Design and Implementation of Alert Detection In A Distributed Wireless Sensor Network (WSN)

Hew Ye Zea
Mohamed Hijan Hameez
School of Information Technology / School of Engineering
Monash University Malaysia
yhew0003@student.monash.edu
mham0009@student.monash.edu

***Abstract* – Wireless Sensor Networks (WSN) are responsible for monitoring and collecting data to a centralized database. In this paper we propose an efficient implementation of sensor nodes in a 2D grid using a cartesian topology and inter process communication (IPC). Message Passing Interface (MPI) is used for communication between nodes and the base station. The end goal of the system is to simulate a Tsunami detection system, by analysing average water height levels and reporting anomalous high readings to a base station for human analysis. In addition, threads will be used for simulating the data generated by the satellite altimeter.**

***Index Terms* – *Wireless Sensor Network,* WSN, cartesian topology, MPI, Message Passing Interface, IPC**

I. INTRODUCTION

Tsunamis such as the devastating boxing day tsunami of 2004, originating from a magnitude 9.1 earthquake off the coast of Sumatra Indonesia, have caused irreversible damage to the lives of many people.While there is nothing we can do to prevent a tsunami, the effects can be mitigated.Up until recent years they have been looked at as a force of nature that is difficult to predict, unlike hurricanes and other more visible phenomena. Now however there are systems that can help with early detection of such tsunamis, and our report is on one such system that uses a Wireless Sensor Network (WSN) to detect anomalous water height readings, subsequently alerting authorities of possible tsunami threats before they reach coastlines across the world.

A WSN can gather and process data at a much faster rate than human interpreters, all at a lower maintenance cost as a small team could manage a significant number of sensors. They are also not prone to human error, where forward observers such as coast guard authorities may miss key signs of a tsunami. Their operation is not restricted by harsh weather conditions or human sleep cycles, and with adequate maintenance they can maintain near round the clock vigilance. In this paper we will be looking at the use of a WSN for monitoring ocean height levels, alongside a simulated satellite altimeter and base station, with the goal of identifying possible tsunami threats before they reach coastlines and showing that the threat of tsunamis can be significantly averted through the use of WSN networks.

Our WSN utilises an Inter Process Communication architecture, where data is exchanged between MPI processes layered in a 2D Cartesian grid topology. Sensor nodes will rely on readings from their neighbours when prompted,

Our proposed hypothesis is that higher tolerances will lead to a higher true alert ratio. We will investigate this by running multiple simulations on varying parameters.

II. Sensor Network Design

A. Main Function

A WSN alert detection system has 2 main components, namely the base station which communicates with the satellite altimeter and sensor nodes. Our system takes in 4 user inputs, m,n, iteration and threshold. This allows dynamic grid arrangement, as the user can freely specify the corresponding row and column (m*n). Threshold represents the value when exceeded, represents a potential event.

Pseudocode 1 – Main Function

```
1 //Request for user input
2 int n,m,iteration,threshold <-
request_user_input()

3 //Broadcast to all process
4 MPI_Bcast(&m_val)
5 MPI_Bcast(&n_val)
6 MPI_Bcast(&iteration)
7 MPI_Bcast(&threshold_val)

8 // Create global MPI communicator
9 // Initialise MPI
10 MPI_Init(&argc, &argv)
11 MPI_Comm_size(MPI_COMM_WORLD,
&comm_size)
12 MPI_Comm_rank(MPI_COMM_WORLD,
&world_rank)
13 //Master slave splitting, split process into two
groups
14 MPI_Comm_Split(MPI_COMM_WORLD,
world_rank == 0, 0, &sub_comm)

15 //1. Base station
16 //2. WSN
17 if(world_rank != 0)
18    // number of alerts sent per
      node
19    wsn_sensor()
20 else
21    // For base station
22    base_station()
```

Then the values entered are broadcasted to all processes. The global MPI Communicator is then created, the base station and sensor nodes are represented as a node of the world communicator.

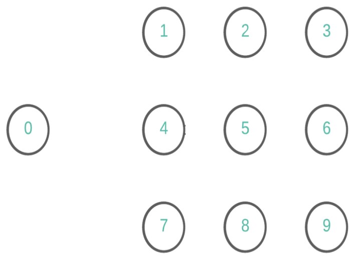


Figure 1 : Nodes of the world_ comm

MPI_Comm_split is used to divide between the base station and the sensor nodes.This splits the processes into two groups, with the master group containing the base station and the slave group containing the sensor nodes. The rank of 0 is used for the base station and all subsequent ranks are for sensor nodes.

B. Sensor Nodes

The sensor nodes are represented in a 2D Cartesian grid-based architecture. Each node can only communicate with its immediate adjacent nodes (i.e. top, bottom, left and right nodes) and the base station only. Pseudocode 2 shows the algorithm to achieve this.

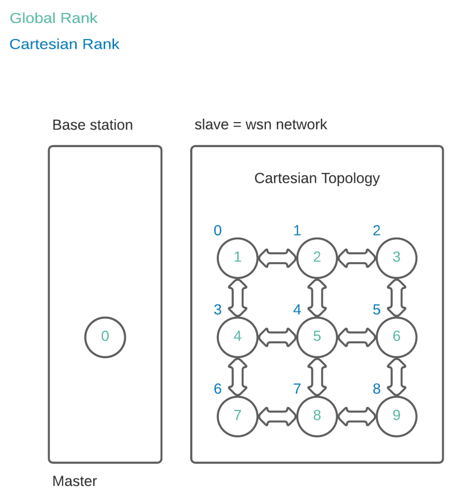


Figure 2 : Nodes after splitting and cartesian creation

Pseudocode 2 – Cartesian Creation

```
1 // Create 2D cartesian topology
2 MPI_Cart_create()
3 int ndims=2, size, sub_rank, reorder, my_cart_rank,
ierr
4 MPI_Comm comm2D
5 wrap_around[0] = 0
6 wrap_aroundp[1] = 0
7 reorder =1

8 // Create cartesian topology
9 MPI_Comm_size(MPI_COMM_WORLD,
&comm_size)
10 ierr = MPI_Cart_create(comm, ndims, dims,
wrap_around, reorder, &comm2D)

11 // Find coordinates in the 2D cartesian
communicator group
12 MPI_Cart_coords(comm2D, sub_rank, ndims,
coord)

13 // use my 2D cartesian coordinates to find my rank
in cartesian groups
14 MPI_Cart_rank(comm2D, coord, &my_cart_rank)

15 // Neighbour node detection
16 MPI_Cart_shift(comm2D, SHIFT_ROW, DISP,
&neighbours[2], &neighbours[3])
17 MPI_Cart_shift(comm2D, SHIFT_ROW, DISP,
&neighbours[0], &neighbours[1])
```

MPI_Cart_create initialised the cartesian topology. **MPI_Cart_coords** is used to the coordinates of each node, while **MPI_cart_shift** is used to identify the adjacent neighbours' rank. Figure 3 represents a 2D cartesian network of 3x7.

