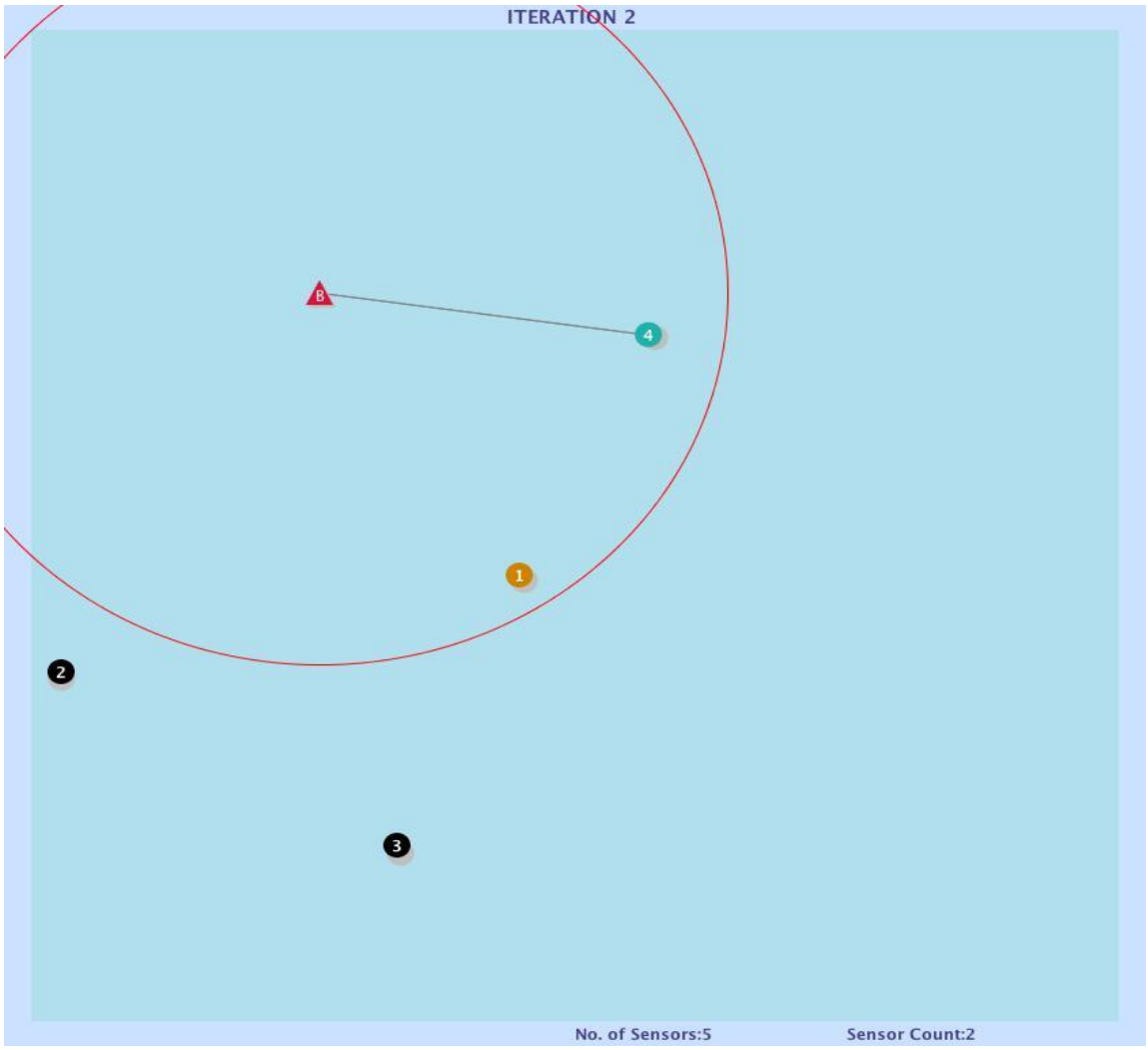

Figure 3.2. Random 5-Sensor Network after one Sensor Failure.

4. IMPLEMENTATION

This section explains how the wireless sensor network is formed. It also explains how the data is transmitted between the sensors and all of the characteristics of the wireless sensor network that are considered.

4.1. Assumptions Taken into Consideration

This algorithm is developed with homogenous sensors in mind, so all the sensors in the wireless sensor network have the same properties and cannot be differentiated by any means. The communication radius of all sensors is considered equal, including the base station. All the sensors in the wireless sensor network are considered bi-directional, which means that all sensors can send and receive data packets. The base station is placed manually in the network. We consider the sensors to have three different states. A sensor is said to be "ALIVE" if the base station can transmit data to it, "FAILED" if it is a failed sensor, or "OUT_OF_BOUNDS" if it is working but the base station cannot transmit data to it.

4.2. Random Graph Generation Algorithm

This subsection explains the creation of the wireless sensor network that forms the basis for the Modified Shortest Path algorithm. A wireless sensor network has to be formed in order to implement any shortest path algorithms. When the sensors are deployed randomly in the geographical area with N x N dimensions, they can interact with other sensors, which are at a distance no farther than the communication radius. Figure 4.1 below represents the deployment of sensors in a network. The sensors are named with integer values, with the maximum number equal to the total number of sensors, e.g., 1, 2, 3, and 4… in the order in which they are deployed. All the sensors are in the "ALIVE" state at the time of deployment. After the sensors are deployed randomly in the geographical area, they have to be interconnected to form a

network. The algorithm below is developed to form the wireless sensor network. Figure 4.2 represents the formation of the network with 10 sensors.

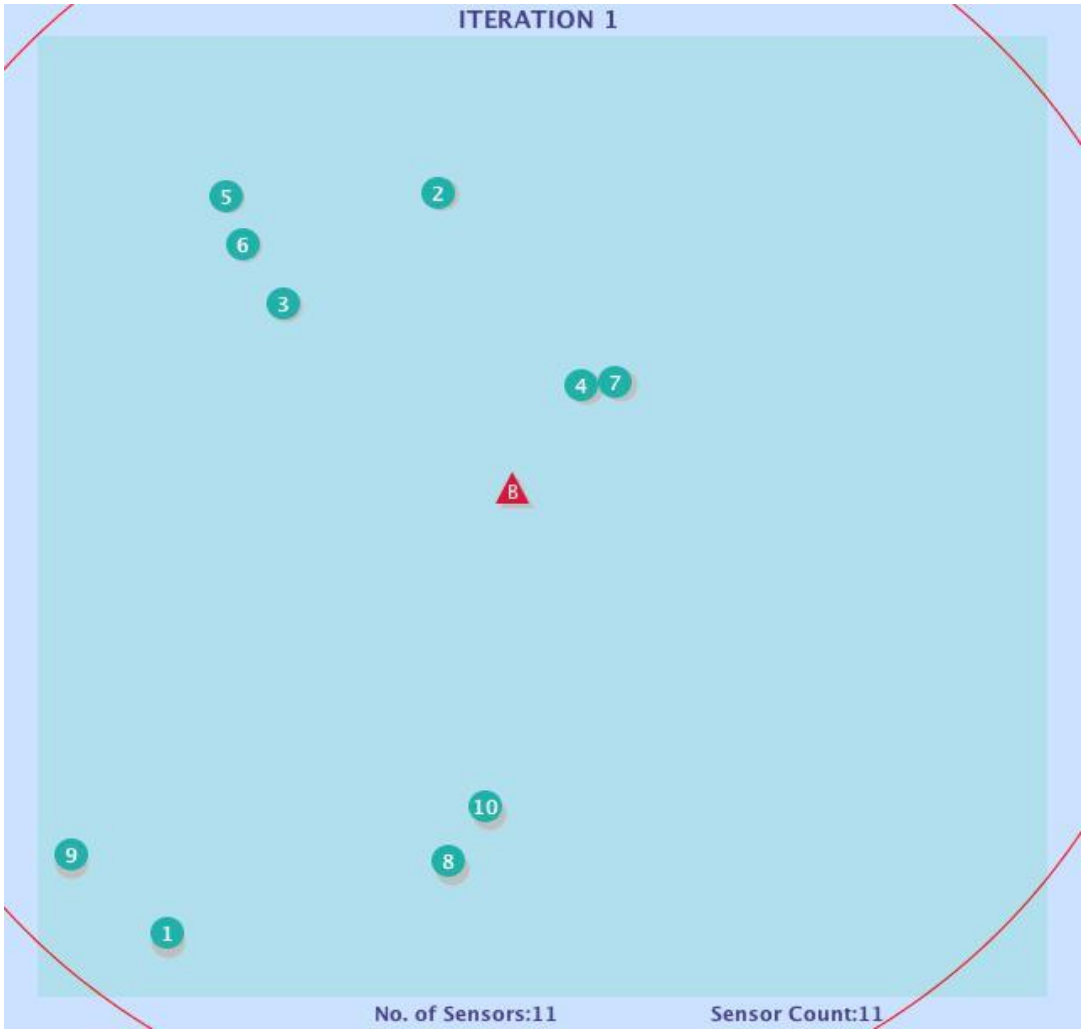

Figure 4.1. Random Deployment of Sensors in the Geographical Area.

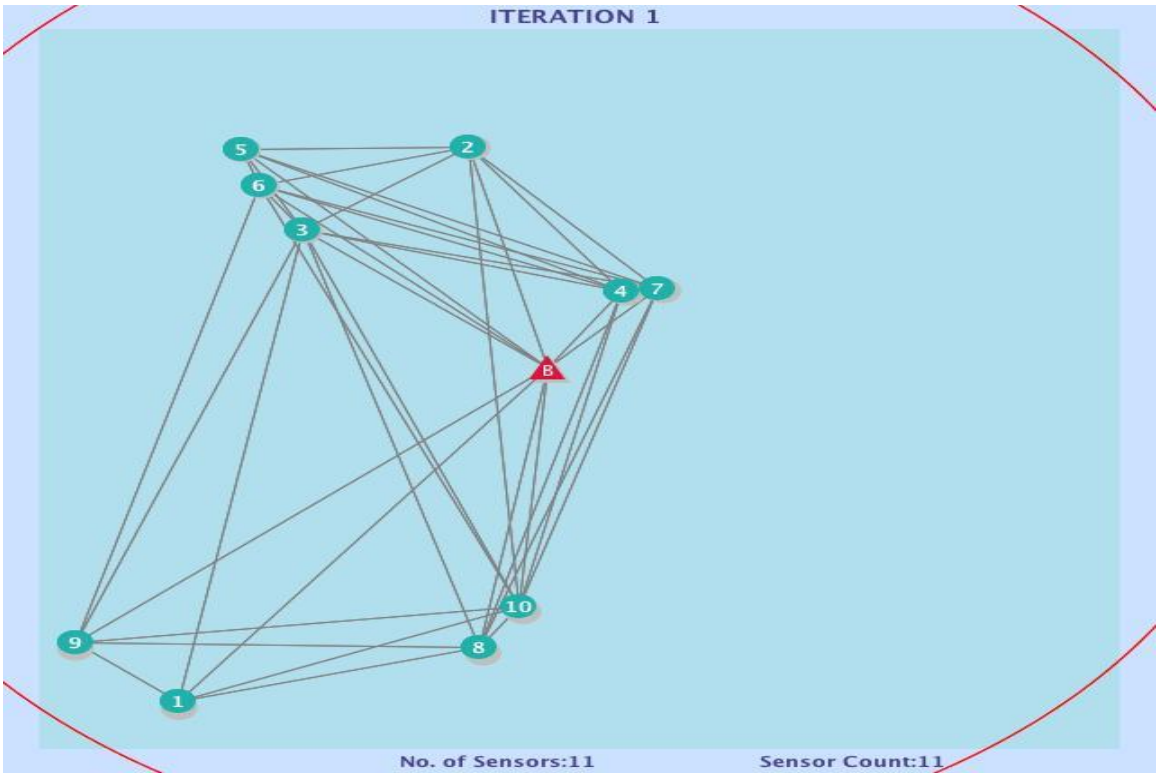

Figure 4.2. Formation of the Network in the Geographic Area.

4.2.1. Algorithm for connecting sensors

STEP 1. Pass communication radius, sensor name and all its information as arguments for this method.

STEP 2. For sensor index “a” = 1 to total number of sensors.

STEP 3. For sensor index “b” = 1 to total number of sensors.

STEP 4. If actual distance between sensor “a” and sensor “b” is greater than the communication radius, then go to STEP 10.

STEP 5. Update links table with destination sensor index value at index “a.”

STEP 6. Update hop count table at index “a” with 1.

STEP 7. Update the shortest distance table at index “a” with actual distance.

STEP 8. Increase sensor index “b” by 1.

STEP 9.

a. If sensor index “b” is greater than the total number of sensors, go to STEP 13.

b. Otherwise go to STEP 4.

STEP 10. Update links table with value -1 at index “a.”

STEP 11. Update the hop count table with the maximum integer value.

STEP 12. Update the distance table with the maximum double value.

STEP 13. Increase sensor index “a” by 1.

STEP 14.

a. If sensor index is greater than the total number of sensors then BREAK.

b. Otherwise go to STEP 3.

4.3. Exceptions

A sensor failure is considered an event that triggers this algorithm to run. We randomly fail sensors to test the algorithm. Sensor failure is not detected. In Figure 4.2, we can see that all the sensors can send information to the base station directly. There are a few scenarios in which the sensors might not be directly connected to the base station but connected to another sensor that is connected to the base station. There is another scenario in which the sensor might not be connected to any sensor. In Figure 4.3, sensor “2” is not connected to the network because it is not connected to any sensor that has a path to the base station. Hence the base station doesn’t know about the existence of sensor “2.” The results presented in this paper are calculated on networks where all the sensors can reach the base station after they are randomly deployed. To find out if all the sensors reach the base station, we run the modified shortest path algorithm on the network and check if all the sensors can reach the base station. Before we

calculate the shortest path, we broadcast a packet that has the sender's information, from all the sensors in the network, including the base station to all remaining sensors.

4.4. Modified Shortest Path Algorithm

This subsection explains the algorithm to find the shortest path between all the sensors in the network, including the base station. A sequence of connected sensors between two sensors in a network with minimal distance is the shortest path between those two sensors. When the sensors are randomly deployed and interconnected based on the communication radius of the sensors, the probability of all the sensors directly connected to the base station is less. Some of the sensors need to transmit information to neighboring sensors if the base station is not within communication range. There might be multiple paths through which the sensors can transmit information to the base station. The algorithm in Figure 4.4 is used to find the shortest path between all the sensors in the network. It is also used to find the alternative shortest path between two sensors if any sensor in the existing shortest path fails.

The algorithm in Figure 4.4 is Floyd's algorithm with additional Boolean tests of what should be true before the shortest path is calculated. In this algorithm, we have two arrays to save information and a third array for the Boolean test. They are the shortest distance between all the sensors in the network, the intermediate sensor to which the neighboring sensor, also called the linking sensor, has to pass the data in order to transmit it to the destination sensor. We use a virtual packet transmission algorithm, which uses the link matrix and finds the pairs of sensors that don't have a shortest path.

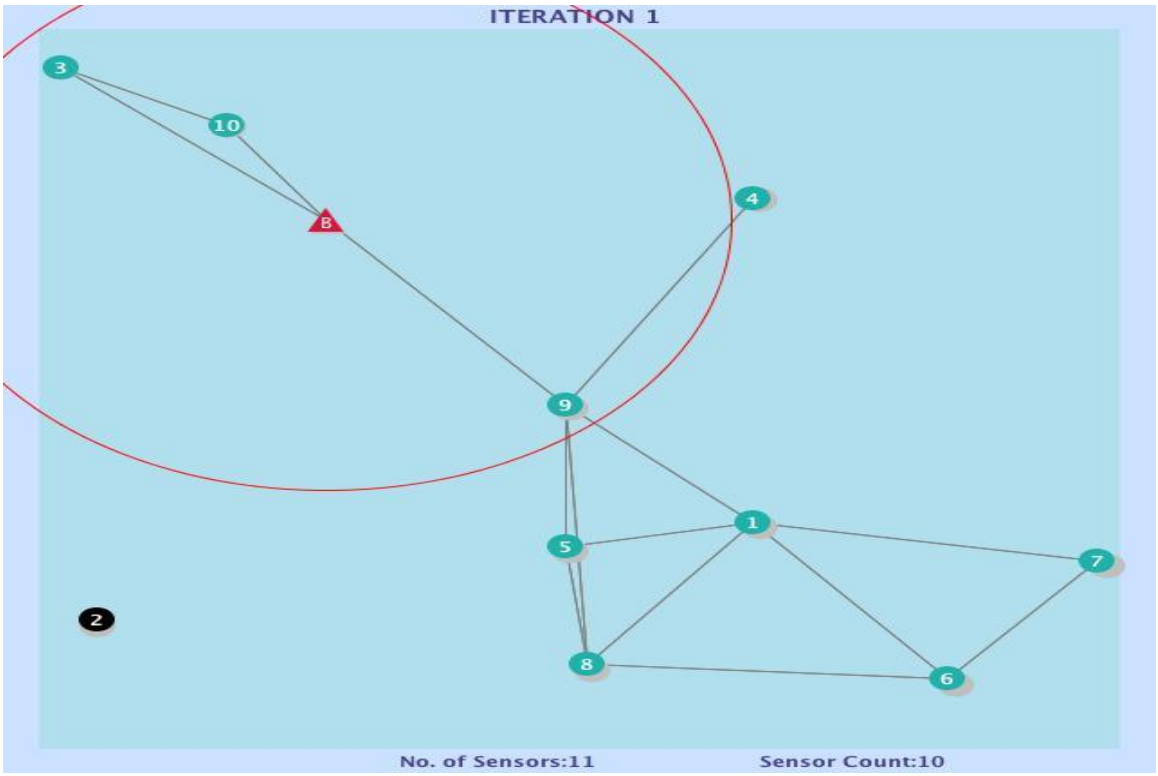

Figure 4.3. Formation of the Network in the Geographic Area.