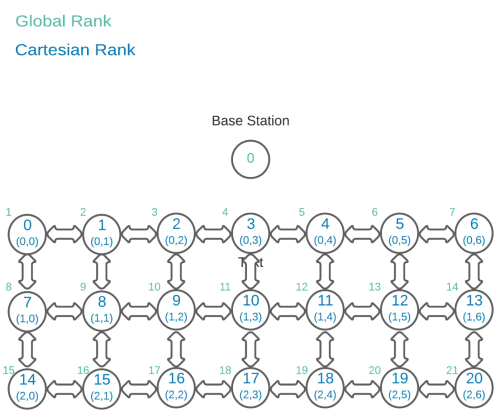


Figure 3 : 3x7 Cartesian Topology

C. Sensor node - Event detection criterion

Pseudocode 3 – WSN network

```
1 // Generate Random Values
2 reading = (random() % (MAX_HEIGHT -
MIN_HEIGHT + 1)) + MIN_HEIGHT

3 // compute moving average
4 update_moving_average(reading)

5 // If moving average exceeds predefined threshold,
there's a possible event
5 if (moving_average > threshold_val)
6   // Send a request to neighbours
7   if(neighbours[i] != MPI_PROC_NULL)
8     MPI_Isend(&moving_average)
9     MPI_Irecv(&neighbour_values[i])
10     message_count++

11 if (matched_neighbours() >= 2)
12   int neigh_coord[ndims]

13 //Send a report to base station
14 MPI_Request req
15 MPI_Status sta
16 MPI_Isend(&report)
17 MPI_Wait(&req, &sta)
18 message_count = 0
```

A sensor node generates a random reading between the predefined **MIN_HEIGHT** and **MAX_HEIGHT** values. Then the sensor note updates the moving average value by using the most recent x values within a window of size **MOVING_AVERAGE_WINDOW**. This means that only the most recent values are used when calculating the moving average.

Should the updated moving average value exceed the user defined threshold, the node will send a request to acquire information from its neighbours on their own moving average values. Upon receiving this information the node will then check to see whether any of the received values correlate with their own within a certain threshold value. The node sends an alert report to the base station if it's moving average is matched with over its 2 neighbours. Figure X represents the technical flowchart of a sensor node.

D. Base Station

The flowchart of a base station is included in figure x. The base station node creates a global array to store reading generated by the satellite altimeter. This global array uses a FIFO approach. Prior to starting the iterations, the base station uses **pthread_create** to fork a new thread to simulate the satellite altimeter. The algorithm for satellite altimeter is described as pseudocode 5.

In each iteration, the base station listens for incoming reports from the sensor nodes and obtains satellite readings from the satellite altimeter periodically. If an alert is received, the base station compares the moving average value contained within the report with the latest satellite readings. If the difference between the reports reading and the satellites reading is less than the given **TOLERANCE**, a true alert is logged. If not, a false alert is logged. The details sent from the sensor node, the reading generated by satellite altimeter and the total number of true and false alerts are also logged.

These iterations repeat for N number of times, where N is provided by the user at run time. When the loop completes, a summary of results is logged and calls **pthread_join** specified by **tid** to terminate.

Pseudocode 4 – Base station

```
1 // Initialise Global Array
2   queue = createFIFO(CAPACITY);

3 // create thread
4   pthread_create(&tid);

5 //  begin iterations
6     for (int iter = 0; iter < iterations ; iter ++ ) {

          // receive alert from sensor nodes

7           MPI_Irecv(&report, 1, struct_type,
MPI_ANY_SOURCE, SEND_TO_ROOT_TAG,
master_comm, &alert_request);

        MPI_Test(&alert_request, &alert_received, &alert_status);

        if abs(reading_difference) < TOLERANCE{
              true_alerts++;
              LogOutput(1,output_file,report);
         }else {
               false_alerts++;
              logOutput(0,output_file,report);
    }
}

// log summary
fprintf(output_file, “Summary”)
fprintf(output_file, “Total number of alerts:”, total_alerts_count)
fprintf(output_file, “True alerts:”, true_alers)
fprintf(output_file, “Correct rate:”, (float)true_alerts/total_alerts_count)
Termination = 1
fclose(output_file)

// terminate thread
pthread_join(tid)
free(satellite_readings)

Return total_alerts_count
```

E. Satellite Altimeter Simulation

Pseudocode 5 – Satellite Altimeter Simulation

```
while (! termination ( {
    // generate random readings
      int random_reading = (rand() % (MAX_HEIGHT - MIN_HEIGHT + 1)) + MIN_HEIGHT;

// generate random coordinates within 2D cart
int rand_x = rand() % ( row + 1 );
 int rand_y = rand() % ( col + 1 );

// if global array is not full
      if (!(isFull(satellite_readings))){
          // if queue is not full
          enqueue();
      }
      else{
          // if queue is full
         // remove one item and push new item
            dequeue();
             enqueue();
 }
 sleep(UPDATE_SA);
}
```

The pseudocode above represents the simulation of the satellite altimeter. While the termination signal has not been received, the satellite altimeter will generate random readings and a set of random coordinates associated with said reading. It stores these in a fixed sized shared global array. Should the array become full the first value is removed and the new value placed at the end of the global array. The satellite altimeter will then sleep for a set amount of time before reactivating and generating another set of values. The technical flowchart is shown in figure x.

III. Results and Discussion

A. Simulation Specifications

Simulation of the program is performed on a Ubuntu virtual machine running on Virtualbox on a 2018 MacBook Air. The specifications of the systems and the simulation parameters are detailed below.

MacBook Air

Specification	**Value**	**Description**
CPU	2	Number of CPUs/ cores
RAM	8	Amount of memory

Virtual Machine

Specification	**Value**	**Description**
CPU	2	Number of CPUs/ cores
RAM	3005 MB	Amount of memory
Ubuntu	64bit	Operating system

Fixed simulation parameters

Parameter	**Value**	**Description**
Rows	6	Rows in the grid
Columns	7	Columns in the grid
Number of processes	43	Processes used in the simulation
Threshold	6100	Threshold value is meters
Reading range	5600-7000	Generate random numbers in this range. Value in meters.

In order to test our proposed hypothesis we performed a number of simulations against a fixed set of parameters. These parameters are defined above.

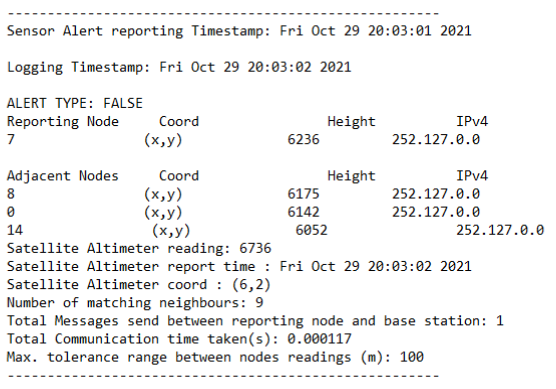
Variable : Iterations

Value	Description
10	Run for 10 iterations
20	Run for 20 iterations
40	Run for 40 iterations
100	Run for 100 iterations

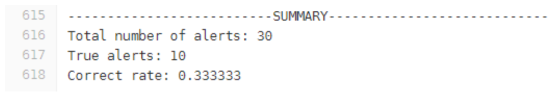
Variable : Tolerance

Value	Description
50	+/- 50 tolerance
100	+/- 100 tolerance
250	+/- 250 tolerance

B. Screenshot log files



```
------------------------------------------------------
Sensor Alert reporting Timestamp: Fri Oct 29 20:03:01 2021

Logging Timestamp: Fri Oct 29 20:03:02 2021

ALERT TYPE: FALSE
Reporting Node     Coord              Height          IPv4
7                 (x,y)             6236         252.127.0.0

Adjacent Nodes     Coord              Height          IPv4
8                 (x,y)             6175         252.127.0.0
0                 (x,y)             6142         252.127.0.0
14                 (x,y)             6052                252.127.0.0
Satellite Altimeter reading: 6736
Satellite Altimeter report time : Fri Oct 29 20:03:02 2021
Satellite Altimeter coord : (6,2)
Number of matching neighbours: 9
Total Messages send between reporting node and base station: 1
Total Communication time taken(s): 0.000117
Max. tolerance range between nodes readings (m): 100
------------------------------------------------------
```



```
-------------------------SUMMARY--------------------------
Total number of alerts: 30
True alerts: 10
Correct rate: 0.333333
```