STEP 1. Start with sensor “I” as the intermediate sensor, where “I” is the index of the sensor, to check if it can create the shortest paths in the network.

STEP 2.

a. If the index of “S” reaches the maximum index of the sensors, then increment index “I” and go to STEP 9.

b. Check if the sensor “I” is “ALIVE.” If not, we cannot use the sensor to create a path.

STEP 3. Start with sensor “S” as the source sensor, where “S” is the index of the sensor.

STEP 4.

a. If the index of “S” reaches the maximum index of the sensors, then increment index “I” and go to STEP 2.

b. Check if the indexes of sensors “S” and “I” are the same and sensor “S” is “ALIVE.” If the indexes are the same or sensor “S” is not “ALIVE,” then increment index “S” and check again.

c. Start with sensor “D” as the destination sensor, where “D” is the index of the sensor.

STEP 5.

a. Check if the indexes of sensors “D” and “I” are the same and sensor “D” is “ALIVE.” If the indexes are the same or sensor “D” is not “ALIVE,” then increment index “D” and check again.

b. If the index “D” reaches the maximum index of the sensors, then increment index “S” and go to STEP 4.

c. Check if there exists a shortest path between the pair of sensors with index “S” and “D.” If there is, increment index “D” and repeat STEP 6. Otherwise continue.

STEP 6. Check if the sum of distances between sensors “S” and “I” and sensors “I” and “D” is less than the existing path distance. If it’s greater then increment sensor index “D” and go to STEP 6.

STEP 7.

a. Update the link matrix with the intermediate sensor index.

b. Update the distance matrix with the sum of distances calculated in STEP 7.

c. Calculate the hop count between sensors “S” and “D” and update the hop count matrix.

d. Increment the sensor index “D” and go to STEP 6.

STEP 8. STOP

```
for (int k = 0; k < numSensors; k++)
{
   if (sensor[k].getState()==STATE.ALIVE)
   {
      for (int i = 0; i < numSensors; i++)
      {
         if ((k != i) && (sensor[i].getState()==STATE.ALIVE))
         {
            for (int j = 0; j < numSensors; j++)
            {
               if ((k != j) && (sensor[i].getPacket(j) == -1) && (sensor[j].getState()==STATE.ALIVE))
               {
                  if (sensor[i].getDist(j) > (sensor[i].getDist(k) + sensor[k].getDist(j)))
                  {
                     sensor[i].setLink(j, k);
                     sensor[i].setDist(j, sensor[i].getDist(k)+ sensor[k].getDist(j));
                     sensor[i].setHopCount(j,sensor[i].getHopCount(k)+ sensor[k].getHopCount(j));
                  }
               }
            }
         }
      }
   }
}
```

Figure 4.4. Modified Shortest Path Algorithm.

We calculate the shortest path between all the sensors in the network by starting with the sensor with index "0" as the intermediate sensor. We are checking if a shortest path exists between a pair of sensors in the network not directly connected to each other with sensor "0" as the intermediate sensor. Next, we increment the sensor index by 1 and check if a shortest path exists between a pair of sensors in the network other than the path we already know. We continue to check with all the sensors as the intermediate sensor by incrementing the index by 1. Using this method, we detect a path for transmitting data between sensors that don't have a link between them. The path might consist of more than one intermediate sensor. We don't

save the whole path of transmission; instead, we just save the neighboring sensor to which the data has to be transmitted. The neighboring sensor might not have a direct link to the sender, but it might have an intermediate sensor that transmits the data from the sender. Hence the information we save in the array about the links is like a loop that has all the transmission path information but not the whole path. When a sensor fails in the network, this algorithm is triggered to find the shortest path between sensors that were affected by the failure. We need to find the affected paths since we don't save the whole path of transmission between the sensors. In Figure 4.4, a Boolean test "sensor [I].getPacket(j) == -1" is done to check if a shortest path has to be calculated between the set of sensors "S" and "D." If the Boolean test is true, the existing shortest path was affected due to a sensor failure. The algorithm in section 4.5 discusses how an affected path can be found that helps save time instead of recalculating the shortest path for the whole network.

4.5. Path Check Algorithm

This subsection explains an algorithm that checks the existing paths to see if they are affected because of a sensor failure. We can assume it as a virtual broadcasting of packets from all the sensors in the network to all the sensors. The packet contains sender information. The algorithm in Figure 4.6 uses the information about the neighboring sensors from the modified shortest path algorithm to check whether the existing shortest paths are affected because of the failure of a sensor. Each sensor has information about all the sensors that it can communicate. In this algorithm we check to see if any paths are affected because of the failed sensor.

STEP 1. Initialize the information that each sensor has about the communicating sensors to null.

STEP 2. For sensor index “a” = 1 to total number of sensors.

STEP 3. If sensor “a” is in the “ALIVE” state, continue, or else go to STEP 10.

STEP 4. For sensor index “b” = 1 to total number of sensors.

STEP 5. If sensor “b” is in the “ALIVE” state, continue, or else go to STEP 9.

STEP 6. If the link between sensor “a” and sensor “b” is “-1,” then set the Packet value to “-1” and go to STEP 9, or else continue.

STEP 7. If the link between sensor “a” and sensor “b” is “b,” then set the Packet value to “a” and go to STEP 9, or else continue.

STEP 8. If check_Path (sensor “a,” link between sensor “a” and sensor “b”) then call method send_Packet (sensor “a,” a linking sensor between sensor “a” and sensor “b,” sensor “b”).

STEP 9. Increase the sensor “b” index by 1.

STEP 10. Increase the sensor “a” index by 1.

```
private boolean check_Path(int sender, int receiver) {
   boolean check = false;
   if (network.sensor[sender].getLink(receiver) == -1) {
      return false;
   } else if (network.sensor[sender].getLink(receiver) == receiver) {
      check = true;
   } else if (check_Path(network.sensor[sender].getLink(receiver),
         receiver)) {
      check = check_Path(sender, network.sensor[sender].getLink(receiver));
   }
   return check;
}
```

Figure 4.5. Check Path Algorithm.

In the pathCheck algorithm we see two methods: check_Path and send_Packet. We show the check_Path algorithm in Figure 4.5. In the pathCheck algorithm, step 8 is executed if

the link between sensor “a” and sensor “b” is neither “-1” nor sensor “b.” The link between sensor “a” and sensor b” is some sensor “x” in the network. To virtually send a packet from sensor “a” to sensor “b,” the check_Path algorithm checks if the path between sensor “a” and sensor “x” is affected because of a sensor failure in the network. If the path is affected it returns a Boolean value of “FALSE,” otherwise it returns a Boolean value of “TRUE.” There might be a chance that sensor “a” and sensor “x” are connected through an intermediate sensor. We check both the individual paths to see if they are affected. This is a recursive method that checks the whole path up to the intermediate sensor and returns a Boolean value.

```
private void pathCheck() {
    network.initPacket();
    for (int i = 0; i < network.getNumSensors(); i++) {
        if (network.sensor[i].getState() == STATE.ALIVE) {
            for (int j = 0; j < network.getNumSensors(); j++) {
                if (network.sensor[j].getState() == STATE.ALIVE) {
                    if (network.sensor[i].getLink(j) == -1) {
                        network.sensor[i].setPacket(j, -1);
                    } else if (network.sensor[i].getLink(j) == j) {
                        network.sensor[i].setPacket(j, i);
                    } else {
                        if (check_Path(i, network.sensor[i].getLink(j)))
                            send_Packet(i, network.sensor[i].getLink(j), j);
                    }
                }
                if (network.sensor[i].getPacket(j) == -1) {
                    network.sensor[i].setLink(j, -1);
                    network.sensor[i].setDist(j, Double.POSITIVE_INFINITY);
                    network.sensor[i].setHopCount(j, 0);
                }
            }
        }

    }
}
```

Figure 4.6. PathCheck Algorithm.

4.6. Algorithm

The algorithm is developed to obtain the shortest path in a wireless sensor network when sensors fail. The total number of sensors and the communication radius are supplied as input parameters. Each sensor in the wireless sensor network communicates with all the other sensors in the network, including the base station. All the sensors within communication range of the sensor are its neighbors. A packet is sent with its sensor's name from each sensor in the network to all other sensors in the network. The algorithm finds the shortest path in terms of distance between each pair of sensors in the network, including the base station (SOURCE, DESTINATION), using Floyd's algorithm if the packet sent by the source is not received by the destination. The network is checked to see if all the sensors can reach the base station. Sensors are failed randomly one at a time and the alternate shortest path is found for all pairs of sensors if the failed sensor exists in the existing shortest path. The algorithm in this paper is explained below.

STEP 1. The total number of sensors and communication radius are the inputs.

STEP 2. The base station is placed at the coordinates (X, Y), where the distance between (0, 0) and (X, Y) is equal to the communication radius.

STEP 3. The remaining sensors are placed randomly in the geographical area.

STEP 4. A sensor network is formed connecting all the sensors.

STEP 5. Packets from each sensor are sent to all the sensors in the network.

STEP 6. Find the shortest path between the pair of sensors (SOURCE, DESTINATION) for which the destination hasn't received the source packet.

STEP 7. If any sensor does not have a path to the base station, then BREAK.

STEP 8. Fail sensors randomly, excluding the base station.

STEP 9. Update the sensor network with the failed sensors.

STEP 10.

a. If all the remaining sensors do not have a path to base station, then BREAK.

b. Otherwise go to STEP 5.

5. TEST RESULTS

This section provides the experimental analysis of the performance of the network in finding the shortest path based on the distance between the sensors and CPU Process Time. We also analyze the probability of the creation of a network based on the total number of sensors and the radius.

5.1. Network Creation Evaluation

When the network is created, we check to see if all the sensors can reach the base station. We calculate the shortest path between all the sensors in the network and check if any of the sensors cannot reach the base station. If all the sensors have a path to the base station, then all the sensors in the network can communicate with each other and vice versa. Figure 5.1 shows the graphical representation of the probability of the creation of a network vs. the total number of sensors for different values of radius in a 600 x 600 area. The data for the above graph is given in Table 5.1.The interval for the number of sensors is from 20 to 190 with a unit difference of 10, and the radius is from 60 to 190 with a unit difference of 10 in this result set. We can see from the graph that the probability of creating a network increases with an increase in the number of sensors and gradually stops increasing from a specific total number of sensors. Since sensors are deployed randomly in the area, we would like to know how many sensors are required to be deployed based on the radius. The information that we can gather from this graph is the minimum number of sensors needed to make sure that the graph is created when the network area and the radius of the sensor are known.

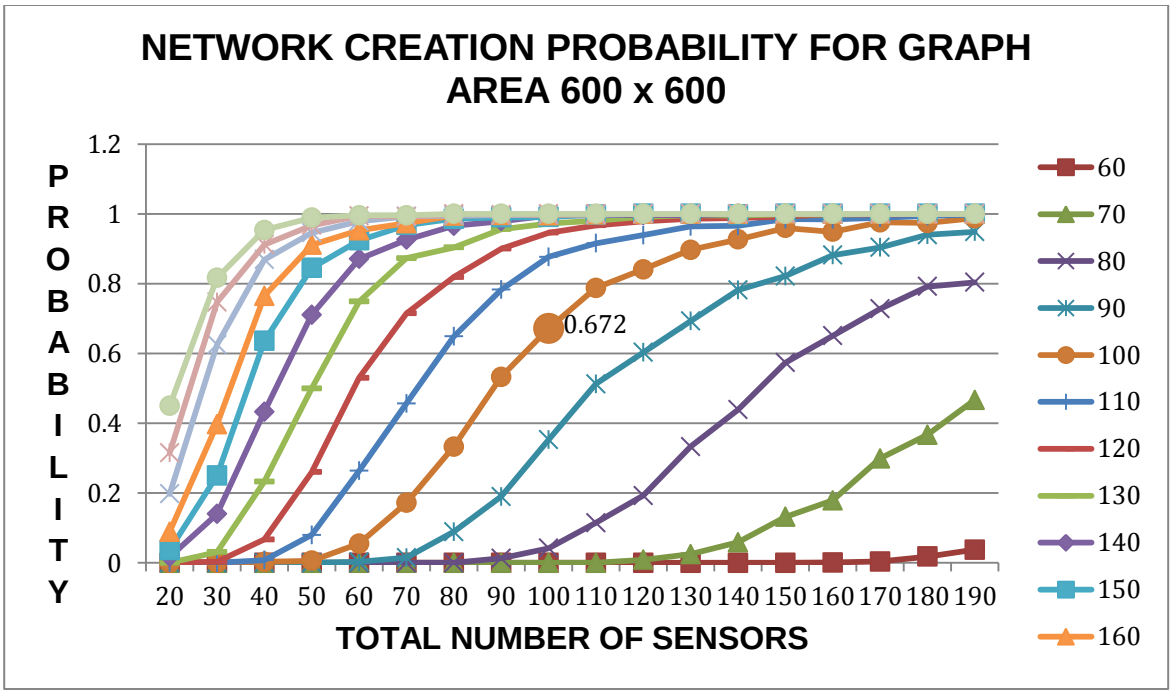

Figure 5.1. Network Creation Probabilities for Graph Area 600 x 600.