C. Results

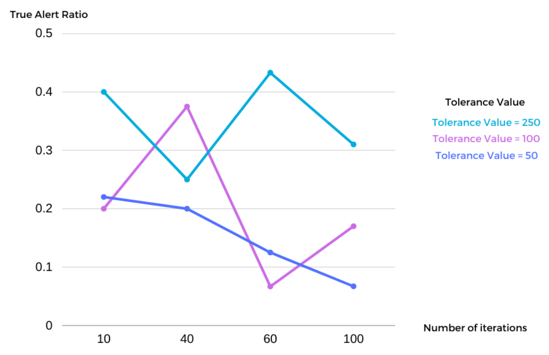


Figure 4 : Relationship between Tolerance and true alert ratio

Iterations	Total Alerts	True alerts	Correct ratio
100	100	17	0.17
10	10	2	0.2
40	40	15	0.375
60	60	30	0.5

Figure x : Tolerance value = 50

Iterations	Total Alerts	True alerts	Correct ratio
100	100	22	0.22
10	10	2	0.2
40	40	5	0.125
60	60	4	0.067

Figure x : Tolerance value = 100

Iterations	Total Alerts	True alerts	Correct ratio
100	100	31	0.31
10	10	4	0.4
40	40	10	0.25
60	60	26	0.433

Figure x : Tolerance value =250

C. Analysis

The true alert ratio represents the ratio of true alerts divided by the total number of iterations. Based on the results, if you look at a certain number of iterations (for example, 10 iterations) and the true alert ratio for 10 iterations at different tolerances, the true alert ratio increases as the tolerance increases.(0.4 at tolerance of 250, while its only 0.2 at a tolerance of 50). This makes sense because the larger the acceptable tolerance, the more values will be accepted and not rejected as more values will fit within the tolerance window above the threshold.

In our opinion, we believe that this significant increase in true alert ratio can be “deceiving”. Since a higher tolerance ratio may include several very low sea column water height values as well, This may lead to “true” alerts which are in fact false, as no tsunami was present. Therefore, this should be tempered to avoid too many false positives.

D. Summary

In this report, we have used an MPI framework based wireless sensor network to set up a series of node based sensors that generate water height readings, communicate with adjacent nodes and communicate with the base station when necessary. From our findings our hypothesis appears to have held true, as the tolerance value plays a crucial part in determining the system's effectiveness in detecting a true alert.

Should the values be too large or too small, the results may include false alerts, potentially compromising the system. Our suggestions from our findings indicate a shorter tolerance works best for accurate warnings.

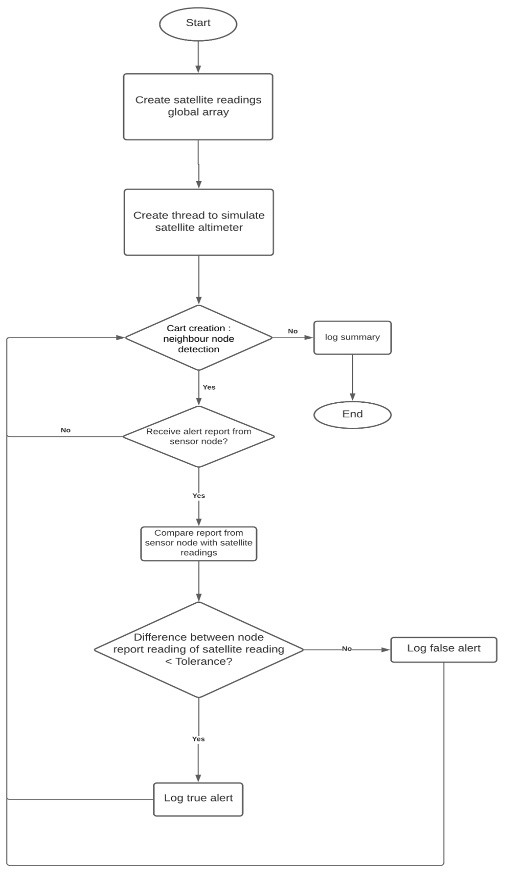

Start
Create satellite readings global array
Create thread to simulate satellite altimeter
Cart creation : neighbour node detection
No
log summary
End
Yes
Receive alert report from sensor node?
No
Yes
Compare report from sensor node with satellite readings
Difference between node report reading of satellite reading < Tolerance?
No
Log false alert
Yes
Log true alert

Figure x : base station flowchart

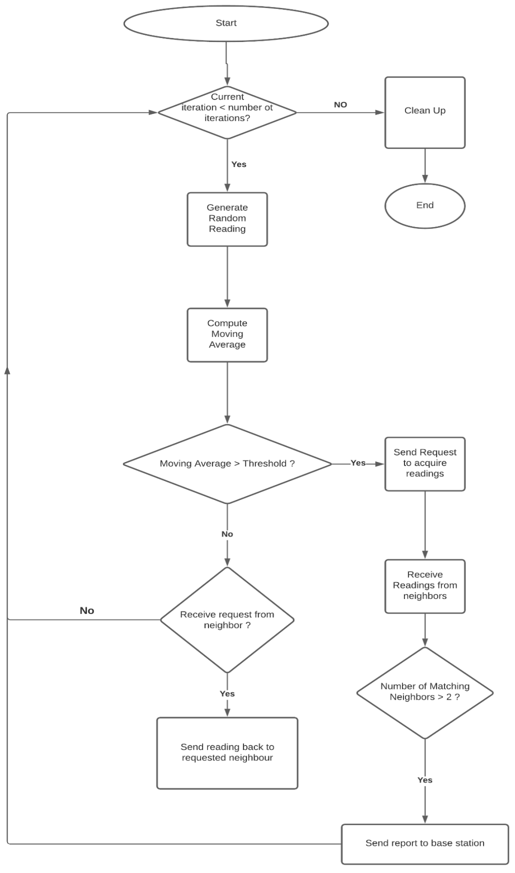

Start
Current iteration < number ot iterations?
NO
Clean Up
End
Yes
Generate Random Reading
Compute Moving Average
Moving Average > Threshold ?
Yes
Send Request to acquire readings
Receive Readings from neighbors
Number of Matching Neighbors > 2 ?
Yes
Send report to base station
No
Receive request from neighbor ?
No
Yes
Send reading back to requested neighbour

Figure x : Sensor node flowchart

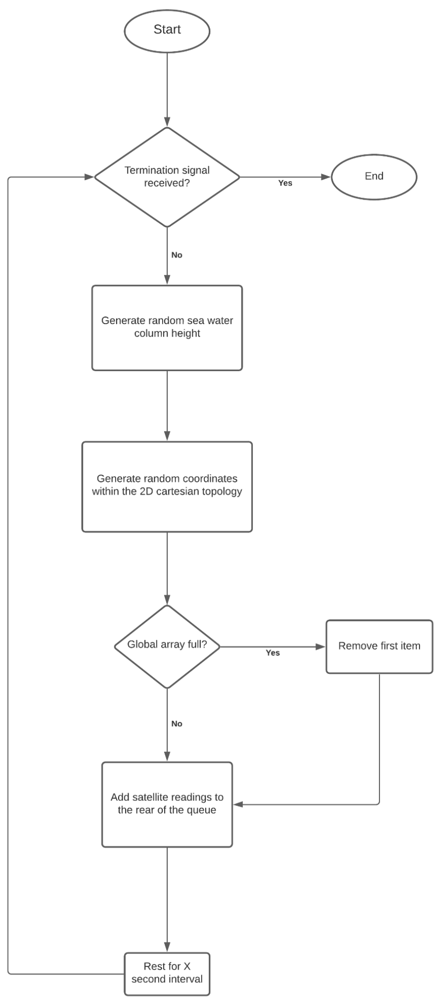

Start
Termination signal received?
Yes
End
No
Generate random sea water column height
Generate random coordinates within the 2D cartesian topology
Global array full?
Yes
Remove first item
No
Add satellite readings to the rear of the queue
Rest for X second interval

Figure x : Satellite Altimeter Flowchart

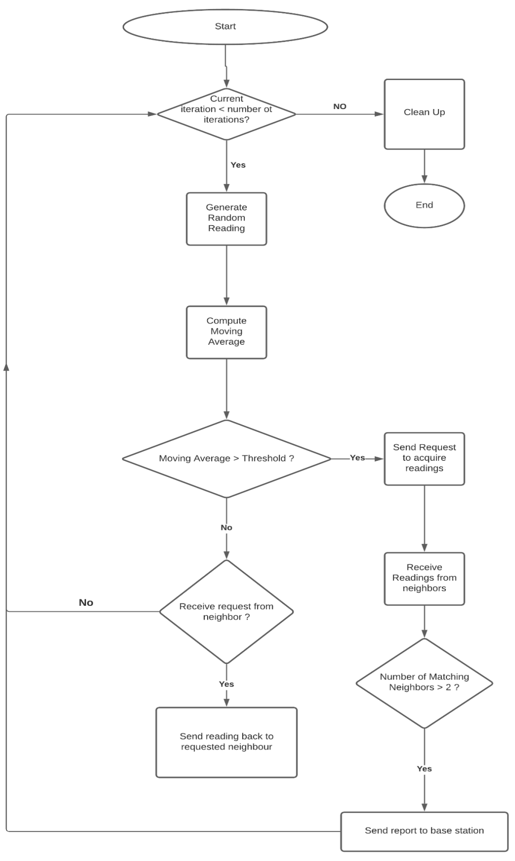

Start
Current iteration < number of iterations?
NO
Clean Up
End
Yes
Generate Random Reading
Compute Moving Average
Moving Average > Threshold ?
Yes
Send Request to acquire readings
Receive Readings from neighbors
No
No
Receive request from neighbor ?
Number of Matching Neighbors > 2 ?
Yes
Send reading back to requested neighbour
Yes
Send report to base station

Figure:
After the current reading is calculated randomly, the WSN node will update its moving average. Afterwards should it exceed the predefined threshold, the node will request information from its neighbours on their own moving average values. After receiving this information the node will then check to see whether any of the received values correlate with their own within a certain threshold value. If 2 or more do, then an alert report is sent to the base station node.

A disaster is an unforeseen event, which can overwhelm the capacity of the affected people to manage its impact. Many people are periodically exposed to natural disasters in their life, and most disasters, or more correctly hazards that lead to disasters, cannot be prevented. However, their effects can be mitigated.

Tsunamis such as the devastating boxing day tsunami of 2004, originating from a magnitude 9.1 earthquake off the coast of Sumatra Indonesia, have caused irreversible damage to the lives of many people.[1] Up until recent years they have been looked at as a force of nature that is difficult to predict, unlike hurricanes and other more visible phenomena. Now however there are systems that can help with early detection of such tsunamis, and our report is on one such system that uses a Wireless Sensor Network (WSN) to detect anomalous water height readings, subsequently alerting authorities of possible tsunami threats before they reach coastlines across the world. A WSN can gather and process data at a much faster rate than human interpreters, all at a lower maintenance cost as a small team could manage a significant number of sensors. They are also not prone to human error, where forward observers such as coast guard authorities may miss key signs of a tsunami. Their operations are not hindered by harsh weather conditions or human sleep cycles, and with adequate maintenance they can maintain near round the clock vigilance. In this paper we will be looking at the use of a WSN for monitoring ocean height levels, alongside a simulated satellite altimeter and base station, with the goal of identifying possible tsunami threats before they reach coastlines and showing that the threat of tsunamis can be significantly averted through the use of WSN networks.

Our WSN utilises an Inter Process Communication architecture, where data is exchanged between MPI processes layered in a 2D Cartesian grid topology. Sensor nodes will rely on readings from their neighbours when prompted,

Our proposed hypothesis is that higher tolerances will lead to a higher ratio of “correct” identifications, however this should be tempered to avoid too many false positives, and also that the system will display efficient scalability with iterations and size.

Environment monitoring has become an increasingly important tool to predict future events of natural disasters (ie: Tsunami, Earthquake, Forest fire etc.)
Wireless Sensor Network has been widely used in many areas especially for surveillance and monitoring in agriculture and habitat monitoring.

An intelligent and smart Wireless Sensor Network system can gather and process a large amount of data from the beginning of the monitoring and manage air quality, the conditions of traffic, to weather situations. In this paper, we discuss and review wireless sensor network applications for environmental monitoring.

In recent years, the issue of forest fires that transpired in distant parts of the globe such as Australia, Russia and the United States of America (USA) have resulted in increased global awareness and concern on the profound consequences of destructive conflagrations to the public health [3]. As a result, governmental and non-governmental organizations are collaborating to incorporate ground based wireless sensors to detect abnormalities in the surface temperature for actionable intelligence.

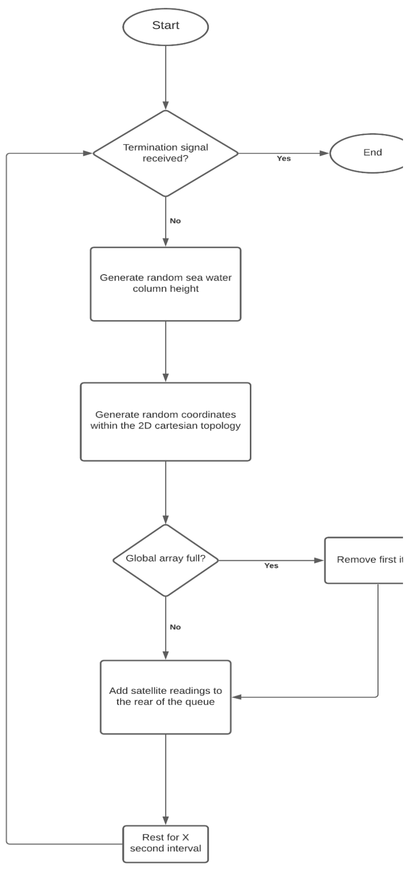
This work attempts to simulate a wireless sensor network (WSN) and satellite network for alert detection through an Inter Process Communication (IPC) architecture. An IPC is a mechanism that allows the exchange of data between processes [4]. The motive for implementing the simulation in a 2D Cartesian grid topology where sensor nodes communicate between their adjacent neighbours is because this architecture offers scalability and efficiency [5].

The hypothesis proposed is that for a given grid size, the rate of true alters will not be larger than the rate of false alters while the number of iterations increases and also different grip with the same number of iterations will also affect the rate as well.

Here is an overview about the following section: Section II introduces the design and architecture of the detection system. Section III analyses the performance of the architecture and Section IV concludes the report.

- Paraphrase to form our introduction
    -
    - The aim of the paper : design ,implement and analyze a distributed wireless sensor network application for tsunami detection.
    - Mention the deadliest tsunami https://www.history.com/news/deadliest-tsunami-2004-indian-ocean
    - The paper contains 3 sections

- Algorithm pseudocod


Start
Termination signal received?
Yes
End
No
Generate random sea water column height
Generate random coordinates within the 2D cartesian topology
Global array full?
Yes
Remove first i
No
Add satellite readings to the rear of the queue
Rest for X second interval

e
- Result tabulation analysis
- Conclusion

- IMportant to mention our hypothesis : Our proposed hypothesis is, given a fixed grid size, the number of true alerts over the total number of alerts received (true alert ratio) decreases as the tolerance window increases.
-

The hypothesis proposed is that for a given grid size, the rate of true alters will not be larger than the rate of false alters while the number of iterations increases and also different grip with the same number of iterations will also affect the rate as well.