The graph in Figure 5.2 represents the same information for an area of 800 x 800. The data for the graph is shown in Table 5.2. The interval for the number of sensors is from 20 to 190 with a unit difference of 10, and the radius is from 60 to 190 with a unit difference of 10 in this result set. When the results are compared to the graph in Figure 5.1, the minimum number of sensors required for the creation of a network changes by a large number. The probability of creating a network for 190 total sensors with a radius of 100 in the 600 x 600 area is 0.988, whereas in the 800 x 800 area it is 0.694. That is a difference of almost 0.3 in the creation of a network with the same number of sensors when the area is increased 200 x 200. From this we can approximately calculate the minimum number of sensors required to create a network when the probability is calculated for one area size. In Figure 5.1, the probability of creating a network with 100 sensors and 100 as the radius of the sensor is 0.672. We can calculate approximately

Table 5.1. Network Creation Probabilities for Graph Area 600 x 600

sensors	60	70	80	90	100	110	120	130	140	150	160	170	180	190
20	0	0	0	0	0	0	0.001	0	0.017	0.037	0.089	0.198	0.315	0.45
30	0	0	0	0	0	0	0.003	0.031	0.14	0.25	0.397	0.625	0.747	0.817
40	0	0	0	0	0.003	0.008	0.067	0.233	0.433	0.636	0.765	0.869	0.912	0.954
50	0	0	0	0	0.006	0.08	0.26	0.5	0.711	0.845	0.911	0.946	0.967	0.989
60	0	0	0	0.003	0.055	0.264	0.53	0.749	0.871	0.924	0.953	0.979	0.994	0.996
70	0	0	0	0.015	0.172	0.457	0.715	0.873	0.926	0.968	0.973	0.993	0.992	0.996
80	0	0	0	0.089	0.333	0.649	0.819	0.904	0.966	0.986	0.995	0.995	1	1
90	0	0	0.013	0.19	0.533	0.783	0.9	0.956	0.979	0.99	0.997	0.996	0.998	1
100	0	0	0.042	0.353	0.672	0.877	0.946	0.972	0.995	0.991	0.996	0.998	1	1
110	0	0	0.115	0.513	0.788	0.916	0.967	0.981	0.996	0.998	0.999	0.998	1	1
120	0	0.009	0.193	0.603	0.841	0.939	0.979	0.992	0.997	1	1	1	1	1
130	0	0.025	0.333	0.693	0.897	0.964	0.986	0.996	0.996	1	1	1	1	1
140	0	0.059	0.439	0.782	0.927	0.966	0.988	0.995	1	0.999	1	1	1	1
150	0	0.132	0.574	0.822	0.96	0.985	0.993	0.999	1	1	1	1	1	1
160	0.001	0.179	0.651	0.882	0.949	0.984	0.999	0.998	0.999	1	1	1	1	1
170	0.004	0.299	0.728	0.904	0.976	0.989	0.998	1	1	1	1	1	1	1
180	0.018	0.367	0.792	0.94	0.974	0.994	1	1	1	1	1	1	1	1
190	0.038	0.467	0.804	0.949	0.988	0.998	1	1	1	1	1	1	1	1

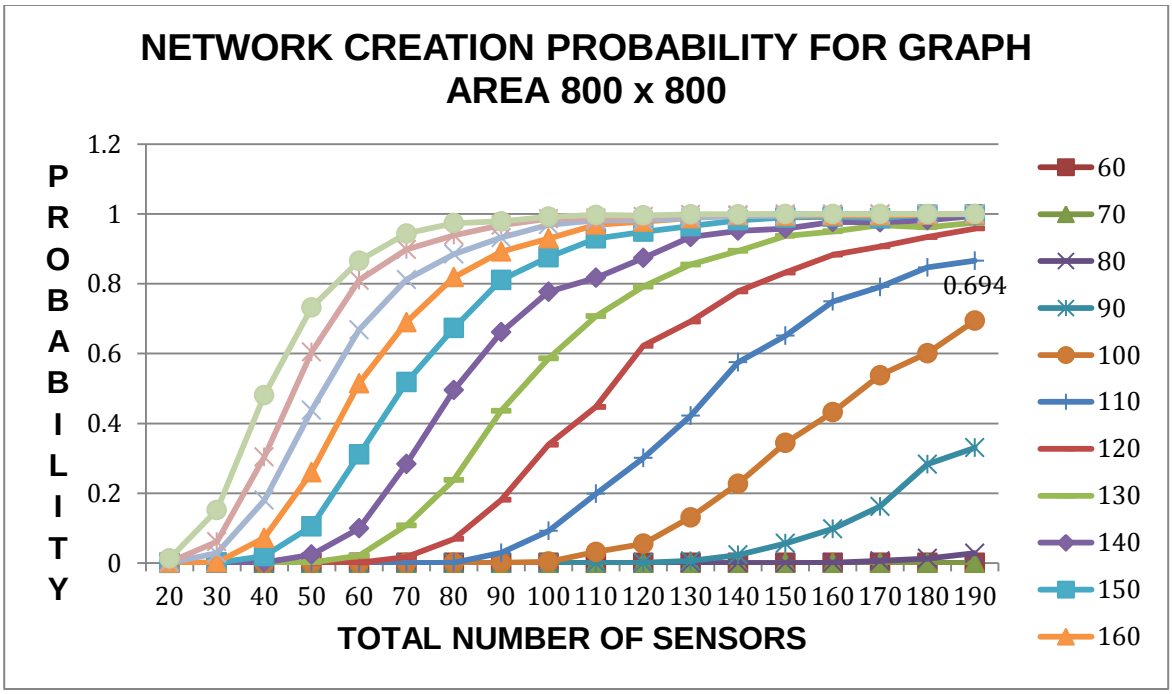

Figure 5.2. Network Creation Probabilities for Graph Area 800 x 800.

the minimum number of sensors required for an 800 x 800 area with the same probability. The minimum number of sensors required for an 800 x 800 area = (Minimum number of sensors required by 600 x 600)* (800 * 800) / (600 * 600). The approximate minimum number of sensors required by an 800 x 800 area to create a network with a probability of 0.672, with a sensor radius of 100 is (100 * 800 * 800) / (600 * 600), and is approximately 178. The probability of creating a graph with 190 sensors and a radius of 100 in an 800 x 800 area is 0.694. We can calculate the approximate number of sensors required using a test case scenario of just one network area.

5.2. Test Cases

In the following set of test cases for each random network generated, a sensor is selected randomly, and the difference in the shortest path and hop count is calculated assuming

that it fails. The number of sensors and the communication radius is different for each random network generated.

5.2.1. Test case 1

In this test case the network is generated with 11 sensors with a communication radius of 300 in a network area of 600 x 600. These sensors are deployed randomly using the random graph generation algorithm discussed in section 3.2. The resultant network is shown in Figure 5.4. A sensor is selected randomly and is assumed to be the failed sensor. The impact of the failed sensor on the network is seen by calculating the difference in the shortest path distances and hop counts. We can see in Figure 5.5 the impact of sensor failures while measuring distance on a Random 11-Sensors Network. The second sensor failure has an impact on the sensor network with respect to the shortest distance and hop count. There is a difference of 751.22 in the shortest distance after the second sensor failure, as shown in Figure 5.5. There is no difference of shortest distances after any other sensor failures. There is a difference of 3 in the hop counts after the second sensor failure as shown in Figure 5.5. There is no difference of hop counts after any other sensor failures. The number of sensors that can reach the base station after each sensor failure is shown in Figure 5.3. We can see that the number of sensors that can reach the base station after failure 4 is seven, and after failure 5 is three. We can see in Figure 5.4.5 that four failed sensors are shaded in orange. Figure 5.4.6 shows sensor 10 fail, and the impact of the failure is seen in Figure 5.4.6, where three sensors are shaded in black and they cannot reach the base station because of the failure. Hence we can conclude that the failure of a sensor impacted the shortest path distance, the hop count after the second failure, and the connectivity of other sensors after the fifth failure in this randomly generated 11 sensors network.

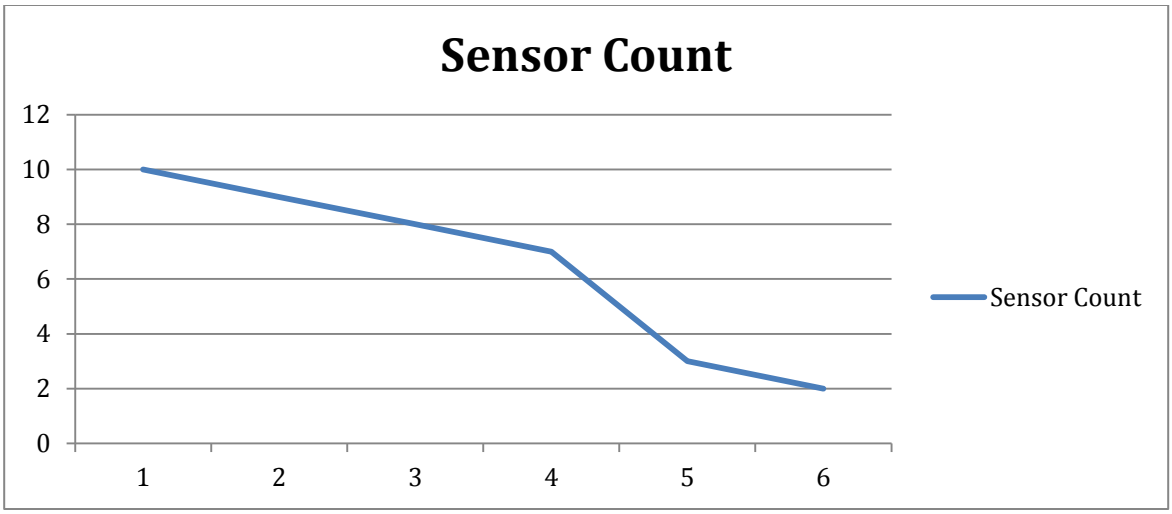

Figure 5.3. Number of Sensors Reached by the Base Station after Sensors Fail.

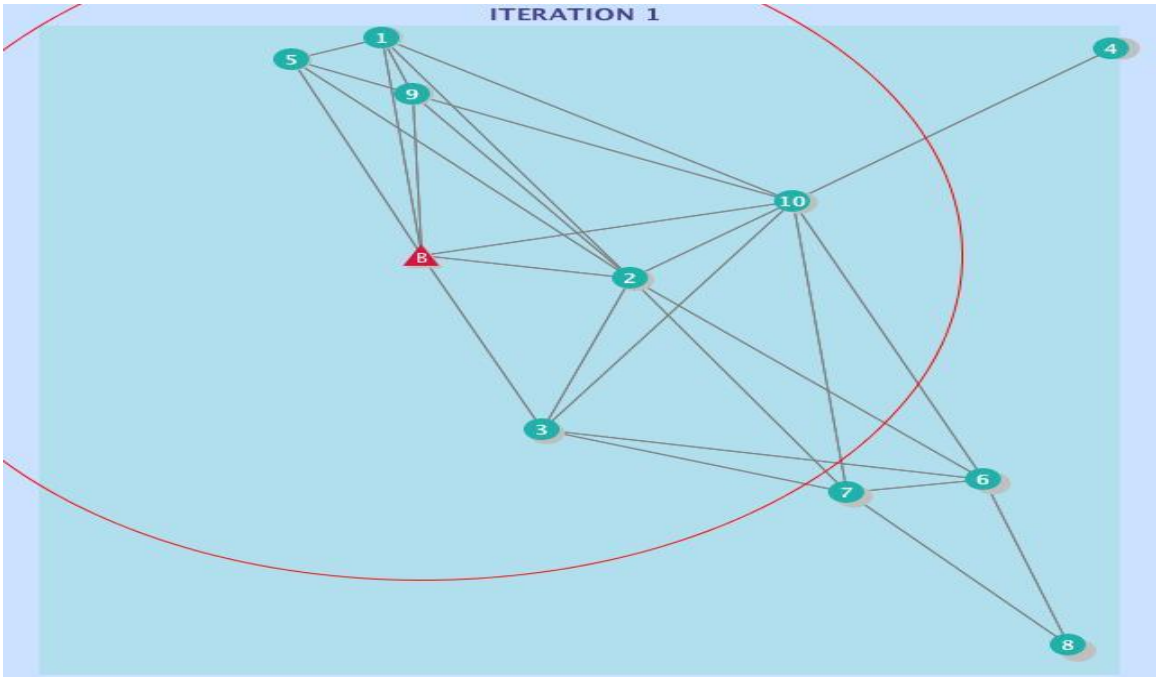

Figure 5.4. A Random 11-Sensor Network.

Table 5.2. Network Creation Probabilities for Graph Area 800 x 800.

sensors	80	90	100	110	120	130	140	150	160	170	180	190
20	0	0	0	0	0	0	0	0	0.001	0.001	0.003	0.013
30	0	0	0	0	0	0	0	0.001	0.002	0.028	0.061	0.152
40	0	0	0	0	0	0	0.002	0.02	0.072	0.18	0.304	0.481
50	0	0	0	0	0	0.002	0.024	0.105	0.26	0.438	0.605	0.732
60	0	0	0	0	0.001	0.021	0.099	0.311	0.515	0.668	0.81	0.866
70	0	0	0	0	0.017	0.108	0.284	0.518	0.69	0.812	0.898	0.944
80	0	0	0	0.001	0.069	0.238	0.495	0.673	0.819	0.885	0.939	0.974
90	0	0	0.001	0.029	0.181	0.436	0.661	0.811	0.892	0.932	0.968	0.979
100	0	0	0.004	0.092	0.339	0.586	0.777	0.876	0.93	0.969	0.986	0.992
110	0	0	0.032	0.198	0.447	0.707	0.817	0.93	0.969	0.982	0.988	0.997
120	0	0.001	0.055	0.301	0.622	0.792	0.874	0.949	0.978	0.982	0.992	0.996
130	0	0.006	0.131	0.422	0.692	0.855	0.934	0.965	0.988	0.989	0.998	0.999
140	0	0.023	0.227	0.575	0.778	0.894	0.952	0.982	0.997	0.994	0.997	0.999
150	0	0.056	0.344	0.651	0.832	0.937	0.958	0.99	0.995	0.996	1	1
160	0.001	0.098	0.432	0.749	0.883	0.95	0.977	0.992	0.996	0.999	0.999	1
170	0.006	0.162	0.538	0.791	0.907	0.968	0.976	0.988	0.996	1	0.999	1
180	0.013	0.283	0.601	0.847	0.934	0.962	0.982	0.998	0.998	1	1	1
190	0.028	0.33	0.694	0.866	0.958	0.975	0.995	0.999	1	1	1	1

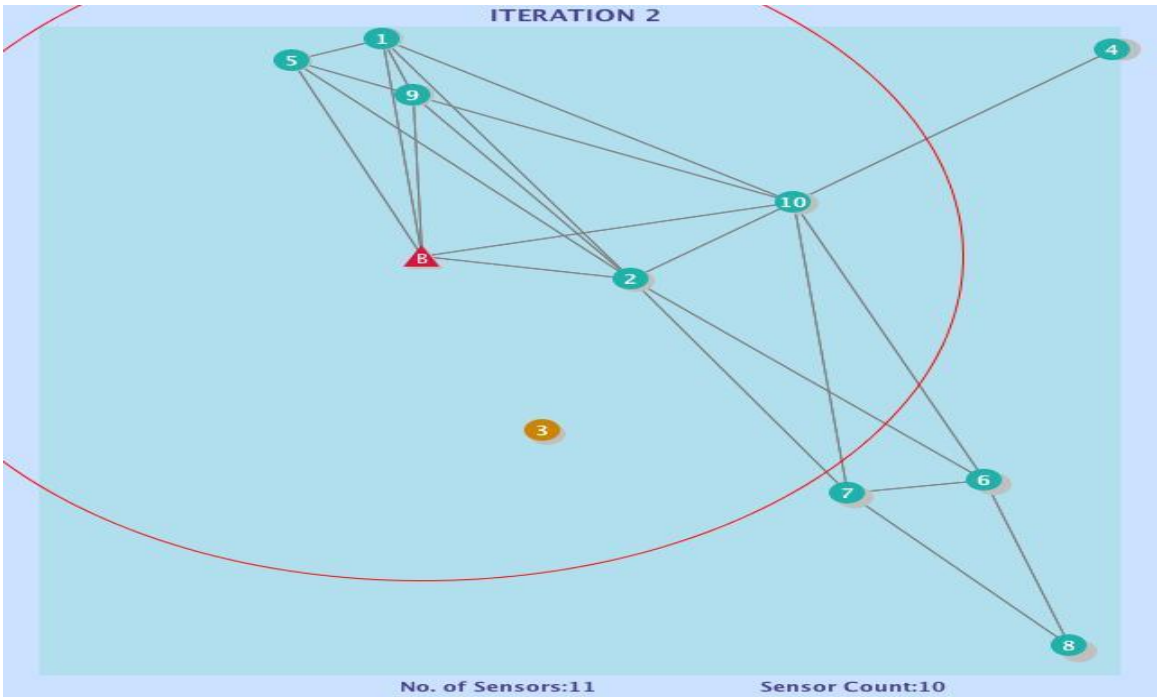

Figure 5.4.1. A Random 11-Sensor Network after One Sensor Fails.

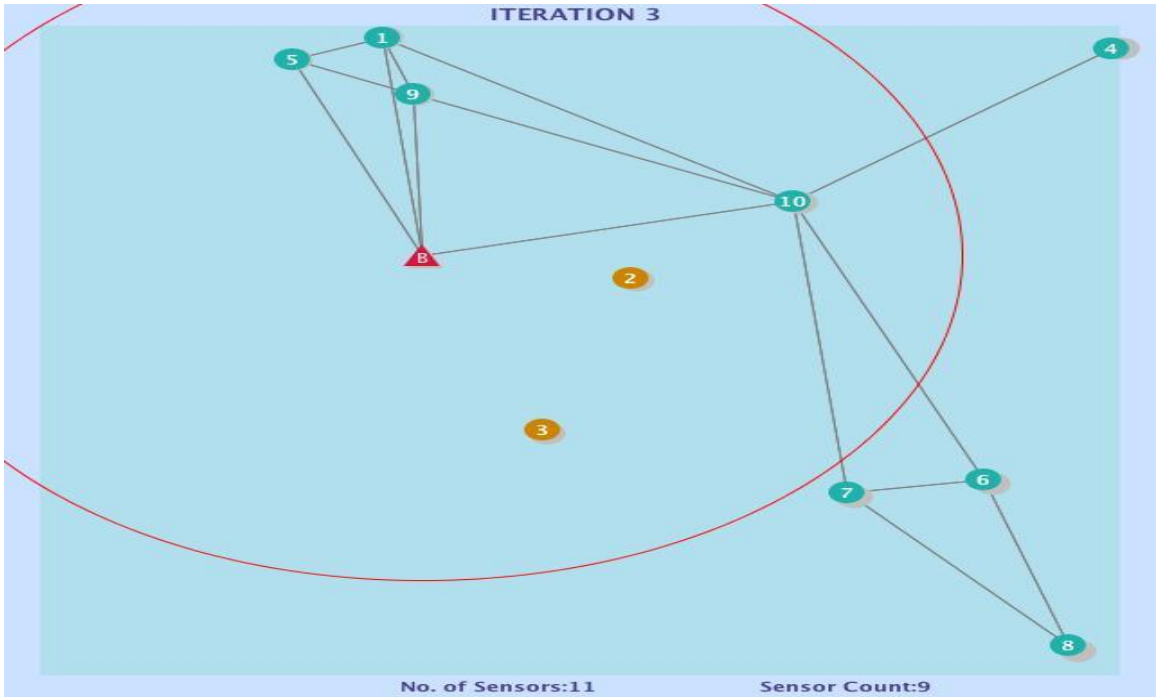

Figure 5.4.2. A Random 11-Sensor Network after Two Sensors Fail.