Conclusion

- Random number generation
- Relationship between tolerance and the number of true alert and iteration
- Picking a good threshold is very important,
  - Large threshold ; allow of true alerts = inaccurate / too sensitive

    6x 7, 10,40,60,100,250 iterations , tolerance 50, threshold 6100

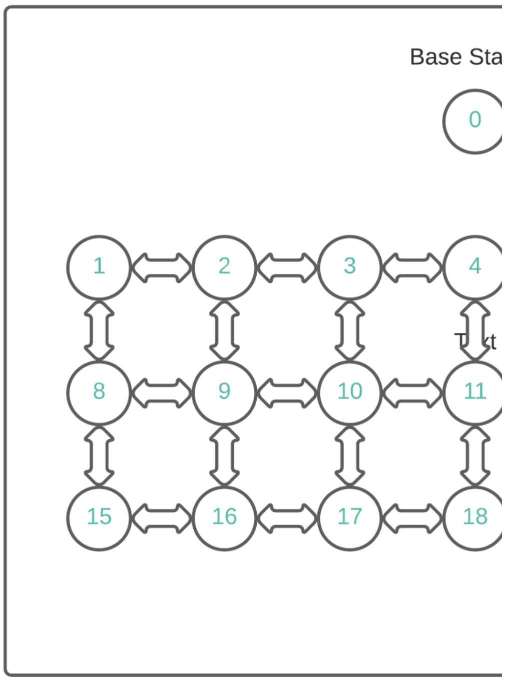
    6x 7, 10,40,60,100,250 iterations , tolerance 100

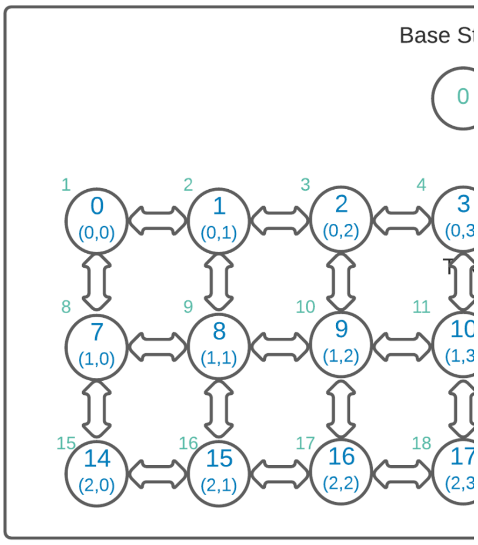
    6x 7, 10,40,60,100,250 iterations , tolerance 250

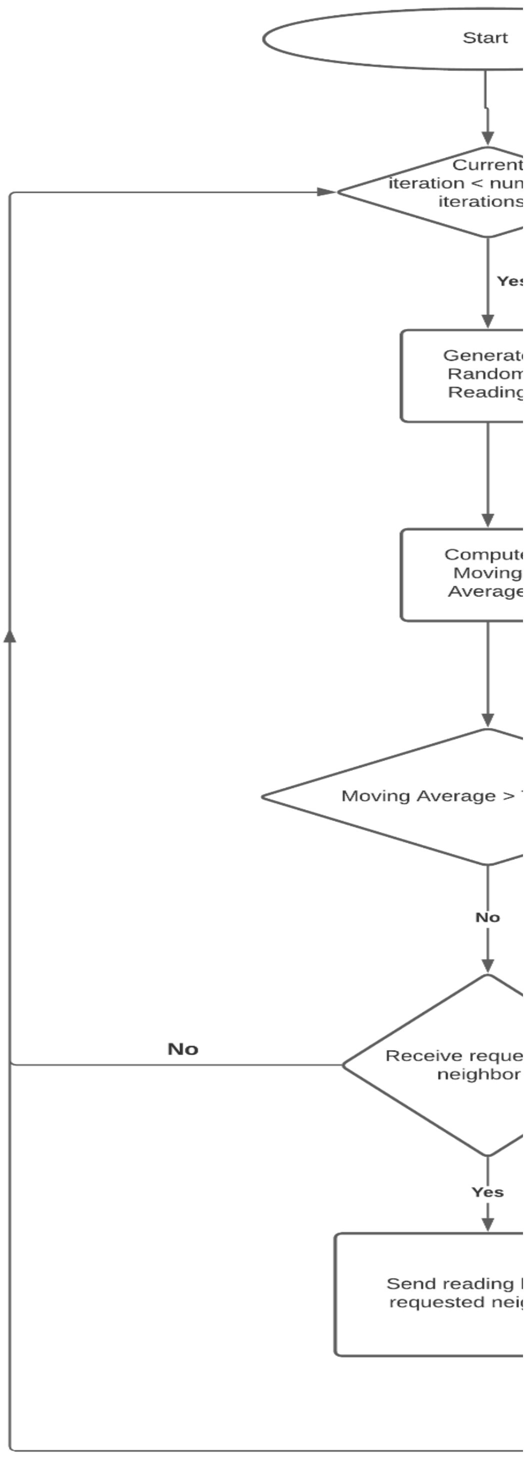
    Analyse threshold

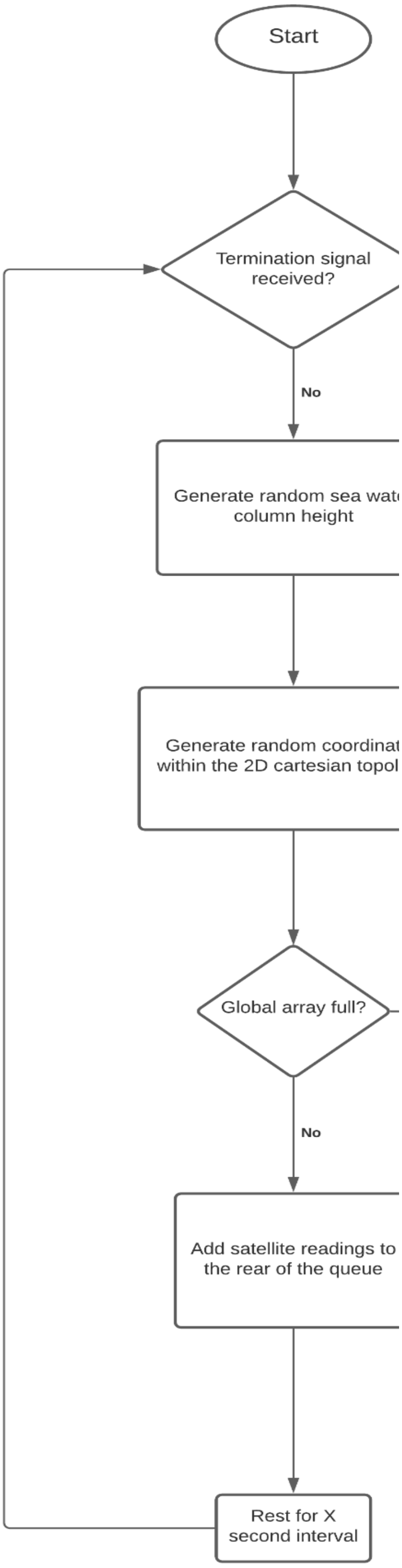
  -

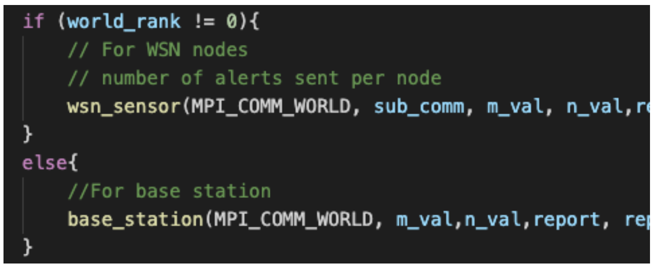
https://lucid.app/lucidchart/8291c584-1813-4cc1-ac38-29d440663d8a/edit?view_items=oCWmRqssja-C&invitationId=inv_e77f38d8-c776-43a1-b5be-f8afe4d3dabe