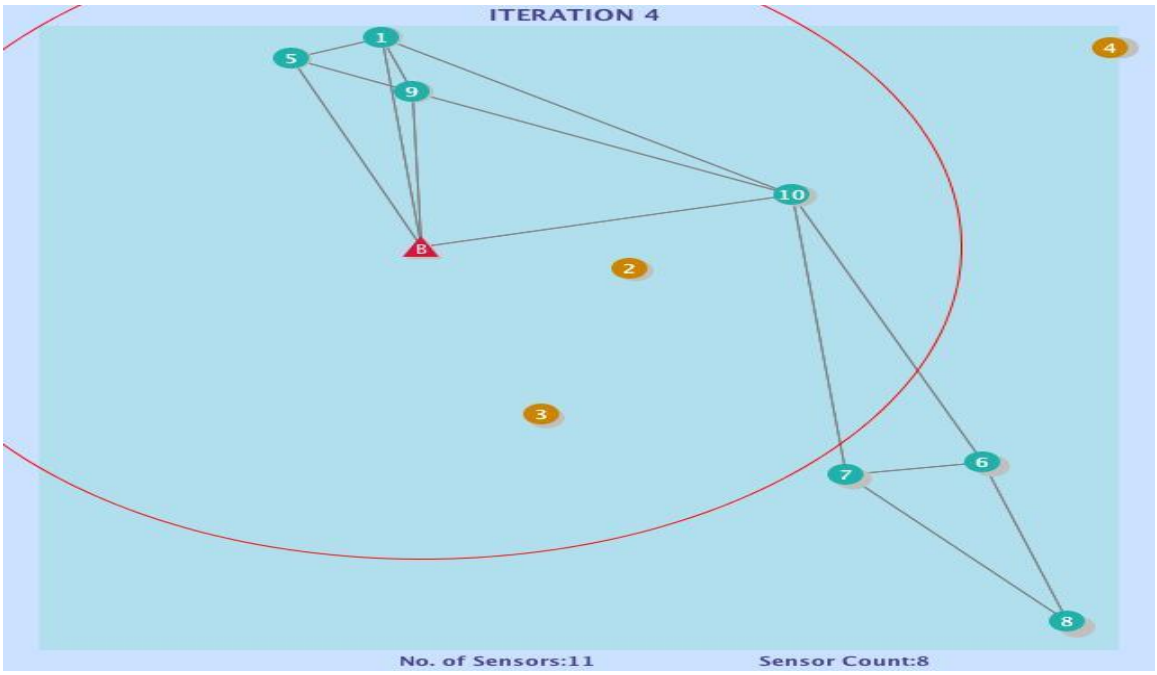

Figure 5.4.3. A Random 11-Sensor Network after Three Sensors Fail.

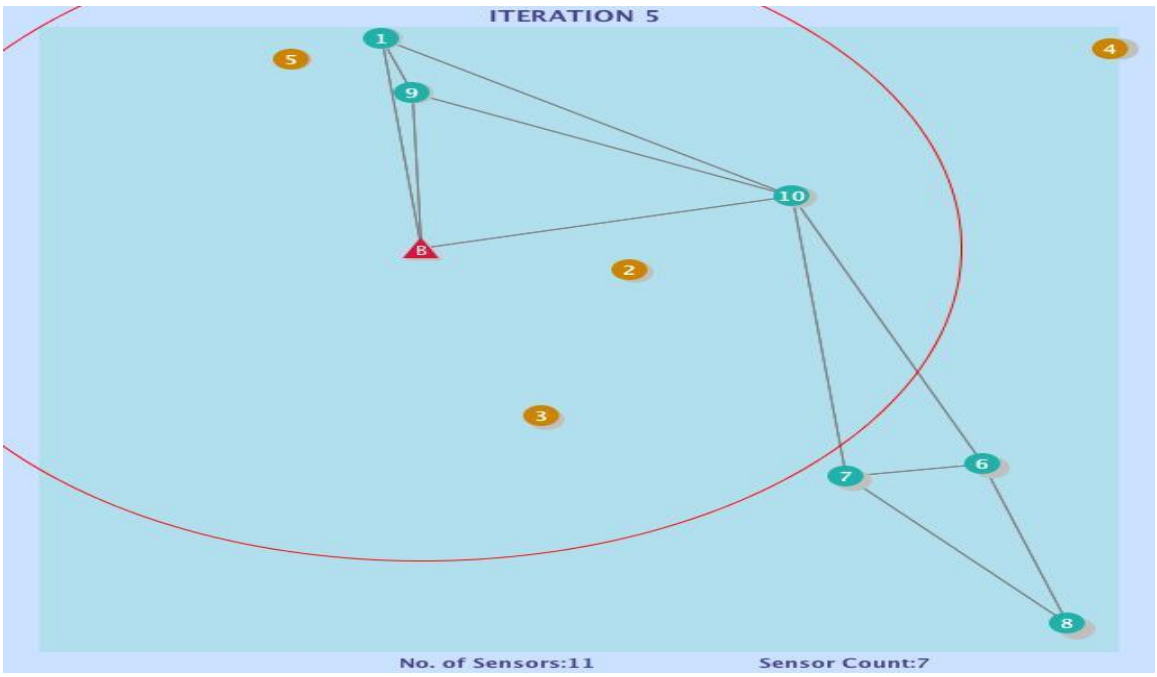

Figure 5.4.4. A Random 11-Sensor Network after Four Sensors Fail.

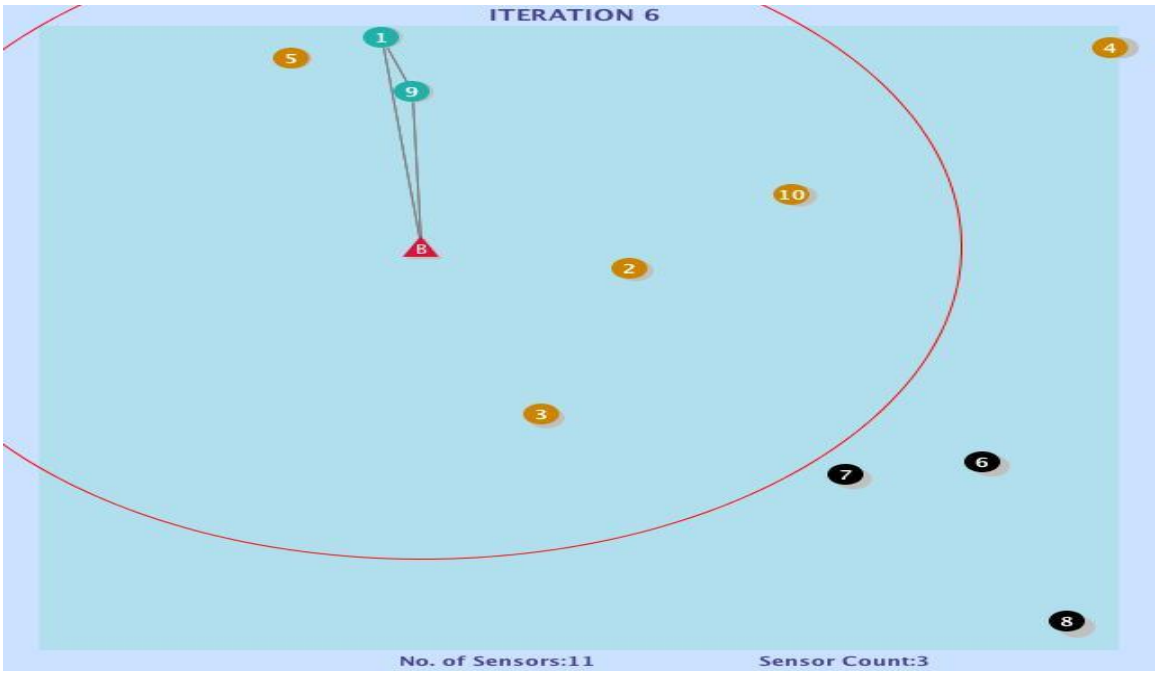

Figure 5.4.5. A Random 11-Sensor Network after Five Sensors Fail.

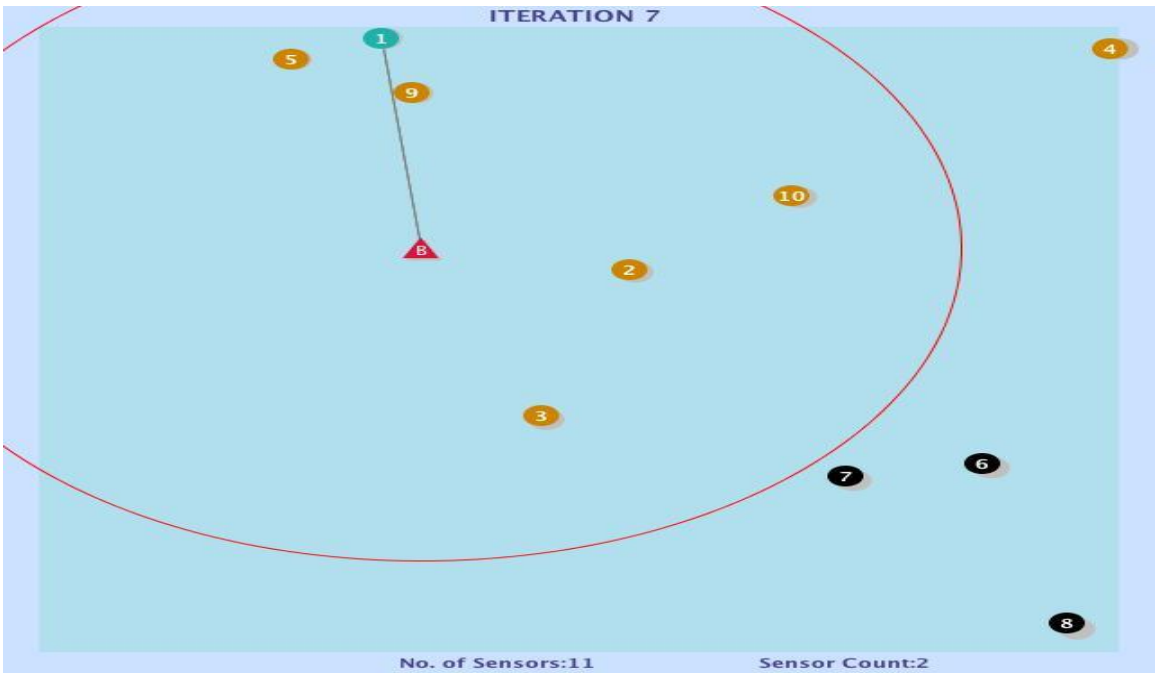

Figure 5.4.6. A Random 11-Sensor Network after Six Sensors Fail.

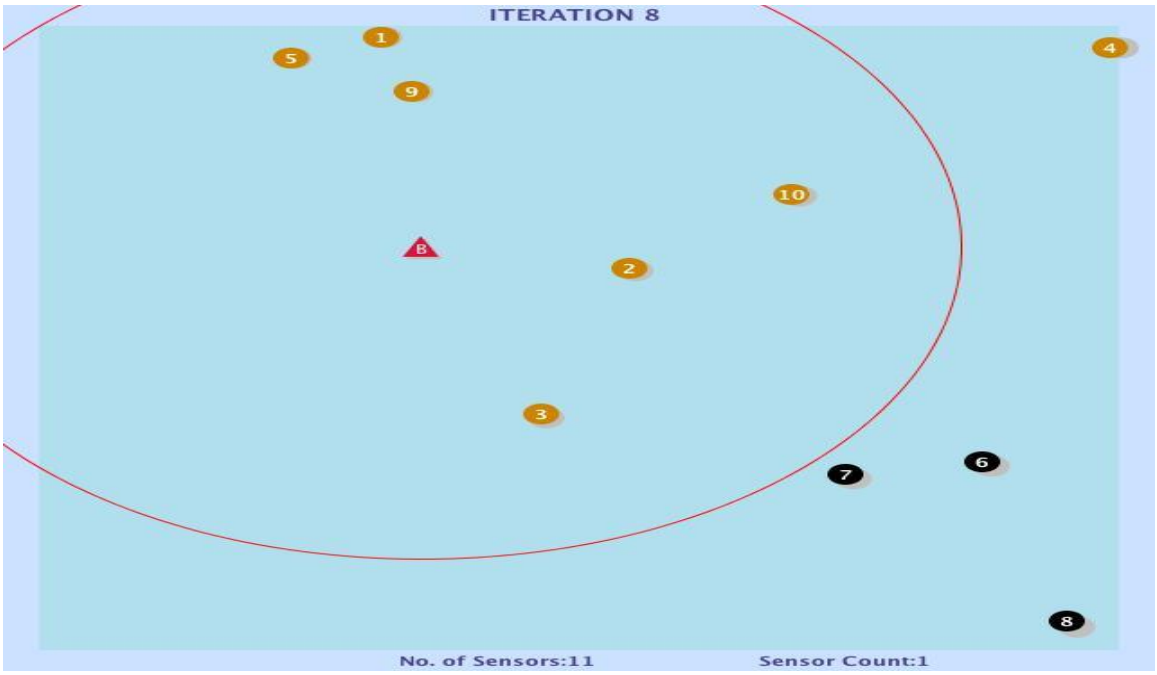

Figure 5.4.7. A Random 11-Sensor Network after Seven Sensors Fail.

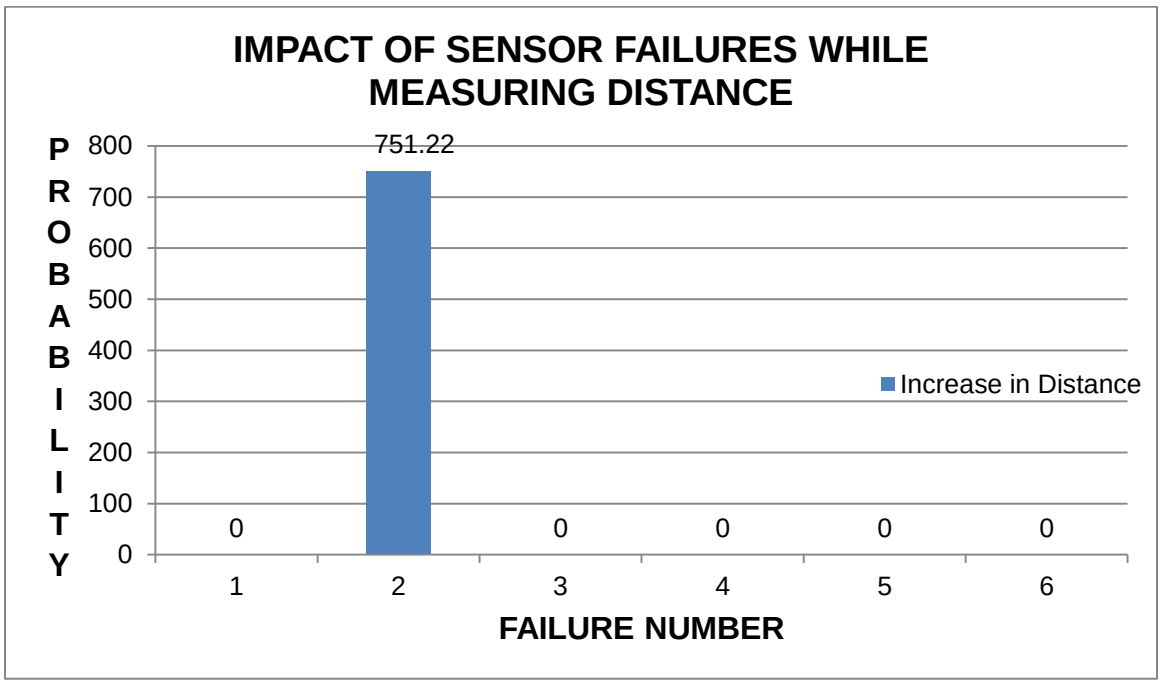

Figure 5.5. Impact of Sensor Failures while Measuring Distance.

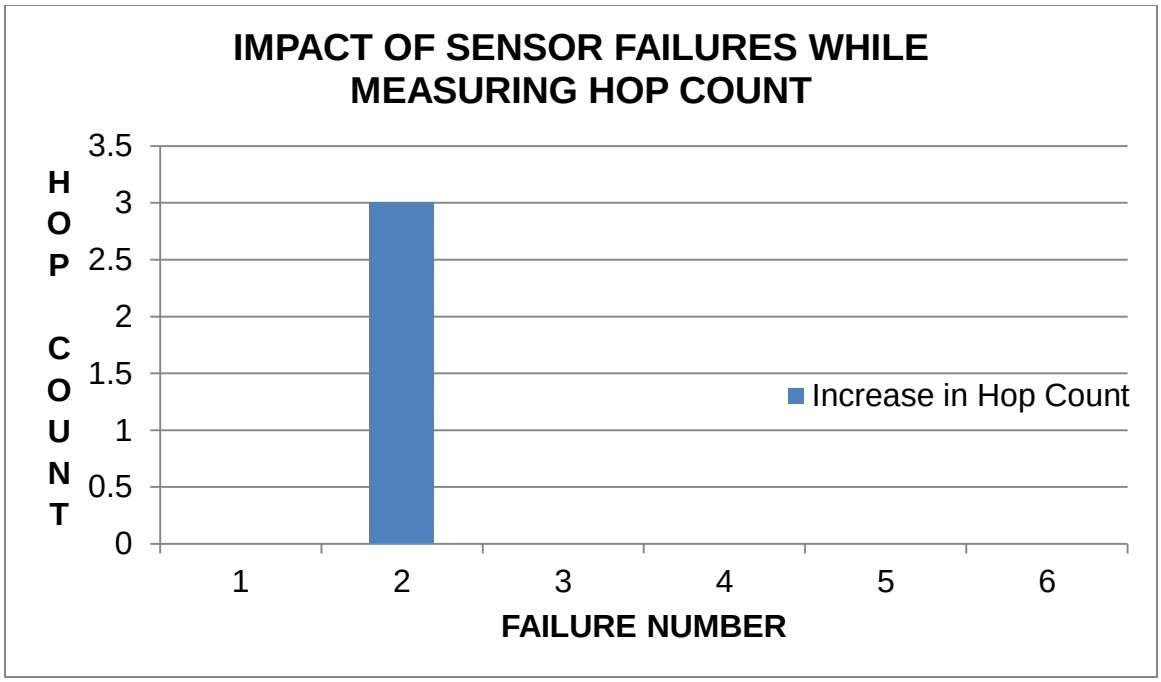

Figure 5.6. Impact of Sensor Failures while Measuring Hop Count.

5.2.2. Test case 2

In this test case, the network is generated with 21 sensors with a communication radius of 180 in a network area of 600 x 600. These sensors are deployed randomly using the random graph generation algorithm discussed in section 4.2. The resultant network is shown in Figure 5.8. A sensor is selected randomly and is assumed to be the failed sensor. The calculations done in this test case are the same as in test case 1. We can see in Figure 5.7 that the number of failures that occurred until the base station was not reachable by any sensor was 8. The previous test case had 6 failures. The impact of failures is more in this test case compared to the previous test case. In Figure 5.7, the number of sensors after failure 4 is 16, and after failure 5 is 5. If we can compare Figure 5.8.4 and Figure 5.8.5, the failure of sensor 13 has caused 10 other sensors to lose connectivity to the base station. The impact of the failures on the total shortest distance

in the network occurred at failures 2 and 4 as shown in Figure 5.9. The impact on the total hop count occurred at failure 4, as shown in Figure 5.10.

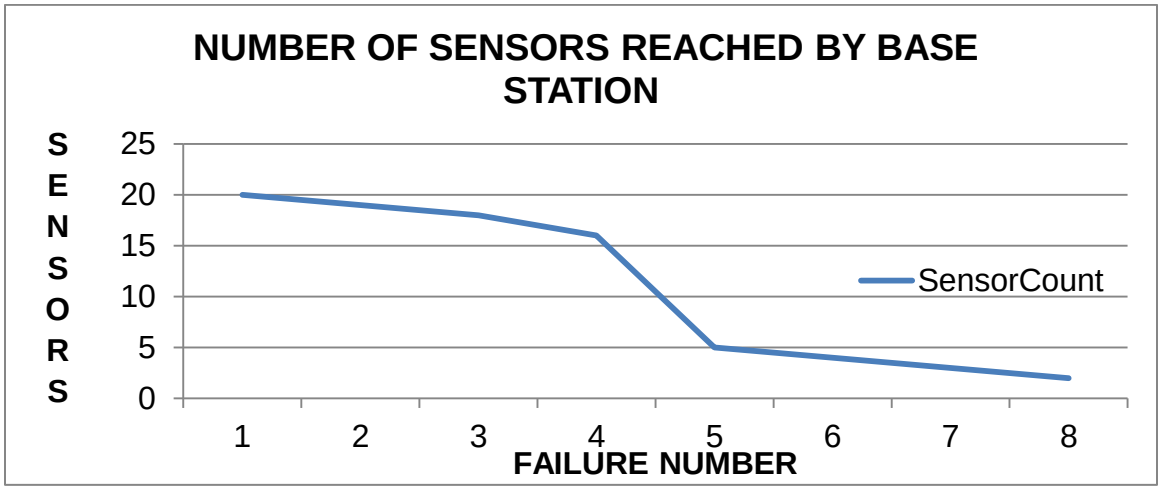

Figure 5.7. Number of Sensors Reached by the Base Station after Sensors Fail.

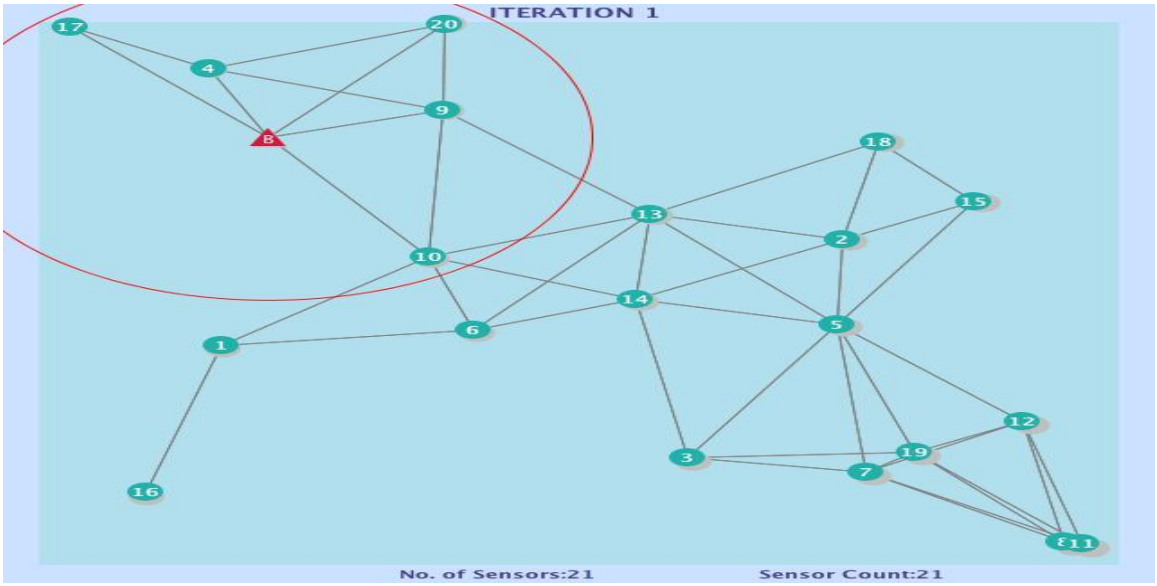

Figure 5.8. A Random 21-Sensor Network.

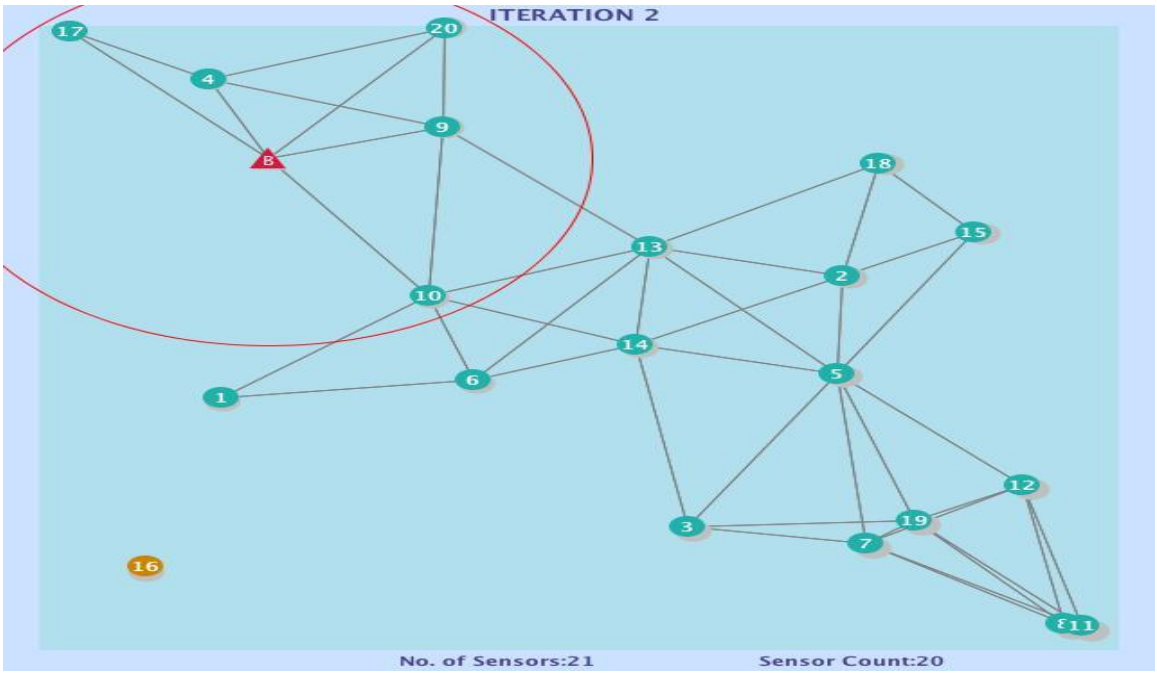

Figure 5.8.1. A Random 21-Sensor Network after One Sensor Fails.

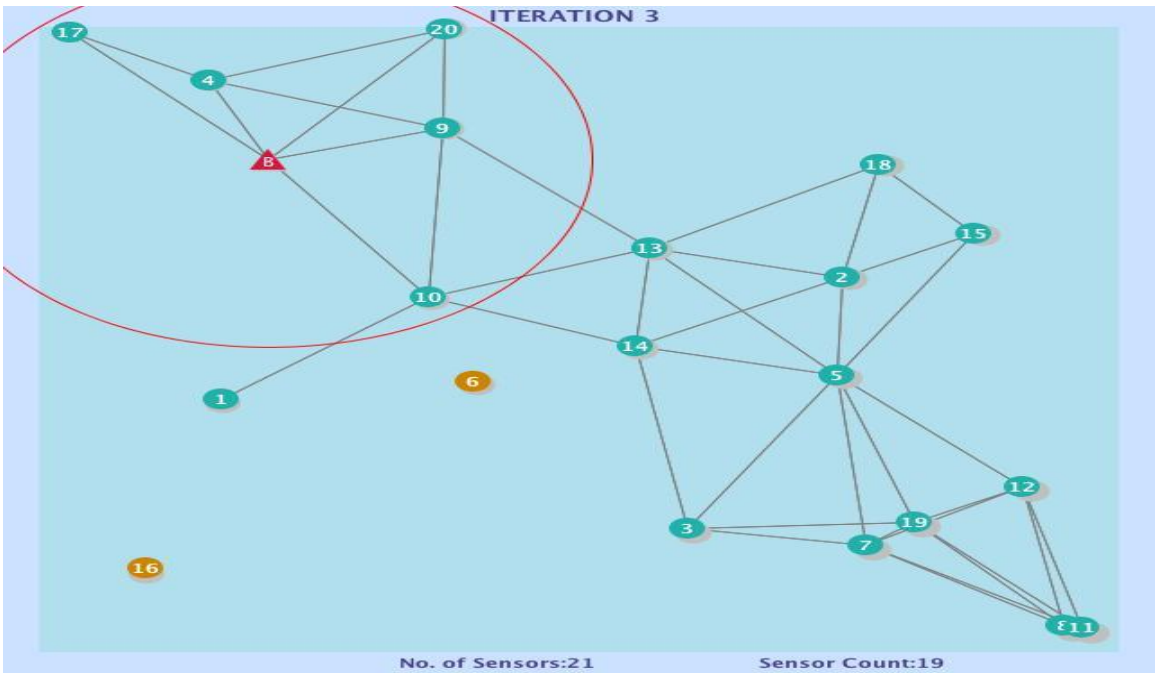

Figure 5.8.2. A Random 21-Sensor Network after Two Sensors Fail.

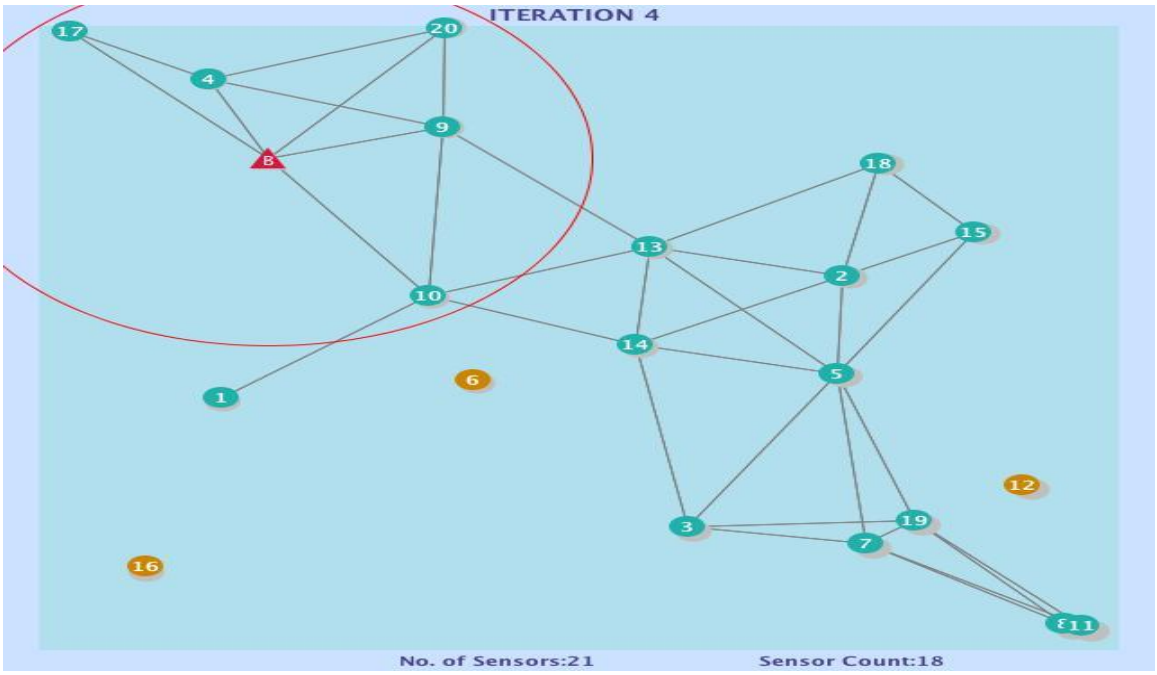

Figure 5.8.3. A Random 21-Sensor Network after Three Sensors Fail.