Base Sta
0
1
2
3
4
T xt
8
9
10
11
15
16
17
18


Base St
0
1
0
(0,0)
2
1
(0,1)
3
2
(0,2)
4
3
(0,3
T
8
7
(1,0)
9
8
(1,1)
10
9
(1,2)
11
10
(1,3
15
14
(2,0)
16
15
(2,1)
17
16
(2,2)
18
17
(2,3

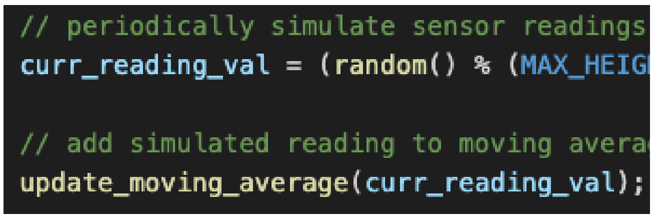

Start
Current
iteration < num
iterations
Yes
Generate
Random
Reading
Compute
Moving
Average
Moving Average >
No
No
Receive reque
neighbor
Yes
Send reading
requested nei


Start
Termination signal received?
No
Generate random sea wat
column height
Generate random coordinat
within the 2D cartesian topol
Global array full?
No
Add satellite readings to the rear of the queue
Rest for X second interval

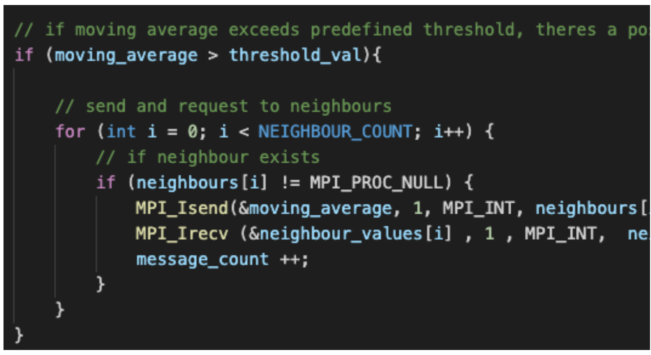
- 

```
if (world_rank != 0){
    // For WSN nodes
    // number of alerts sent per node
    wsn_sensor(MPI_COMM_WORLD, sub_comm, m_val, n_val,re
}
else{
    //For base station
    base_station(MPI_COMM_WORLD, m_val,n_val,report, rep
}
```

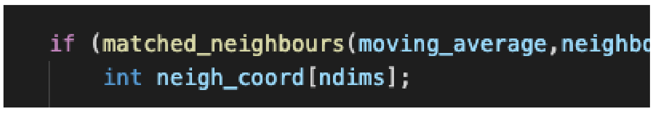
-
-
- **Wsn network**
    - Alert detection



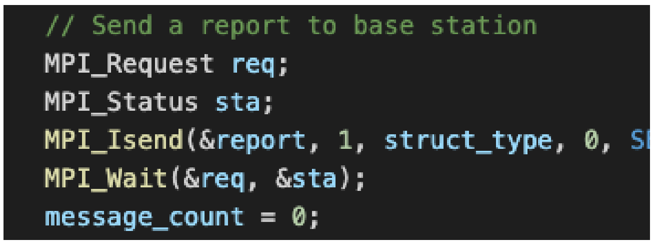
```
// periodically simulate sensor readings
curr_reading_val = (random() % (MAX_HEIG

// add simulated reading to moving avera
update_moving_average(curr_reading_val);
```

- 

```
// if moving average exceeds predefined threshold, theres a po
if (moving_average > threshold_val){

    // send and request to neighbours
    for (int i = 0; i < NEIGHBOUR_COUNT; i++) {
        // if neighbour exists
        if (neighbours[i] != MPI_PROC_NULL) {
            MPI_Isend(&moving_average, 1, MPI_INT, neighbours[
            MPI_Irecv (&neighbour_values[i] , 1 , MPI_INT,  ne
            message_count ++;
        }
    }
}
```

- 

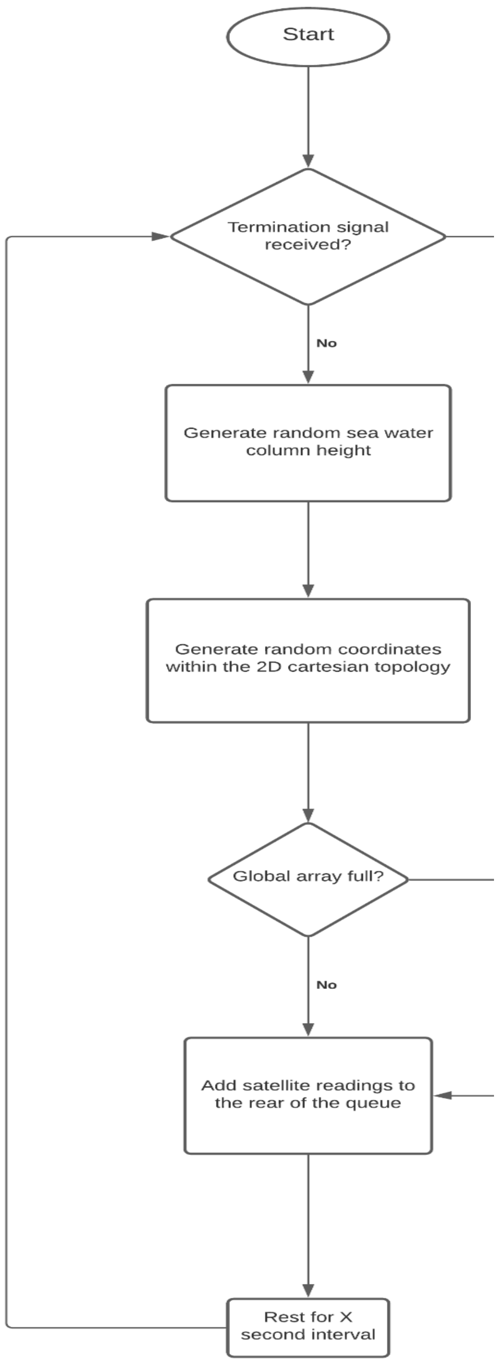
```
  if (matched_neighbours(moving_average,neighbo
      int neigh_coord[ndims];
```

- 

```
  // Send a report to base station
  MPI_Request req;
  MPI_Status sta;
  MPI_Isend(&report, 1, struct_type, 0, SI
  MPI_Wait(&req, &sta);
  message_count = 0;
```

-
- Base station
    - Alert detection
- Satellite simulator

- Generate random readings and push to global array


Start
Termination signal received?
No
Generate random sea water column height
Generate random coordinates within the 2D cartesian topology
Global array full?
No
Add satellite readings to the rear of the queue
Rest for X second interval

WSN Flowchart Annotation:

At the start of the node, it first checks to see if there are any further iterations pending. If there are, they will generate their random reading and then use it to compute the latest moving_average. At the next step it will check to see whether this moving average has exceeded the given threshold. If it has, a request will be sent to all adjacent nodes to provide their moving averages as well. Should more than 2 of the moving averages from adjacent nodes match, a report is sent to the base station. The iteration from earlier repeats until exit conditions are in effect, then it will clean up and exit.