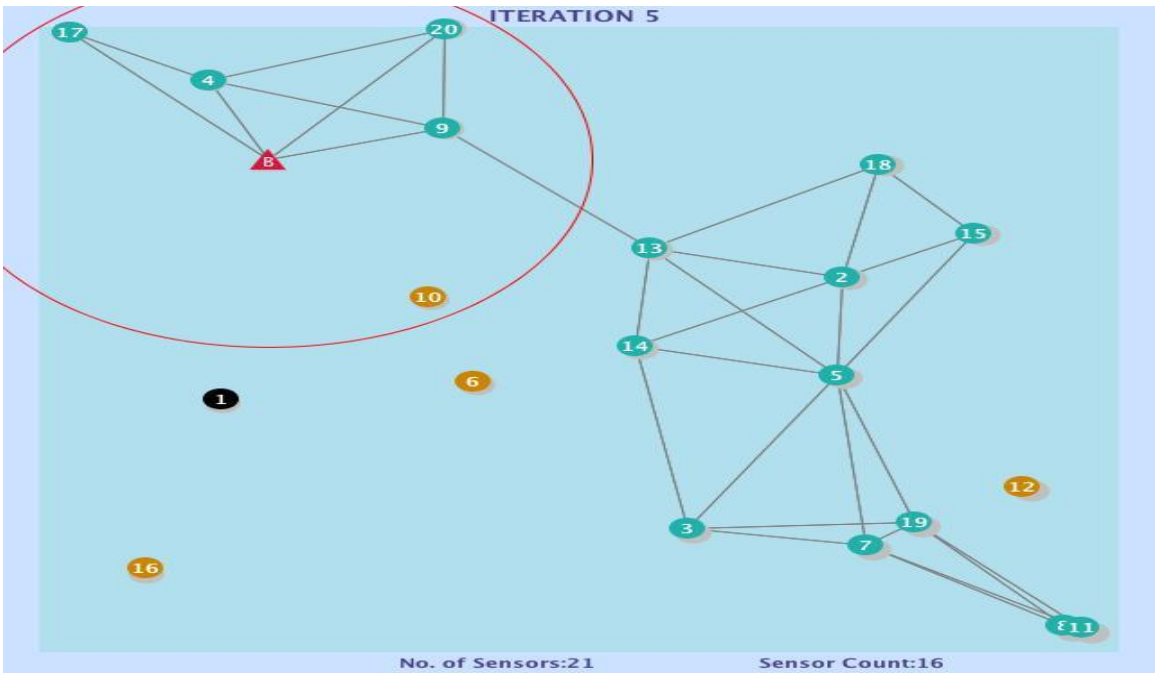

Figure 5.8.4. A Random 21-Sensors Network after Four Sensors Fail.

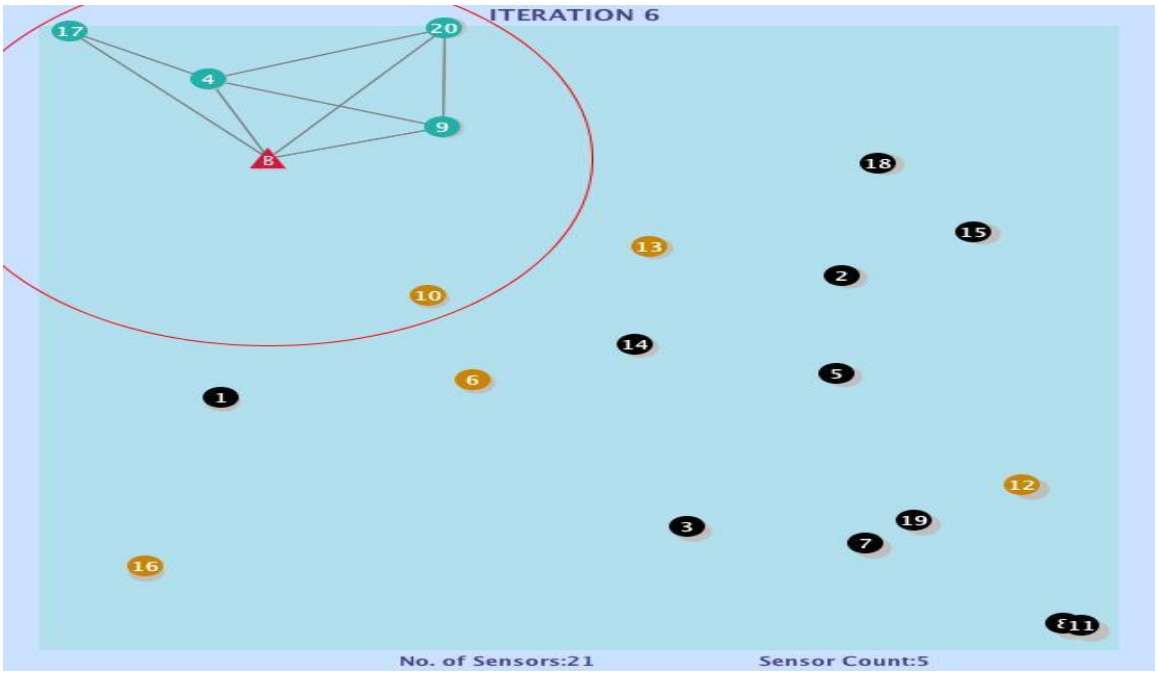

Figure 5.8.5. A Random 21-Sensor Network after Five Sensors Fail.

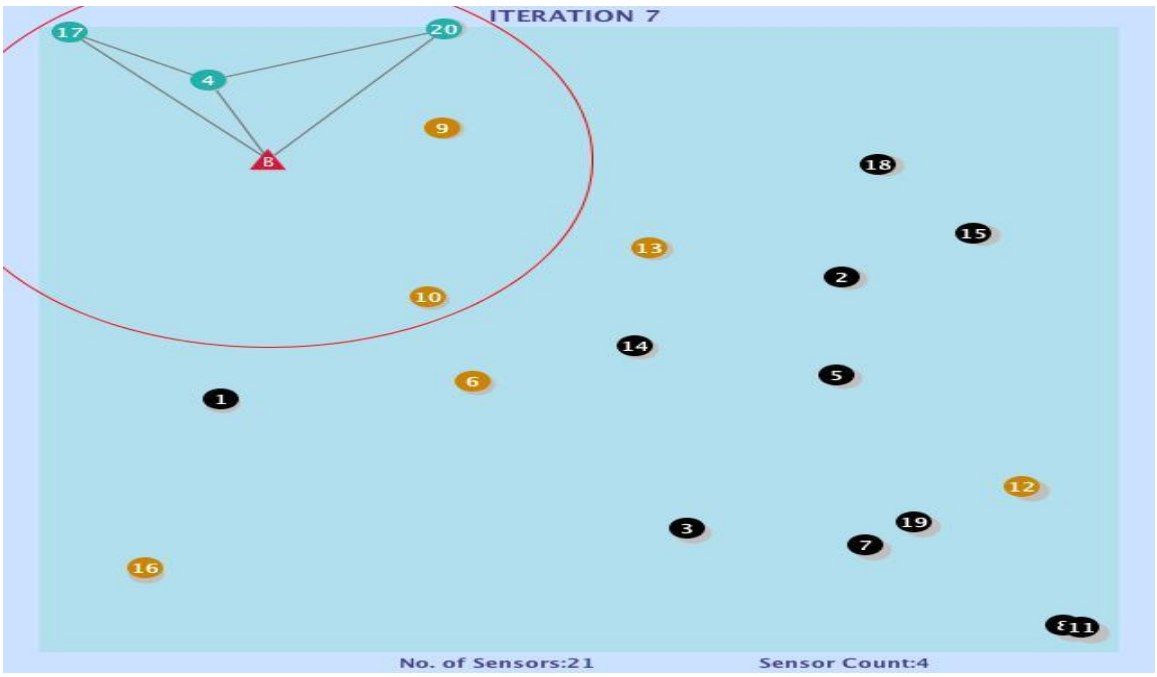

Figure 5.8.6. A Random 21-Sensor Network after Six Sensors Fail.

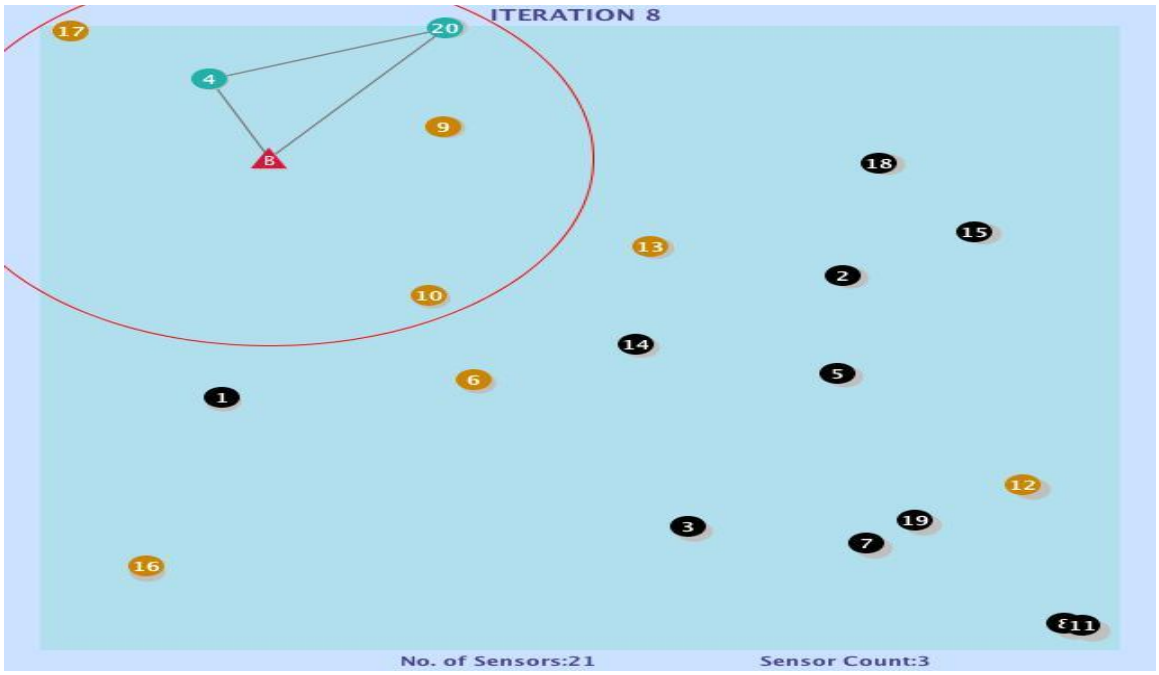

Figure 5.8.7. A Random 21-Sensor Network after Seven Sensors Fail.

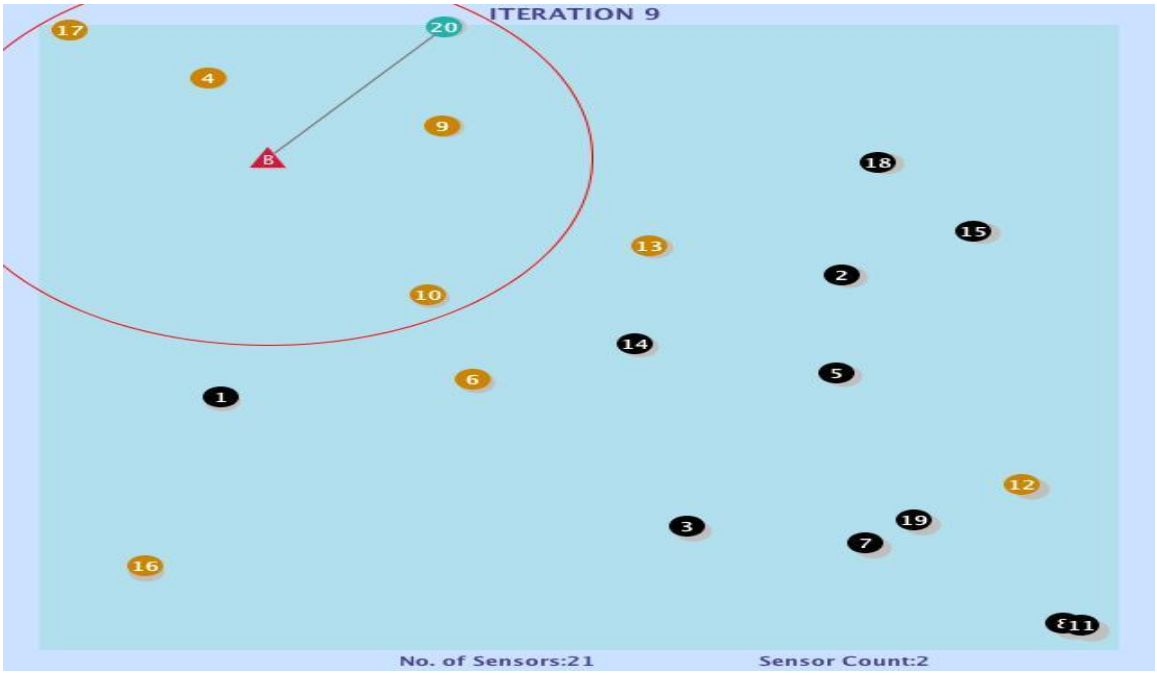

Figure 5.8.8. A Random 21-Sensor Network after Eight Sensors Fail.

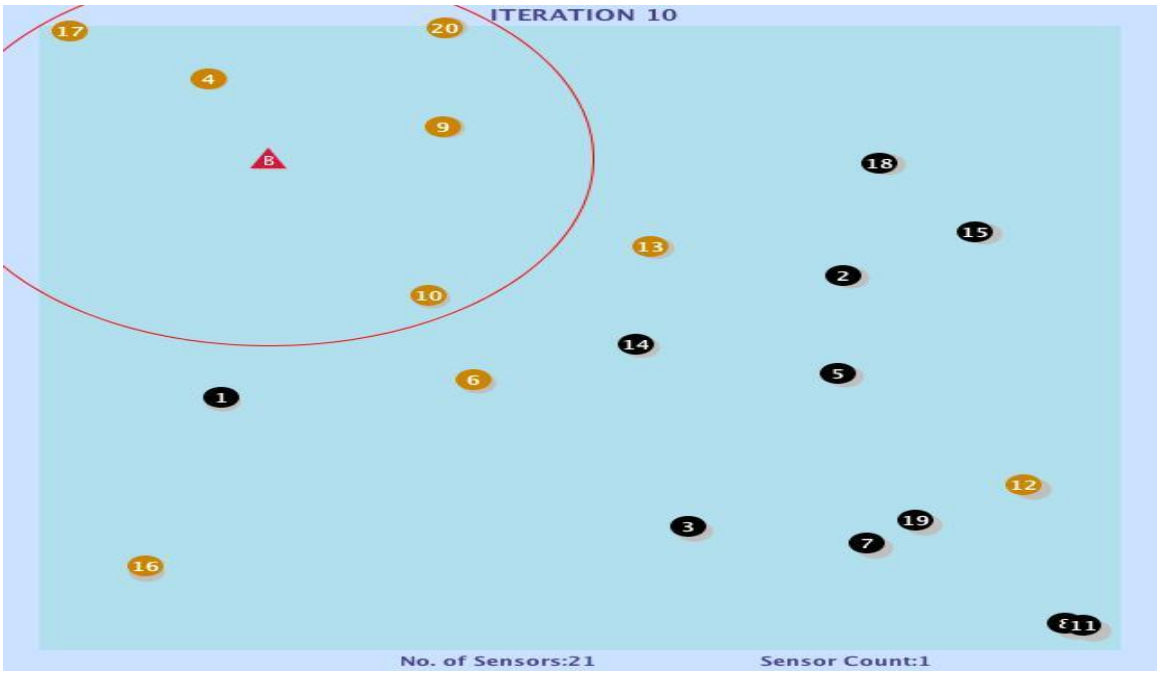

Figure 5.8.9. A Random 21-Sensor Network after Nine Sensors Fail.

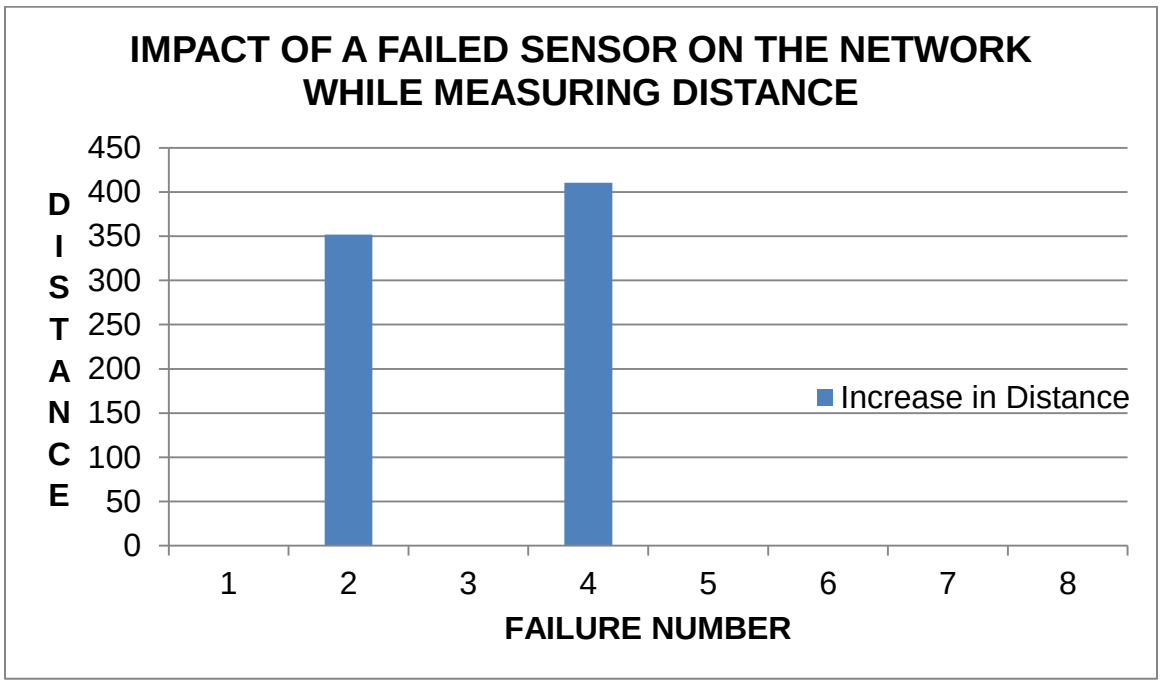

Figure 5.9. Impact of Sensor Failures while Measuring Distance.

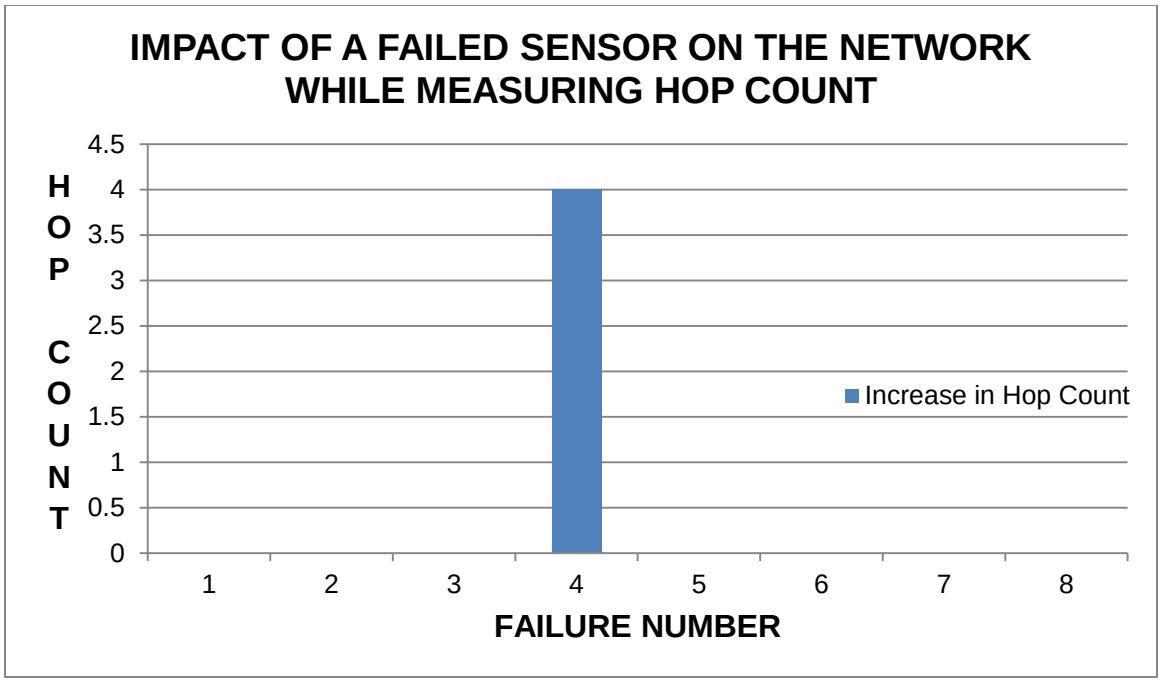

Figure 5.10. Impact of Sensor Failures while Measuring Hop Count.

5.3. Experimental Observations

In this test case we are comparing the results for a network area of 600 x 600 and a network area of 800 x 800 for 190 sensors with a radius of 100. The results are an average of values obtained from 50 different random graphs for each network area. The data table is given in Table 5.3. Figure 5.11 shows a graph with the difference between the total shortest distances after sensor failure for both network areas. The impact of sensor failures on the total shortest distance in a network can be seen in the figure below. When a sensor fails, it might affect the connectivity of other sensors to the base station. The difference might be more in the 800 x 800 network area when compared to the 600 x 600 network area because of the difference in the area. We can see vast changes in the difference as sensors fail in the 800 x 800 area. We can relate it to the network creation probability. The probability of creating a network is higher when there are more sensors in the network area. The change in the total shortest distance will be less when

there are more sensors in the network. The difference between the average total shortest distances is more when a sensor failure affects the connectivity of other sensors in the network. It is the same with the difference between the average total hop counts after each failure. Figure 5.12 shows a graph with the difference between total hop counts after sensor failure for both network areas. Figure 5.13 shows the average number of sensors that can communicate with the base station after each failure in a network. As we can see, the change in hop counts is also more when we compare both the graph areas. When we compare the number of sensors alive after each failure in both the network areas, there will be fewer sensors in the 800 x 800 area when compared to the 600 x 600 area after 20 sensors fail in the network. The number of sensors in the network will have an impact on the runtime. If we see the change of total shortest distance at failures in both the network areas, the 600 x 600 area has more change at some failures after 75 failures when compared to the 800 x 800 area. If we see the change in total hop count, it is similar to the change in shortest distance.

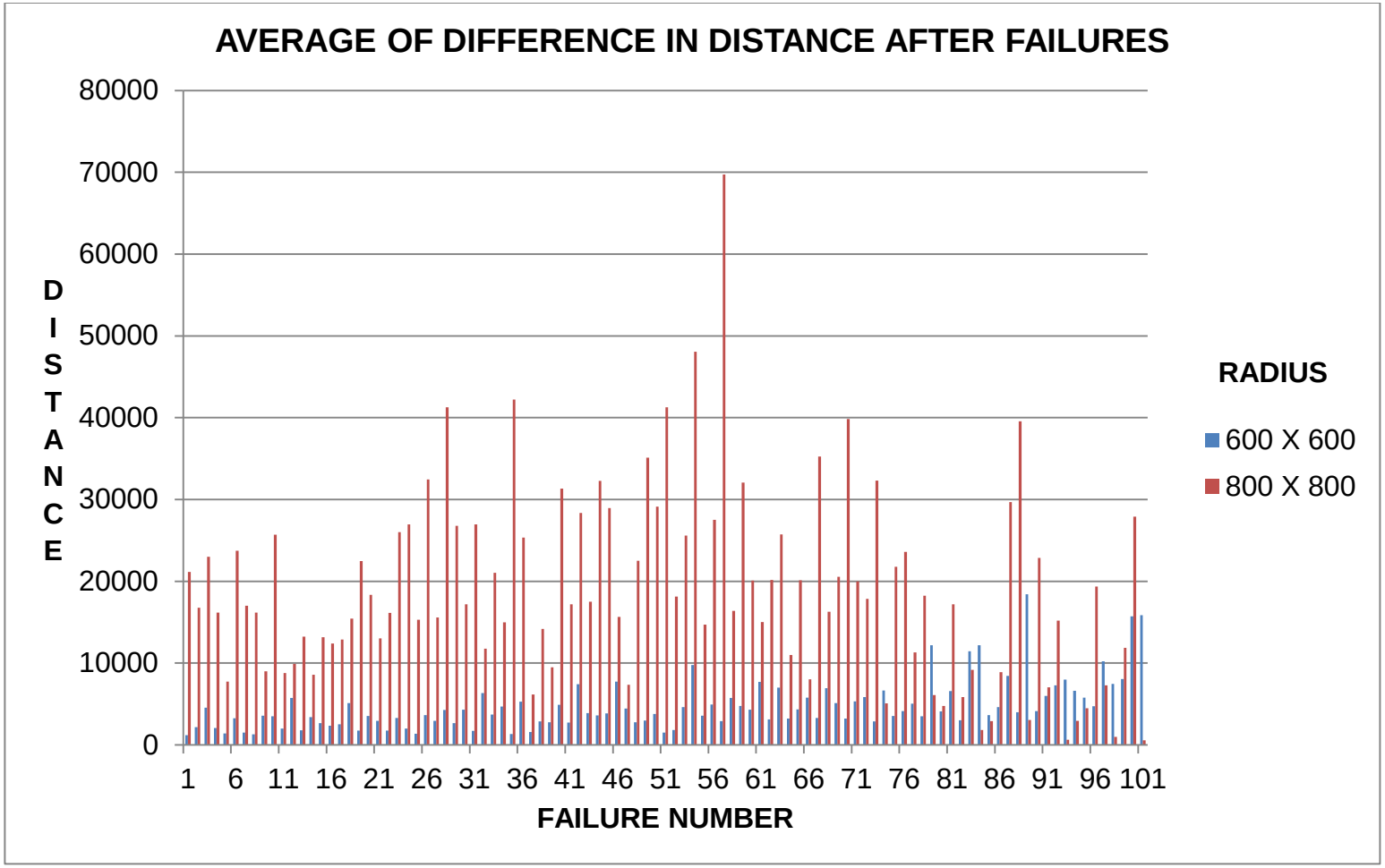

Figure 5.11. Changes in Shortest Distance after Sensor Failures.

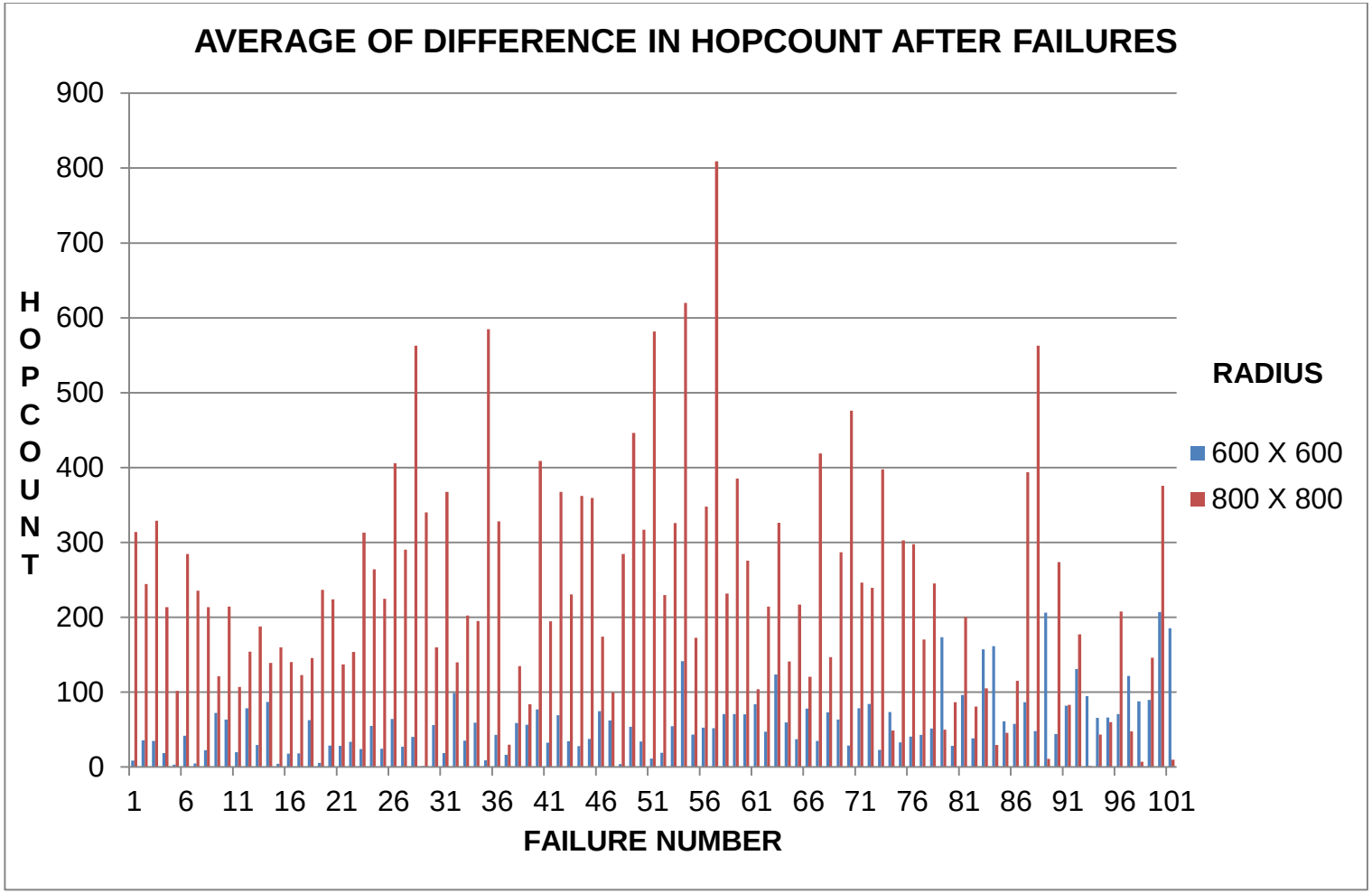

Figure 5.12. Changes in Hop Count after Sensor Failures.

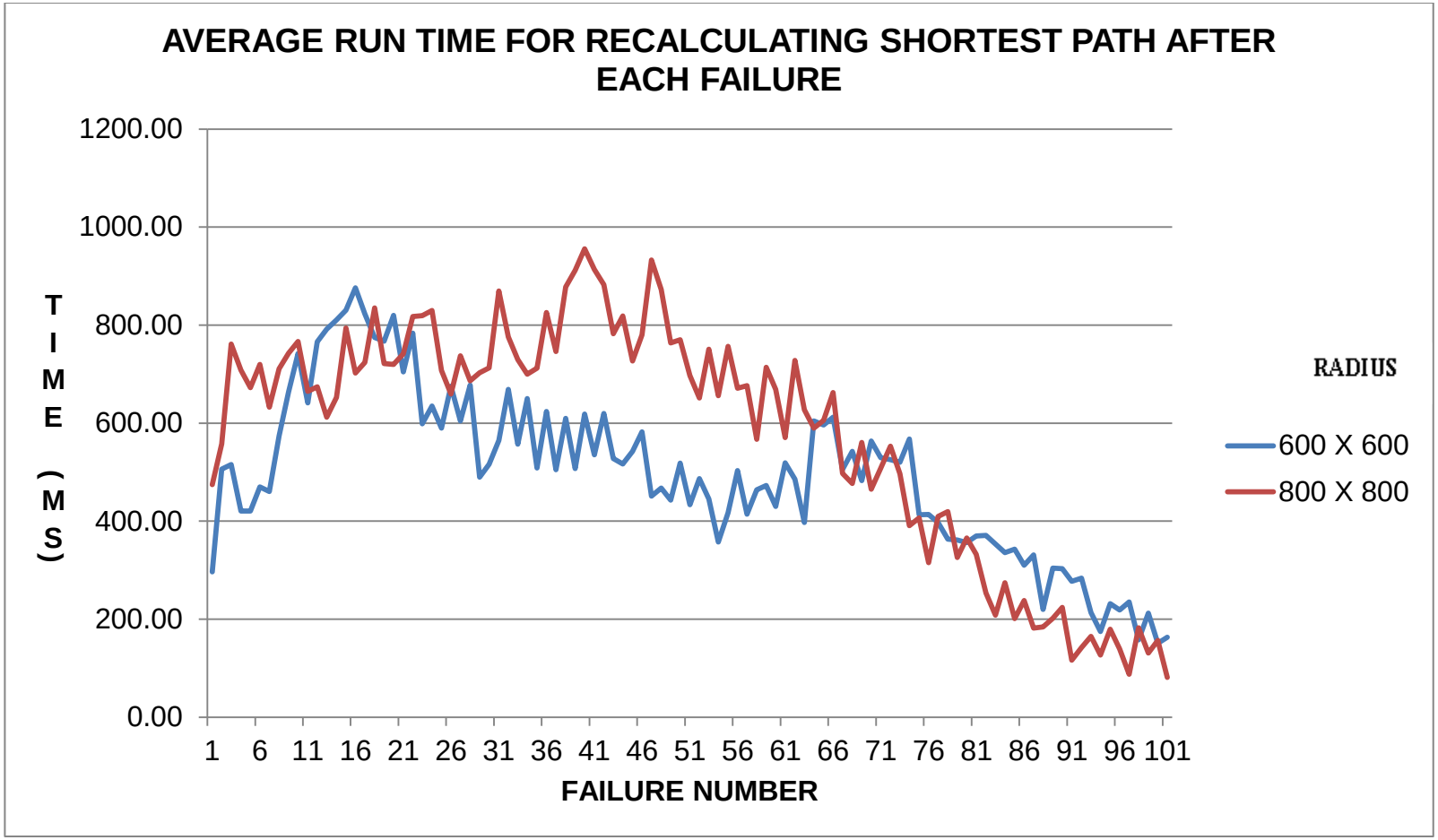

Figure 5.13. Average Run Time for Each Failure.