Base station

-

Base station flowchart annotation:

At the start of the base station node, the satellite altimeter global array is created. Then the altimeter itself is created as a thread. Should there be iterations remaining, the base station will check to see if there are any alerts received. If so, it will compare the value contained within the report with the satellite readings. If the difference between the reports reading and the satellites reading is less than the given tolerance, a true alert is logged. If not, a false alert is logged. These iterations repeat until exit conditions are met, upon which a summary of results is logged and the base station will exit.

Satellite altimeter

Satellite altimeter flowchart annotation:

At the start of iteration, the satellite altimeter will check if the termination signal is received. If not, it will generate a random sea water height reading. Then it will generate random coordinates within the 2D cartesian topology. If the global array for storing such values is full, the first item is removed and then the reading stored. If not then the reading is appended to the rear of the array. The altimeter will then rest for x seconds before resuming iterations. When the termination signal is received the altimeter will clean up and exit.

Screenshot

- Sample log file screenshots

Design and Implementation of Alert Detection In A Distributed Wireless Sensor Network (WSN)

Hew Ye Zea
School of Information Technology
Monash University Malaysia
yhew0003@student.monash.edu

***Abstract* – Wireless Sensor Networks (WSN) are responsible for monitoring and collecting data to a centralized database. In this paper we propose an efficient implementation of sensor nodes in a 2D grid using a cartesian topology and inter process communication (IPC). The number of messages sent between the sensor nodes and the base station are minimized using point-to-point communication. This results in an alert detection system with a drastic speedup improvement. Message Passing Interface (MPI) is used for communication between nodes and the base station.**

***Index Terms* – WSN, cartesian topology, MPI, IPC**

I. INTRODUCTION

Wireless Sensor Networks (WSN) are widely used for collecting environmental data and collating the collected data at a common location which is the base station. WSN is made up of a collection of nodes where each node has processing power and energy constraints [1]. To enable efficient communication between processes, Inter Process Communication (IPC) is required. Message Passing Interface (MPI) constitutes a form of IPC implemented at library level [2] and will be used since sensor nodes and the base station are represented by MPI processes.

Sensor nodes in the WSN network can only communicate with its adjacent neighbours. By using MPI's virtual cartesian topology, a node's coordinates and neighbours can be easily identified and obtained. The topology is reordered by MPI to allow optimal process ordering which improves the efficiency of communication between nodes.

Communication between sensor nodes and the base station are kept at a minimum only when a sensor node is required to send an alert. In this work, the objective is to use MPI to find a form of efficient communication between nodes and the base station. We suggest that there is an efficient way in implementing this in MPI by using point-to-point communication and storing data in a struct to reduce the number of MPI send calls required.

During the simulation, the base station uses POSIX threads to simulate infrared readings while periodically listening for alerts sent by sensor nodes. When it receives an alert, it will compare the alert's reading and timestamp to its own infrared reading produced. To speed up comparisons, a lookup table is used. We hypothesize that as the number of events increases, the communication time required increases.

In the following sections we investigate *"How to implement an efficient alert detection system in a Wireless Sensor Network?"*

II. INTER PROCESS COMMUNICATION – MPI

A. 2D Cartesian Grid Topology

Sensor nodes are arranged in a 2D cartesian grid topology. The base station and sensor nodes are represented by MPI processes. Each node is able to communicate with its immediate adjacent nodes (left, right, top, bottom). The base station has the ability to communicate with all the nodes in the grid.

The program uses a dynamic grid where the height and width of the grid is specified by the user through command line arguments.

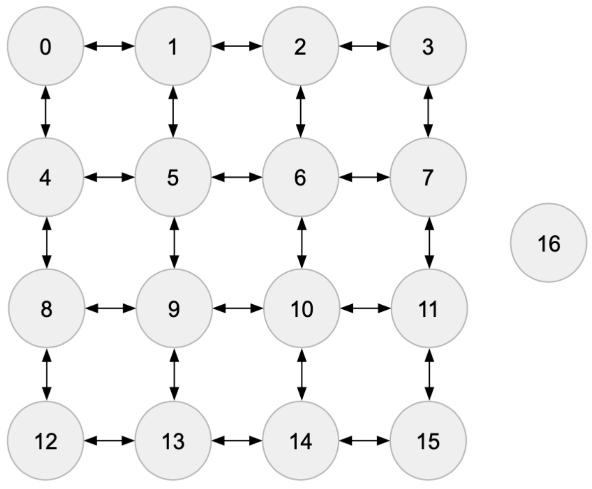


Figure 1: 4 × 4 grid layout. 16 processes starting from 0 represent the sensor nodes while an additional process, process 16 represents the base station.

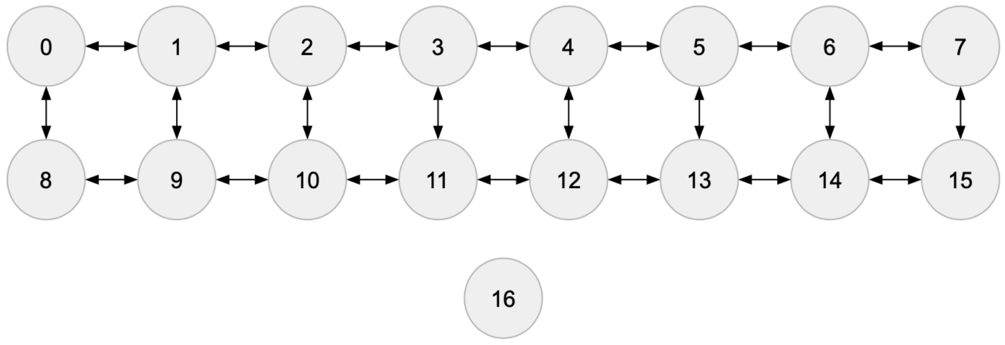


Figure 2: 8 × 2 grid layout. 16 processes represent the sensor nodes and process 16 represents the base station.

1) Cartesian grid implementation: The cartesian virtual topology is used to create a 2D grid for the sensor nodes. This architecture is chosen due to its ability to reorder processes for better communication performance. The cartesian topology is also used in other benchmark applications with n–dimensional grids such as CORAL HACC and Snap [3].

MPI_Cart_create initialises the topology and creates a new communicator for the processes in the 2D grid. Sensor nodes use the new communicator to send and receive messages in the 2D grid.

Algorithm 1 – Initialising a cartesian grid

```
int n = 2;
// col & row specified by user
int dim[2] = {col, row};
int period[2] = {0, 0};
int reorder = 1;
//grid communicator
MPI_Comm comm;
MPI_Cart_create(MPI_COMM_WORLD, n,
dim, period, reorder, &comm);
```

2) Obtaining sensor nodes' neighbours: One of the advantages of MPI's cartesian topology is that the sensor node's immediate adjacent neighbours are easily obtained using **MPI_Cart_shift**. If an adjacent neighbour does not exist in a certain direction ie. process 1 has no top neighbour (Figure 2), **MPI_Cart_shift** returns **MPI_PROC_NULL**.

Algorithm 2 – Locating sensor node's neighbours

```
MPI_Cart_shift(comm, 1, 1, &left,
&right);
MPI_Cart_shift(comm, 0, 1, &up,
&down);
```

B. Event detection criterion

There are two main parts in this section – event detection between sensor nodes and alert detection by the base station. A more detailed flowchart of the simulation can be found in Figure 4.

1) Event detection by sensor nodes: Each node generates a random number which represents a sensor reading. If the sensor reading is above a predefined threshold, it sends its own sensor reading to its neighbours as a message to request for their sensor readings. Sensor nodes use point-to-point communication. Non-blocking **MPI_Isend** is used to prevent deadlocks since blocking MPI primitives can cause deadlocks [4] if calls are not well arranged.