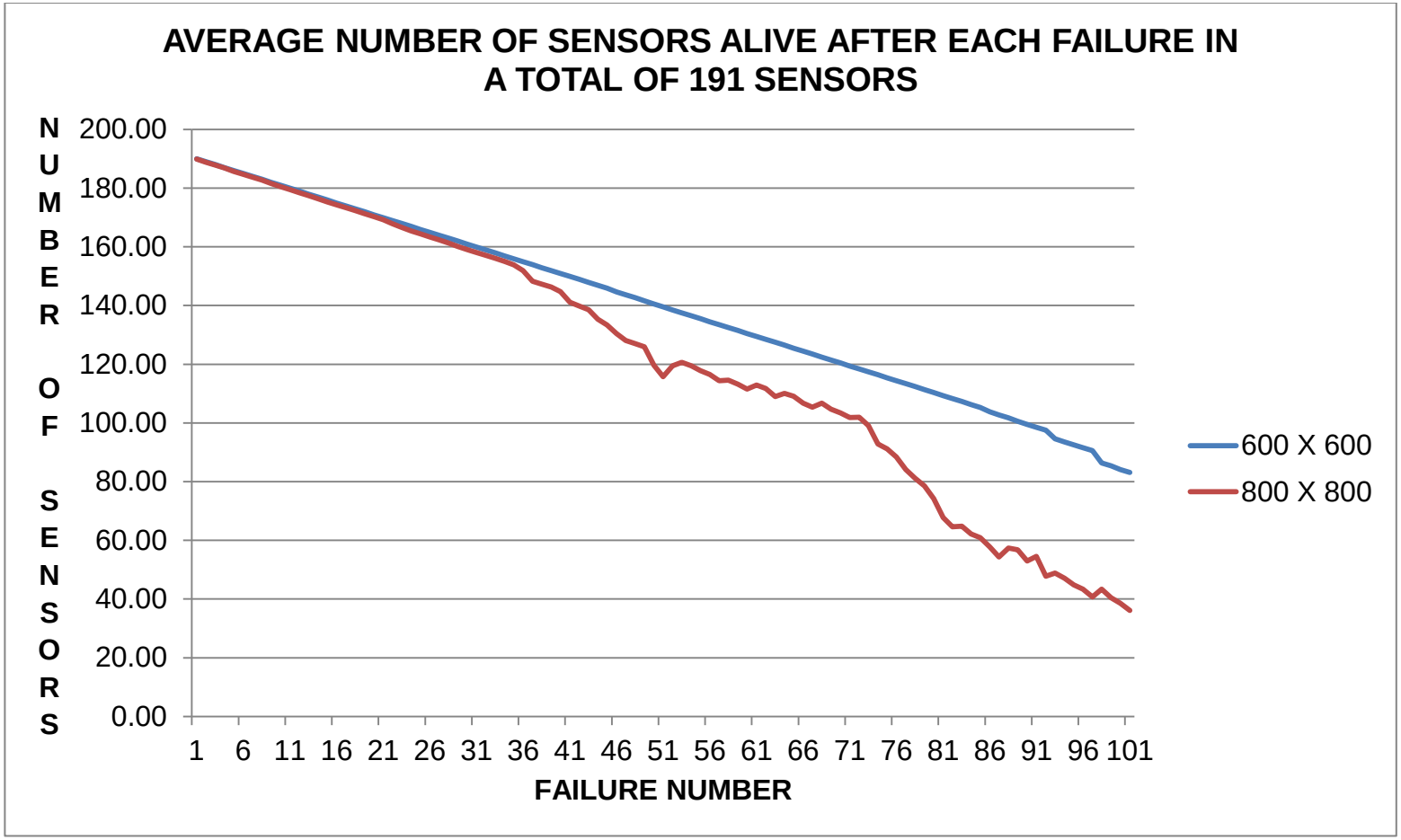

Figure 5.14. Number of Sensors Reached by the Base Station after Sensors Fail.

Table 5.3. Summarized Experimental Results for 190 Sensors with a Radius of 100.

	Difference in a 600 x 600 Area				Difference in a 800 x 800 Area			
Failure	Distance	HopCount	Instance Time	SensorCount	Distance	HopCount	Instance Time	SensorCount
1	1183.99	8.4	296.84	190	21136.82	313.66	474.88	189.88
2	2165.35	35.54	506.32	189	16766.68	244.28	557.68	188.86
3	4557.60	34.74	515.36	188	22987.26	328.9	761.26	187.84
4	2056.21	18.34	420.54	187	16171.14	213.54	708.44	186.84
5	1386.56	3.06	420.66	186	7737.70	101.42	672.18	185.72
6	3253.25	41.48	470.06	185	23737.21	284.64	719.94	184.72
7	1490.76	4.46	460.66	184	17006.32	235.34	632.8	183.72
8	1293.19	22.52	574.06	183	16184.10	213.46	711.22	182.72
9	3553.82	72.16	663.7	182	8985.87	121.26	742.52	181.52
10	3514.68	63.2	741.94	181	25688.05	214.04	766.5	180.52
11	1981.78	19.78	641.7	180	8770.82	106.72	665.06	179.52
12	5731.61	78.48	765.9	179	9913.15	154.04	674.36	178.46
13	1767.34	29.4	791.98	177.96	13235.85	187.76	612.16	177.42
14	3405.65	86.96	810.88	176.94	8588.24	138.8	652.8	176.4
15	2661.40	4.12	830.32	175.94	13160.37	159.82	794.4	175.34
16	2354.15	17.56	875.92	174.94	12402.54	139.96	702.46	174.32
17	2518.03	18.22	822.84	173.94	12881.39	122.82	724.08	173.32
18	5100.52	62.58	775.02	172.94	15440.54	145.52	834.98	172.32
19	1763.65	5.36	767.1	171.94	22457.41	236.74	721.84	171.26
20	3548.25	28.7	820.08	170.94	18335.91	224.02	719.7	170.26
21	2933.14	28.3	704.62	169.94	13024.35	137	741.26	169.24
22	1756.34	33.46	783.42	168.94	16120.47	153.42	817.86	167.82
23	3294.95	24	598.76	167.94	26008.06	313.04	819.34	166.6

(Continued)

Table 5.3. Summarized Experimental Results for 190 Sensors with a Radius of 100. (Continued)

	Difference in a 600 x 600 Area				Difference in a 800 x 800 Area			
Failure	Distance	HopCount	Instance Time	SensorCount	Distance	HopCount	Instance Time	SensorCount
24	1988.90	54.94	635.26	166.94	26945.86	264.14	829.7	165.4
25	1377.89	24.32	589.76	165.94	15294.27	224.68	707.52	164.38
26	3633.57	64.12	675.66	164.94	32434.86	405.52	659.4	163.38
27	2937.83	27.06	604.64	163.94	15569.70	290.26	737.54	162.34
28	4265.85	39.96	677.5	162.94	41270.63	562.8	686.32	161.32
29	2676.77	1.02	489.96	161.94	26770.92	339.96	703.12	160.08
30	4305.04	55.88	516.18	160.94	17199.35	159.64	712.58	159.08
31	1719.91	18.48	564.98	159.94	26944.15	367.58	870.02	158.06
32	6336.42	98.88	669.1	158.94	11765.81	139.7	775.98	157.06
33	3692.61	35.28	557.16	157.92	21028.44	202.24	729.7	156.06
34	4693.08	58.92	649.92	156.92	14994.49	194.88	699.98	155.04
35	1334.84	8.7	508.4	155.92	42226.01	584.68	712.5	153.82
36	5299.08	42.76	623.74	154.92	25346.49	328.08	825.92	151.9
37	1563.43	16.36	505.02	153.92	6155.62	29.56	745.96	148.3
38	2865.16	58.84	610.02	152.92	14174.03	134.78	877.78	147.3
39	2748.28	56.42	507.44	151.92	9475.15	83.7	911.88	146.3
40	4896.59	76.98	618.44	150.92	31314.98	408.88	955.58	144.72
41	2725.67	32.28	535.28	149.92	17185.43	194.44	913.5	141.1
42	7416.92	69.1	619.7	148.92	28347.39	367.64	882.46	139.8
43	3874.33	34.24	527.74	147.92	17484.95	230.42	782.24	138.58
44	3613.92	27.84	516.92	146.92	32285.36	362.06	819.04	135.3
45	3861.43	37.28	542.54	145.92	28952.07	359.48	727.18	133.42
46	7718.62	74.66	582.2	144.64	15642.58	173.96	780.38	130.44
47	4458.77	62.22	451.26	143.64	7363.92	99.5	933.06	128.14
48	2752.19	3.8	467.3	142.64	22509.22	284.62	872.5	127.06

(Continued)

Table 5.3. Summarized Experimental Results for 190 Sensors with a Radius of 100. (Continued)

	Difference in a 600 x 600 Area				Difference in a 800 x 800 Area			
Failure	Distance	HopCount	Instance Time	SensorCount	Distance	HopCount	Instance Time	SensorCount
49	2982.19	53.72	443.12	141.64	35102.05	446.18	763.5	125.92
50	3773.95	33.84	518.4	140.52	29136.99	316.9	770.28	119.64
51	1521.91	11.06	433.42	139.52	41254.39	581.6	696.84	115.76
52	1806.78	18.7	486.96	138.52	18137.98	229.75	651.375	119.5
53	4635.02	54.26	445.32	137.52	25605.36	325.808511	750.6809	120.64
54	9754.76	141.12	357.56	136.52	48061.63	619.851064	656.2766	119.45
55	3555.14	43.06	417.4	135.5	14709.90	172.574468	756.9149	117.81
56	4943.93	52.4	503.54	134.5	27510.60	347.914894	671.2128	116.57
57	2899.07	51.74	414.04	133.5	69743.13	809.085106	676.617	114.34
58	5730.86	70.6	463.78	132.5	16367.47	231.478261	566.8696	114.54
59	4767.58	70.46	472.76	131.5	32070.12	385.282609	714.2826	113.24
60	4299.29	70.26	430.3	130.48	20087.78	275.586957	668.8696	111.5
61	7692.82	83.88	519.14	129.48	15014.58	103.933333	570.8	112.87
62	3101.62	46.88	485.98	128.48	20178.33	214.25	728.0227	111.77
63	7015.04	123.52	397.16	127.48	25723.60	326.045455	627.5227	109
64	3234.90	59.38	604.76	126.48	10995.55	140.790698	590.0233	110.12
65	4353.49	36.98	596.46	125.48	20123.03	216.813953	605.7674	109.09
66	5784.42	77.86	612.3	124.44	8028.00	120.302326	662.4651	106.74
67	3272.55	34.74	505.88	123.44	35243.40	418.976744	497.4186	105.35
68	6922.46	72.88	542.6	122.44	16283.33	146.738095	476.9048	106.79
69	5113.81	63.22	482.54	121.44	20539.16	286.904762	560.9286	104.67
70	3204.57	28.64	563.74	120.42	39827.29	475.904762	465.0714	103.38
71	5308.13	78.2	529.62	119.4	19975.78	246.261905	507.8333	101.81
72	5828.32	84.08	525.68	118.4	17858.09	239.390244	553.2927	101.98
73	2855.79	22.9	520.38	117.4	32312.23	397.512195	496.8537	99.098

(Continued)

Table 5.3. Summarized Experimental Results for 190 Sensors with a Radius of 100. (Continued)

	Difference in a 600 x 600 Area				Difference in a 800 x 800 Area			
Failure	Distance	HopCount	Instance Time	SensorCount	Distance	HopCount	Instance Time	SensorCount
74	6645.00	73.4	567.44	116.4	5083.87	48.4390244	391.0244	92.854
75	3528.28	32.96	413.08	115.4	21781.39	302.536585	406.6585	91.195
76	4127.37	40.42	413.7	114.4	23609.81	297.536585	315.561	88.439
77	5033.47	42.68	397.22	113.4	11322.74	170.05	409.825	84.125
78	3512.88	51.46	363.52	112.4	18233.47	244.974359	419.4615	81.205
79	12183.23	173.16	361.62	111.36	6092.86	49.7179487	325.8205	78.538
80	4086.49	28.22	356.16	110.36	4765.08	86.3846154	365.9744	74.231
81	6597.99	96.18	369.72	109.32	17188.75	200.487179	332.6154	67.769
82	3018.64	38.08	370.88	108.32	5833.77	80.5641026	253.5641	64.615
83	11433.02	157.22	353.42	107.28	9184.73	104.837838	208.1351	64.838
84	12198.21	161.48	335.66	106.28	1825.85	29.2972973	274.4595	62.135
85	3647.15	60.9	343.16	105.28	2903.52	45.6111111	200.9722	60.861
86	4637.34	57.42	309.96	103.76	8905.07	115.138889	238.3056	57.778
87	8419.46	86.62	330.9	102.72	29681.16	393.617647	182	54.294
88	3994.62	48	219.68	101.72	39547.74	562.870968	184	57.419
89	18428.75	206.02	304.04	100.54	3037.69	10.9	201.6	56.767
90	4119.85	44.16	303.2	99.54	22851.03	273.8	223.9667	52.933
91	5976.67	82	277.54	98.54	7038.39	83.0357143	116.6071	54.5
92	7267.75	131.02	283.78	97.54	15192.44	177.071429	142.3571	47.821
93	7995.21	94.46	213.7	94.58	635.26	1	165.1538	48.808
94	6620.37	65.72	175.02	93.54	2925.12	43.32	127	47.12
95	5773.10	65.9	231.9	92.54	4494.94	59.92	179.28	44.84
96	4728.15	70.5	218.7	91.54	19369.04	207.52	139.4	43.4
97	10236.88	121.72	234.94	90.54	7286.14	47.6	87.56	40.72
98	7458.57	87.68	158.16	86.4	989.25	7.09090909	182.4545	43.409

(Continued)

Table 5.3. Summarized Experimental Results for 190 Sensors with a Radius of 100. (Continued)

	Difference in a 600 x 600 Area				Difference in a 800 x 800 Area			
Failure	Distance	HopCount	Instance Time	SensorCount	Distance	HopCount	Instance Time	SensorCount
99	8038.79	89.46	212.08	85.36	11851.38	145.909091	130.6818	40.455
100	15708.22	207	151	84.16	27894.93	375.428571	156.9048	38.571
101	15862.33	185.36	163.36	83.14	553.90	9.57142857	81.09524	36.095

6. CONCLUSION

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

1. Developed an application to explore the algorithm to find the shortest path between all the sensors in a wireless sensor network after sensor failures.
2. Established that a shortest path can be found by not running the shortest path algorithms on the whole network after the failure.
3. Developed a network based on the transmission distance of a sensor.
4. The runtime of the algorithm is calculated for each independent failure.
5. The shortest path distance increases as the number of failed sensors increases.
6. We can plan how many sensors to deploy in a network area by running a simulation, and we can detect how many sensors are required to make sure that the impact of sensor failures won't affect other sensors predicting the number of failures to occur.

This application can be run on a network area with multiple clusters of sensors or with sensors with a different communication radius and obtain the shortest path. This algorithm obtains a shortest path based on the distance, whereas we can apply hop count as the criteria to try to obtain the results.

7. REFERENCES

[1] D. J. Cool, S. k. Das and F. L. Lewis, Smart Environments: Technologies, Protocols, and Applications, Wiley, 2005.

[2] X. Du, M. Zhang and N. E. Kendall, "Self-healing sensor networks with distributed decision making," *International Jounal Sensor Netowrks,* vol. 1, no. 1/2/3, 2006.

[3] I. F. Akyildiz, T. Melodia and K. R. Chowdhury, "A Survey on Wireless Multimedia Sensor Networks," *Computer Networks,* vol. 51, no. 4, pp. 921-960, 2007.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "Wireless sensor netowrks: a survey," *Computer Networks,* vol. 38, no. 4, pp. 393-422, 2002.

[5] M. Marina and S. Das, "On-demand Multipath Distance Vector Routing in Ad Hoc Networks," in *Network Protocols Ninth International Conference on ICNP 2001,* 2001.

[6] C. E. Perkins, "Ad-hoc on-demand distance vector routing," in *Mobile Computing Systems and Applications,* 1999.

[7] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes and D. Wagner, "Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm," *Journal of Experimental Algorithmics,* vol. 15, 2010.

[8] J. B. Orlin, K. Madduri, K. Subramani and K. Williamson, "A faster algorithm for the single source shortest path problem with few distinct positive lengths," *Journal of Discrete*

Algorithms, vol. 8, no. 2, p. 189–198, 2010.

[9] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM,* vol. 34, no. 3, 1987.

[10] M. L. Fredman and D. E. Willard, "Trans-dichotomous algorithms for minimum spanning trees and shortest paths," *Journal of Computer and System Sciences,* vol. 48, no. 3, p. 533–551, 1994.

[11] R. Mohring, H. Schilling, B. Schutz, D. Wagner and T. Willhlam, "Partitioning Graphs to Speedup Dijkstra's Algorthm," *Journal of Experimental Algorithmics,* vol. 11, 2006.

[12] R. W. Floyd, "Shortest Path," *Communication of ACM,* p. 345, 1960.