After the request messages are sent, all nodes use **MPI_Iprobe** to check for any incoming messages. If there exists a message, the sensor node uses a blocking **MPI_Recv** to accept the request message from its neighbour. A blocking receive is used after using **MPI_Iprobe** because we are aware that there exists an incoming message so deadlocks would not occur. Since it received its requesting neighbour's reading, the sensor node could compare its own reading to its neighbour's reading to check if it is in the tolerance range. Only when the reading is in the tolerance range, it sends its own reading back to the neighbour that requested it.

The proposed method reduces the number of messages sent between neighbours in the case of an event since a sensor node only sends its own reading back to the requesting neighbour if it's in the threshold range. Finally, the requesting node uses **MPI_Iprobe** one last time to check if its neighbours sent back their own readings and **MPI_Recv** to accept its neighbours' readings. If a sensor node receives two or more readings, it implies that the node has to send an alert to the base station.
The alert will contain these details in its message that is stored in a struct:

- Sensor reading
- Time taken
- Timestamp of event
- Number of messages exchanged
- Readings received from adjacent nodes
- IP address
- MAC address

2) Alert detection by base station: While the sensor nodes are generating readings in the WSN, the base station spawns five POSIX threads to generate random numbers which represent the infrared imaging satellite's readings. The reports which contain the node rank, reading and timestamp are in a struct which is then stored in an array. The index of the array corresponds to the node rank. Based on Figure 3, if there are 4 processes, the array of size 4 contains reports of node rank 0 to 3.

Node 0	Node 1	Node 2	Node 3
{reading, timestamp}	{reading, timestamp}	{reading, timestamp}	{reading, timestamp}

Figure 3: Lookup array for infrared readings

At the same time, the base station listens for incoming alerts sent by sensor nodes. It uses **MPI_Iprobe** to check for alerts. If it receives an alert, it uses the array in Figure 3 which serves as a lookup table to efficiently compare its own infrared satellite's reading to the sensor node's reading. This process is efficient because the base station only has to check for the content in the index position of the sensor node to obtain the reading generated instead of searching through the whole array.

3) Base station logs: After the base station compares the readings, it logs into a text file whether it is a true or false alert. The details sent from the sensor node, the infrared reading generated and the total number of true and false alerts are also logged for analysis.

4) Sentinel value: In each iteration, the **select(2)** [5] function is called to check if the user has pressed the 'enter' key as sentinel value. When a sentinel value is received, the base station sends a termination message to all processes to complete their current send and receive operations before the program gracefully terminates.

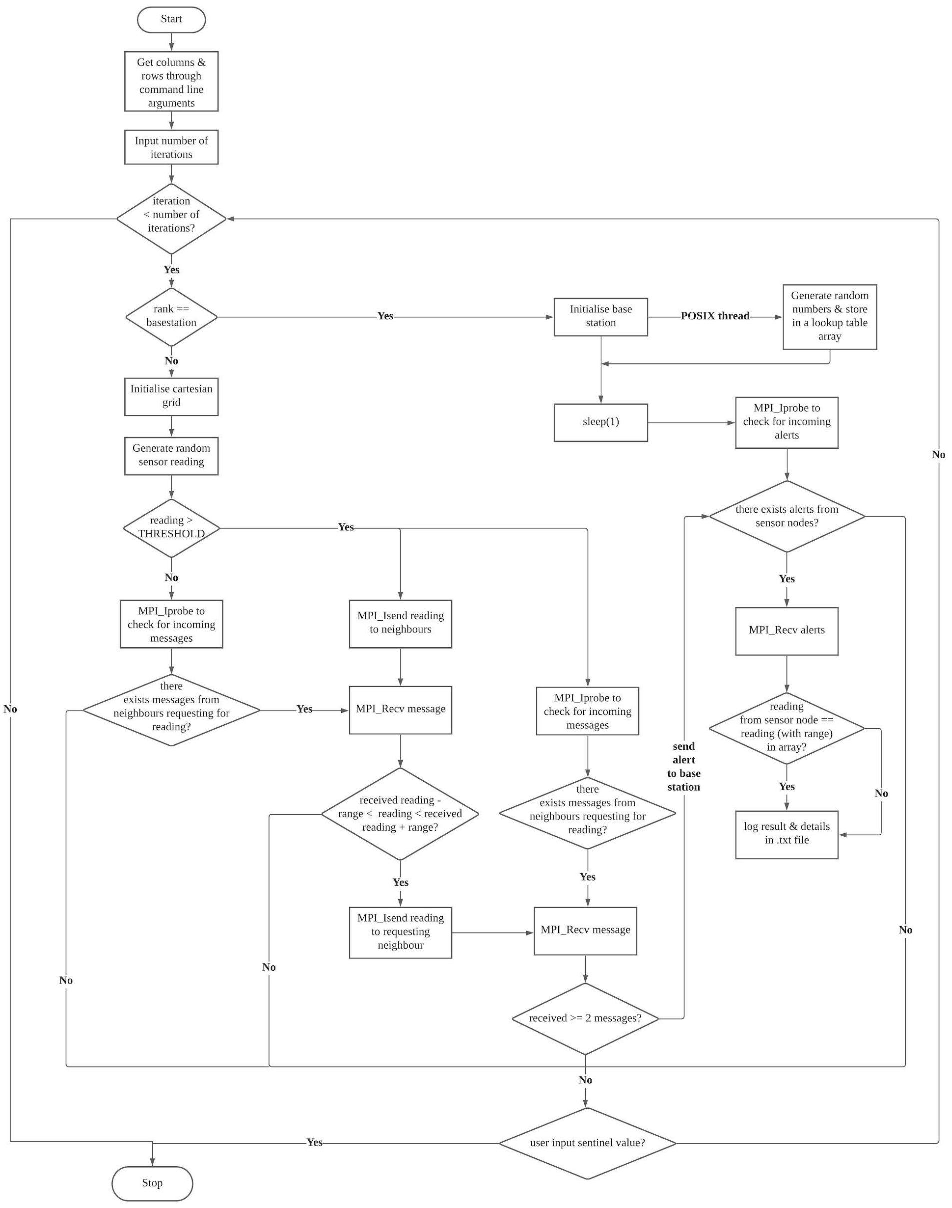

Start
Get columns & rows through command line arguments
Input number of iterations
iteration < number of iterations?
Yes
rank == basestation
Yes
Initialise base station
POSIX thread
Generate random numbers & store in a lookup table array
sleep(1)
MPI_Iprobe to check for incoming alerts
No
Initialise cartesian grid
Generate random sensor reading
reading > THRESHOLD
Yes
No
there exists alerts from sensor nodes?
Yes
MPI_Recv alerts
MPI_Iprobe to check for incoming messages
MPI_Isend reading to neighbours
there exists messages from neighbours requesting for reading?
Yes
MPI_Recv message
MPI_Iprobe to check for incoming messages
send alert to base station
reading from sensor node == reading (with range) in array?
Yes
No
log result & details in .txt file
received reading - range < reading < received reading + range?
there exists messages from neighbours requesting for reading?
Yes
Yes
MPI_Isend reading to requesting neighbour
MPI_Recv message
No
No
No
No
received >= 2 messages?
No
user input sentinel value?
Yes
No
Stop

Figure 4: Flowchart of the WSN simulation

III. RESULTS AND DISCUSSION

A. Simulation Specifications

Simulation of the program is performed on a local MacBook Pro computer and MonARCH's cluster computing setup (refer to Figure 9 for the specifications of the MonARCH job). Each simulation lasts for 100 iterations. The specifications of the systems and the simulation parameters are detailed below.

MacBook Pro

Specification	Value	Description
CPU	4	Number of CPUs/ cores
RAM	16	Amount of memory

MonARCH Xeon-E5-2680-v4 @ 2.40GHz

Specification	Value	Description
CPU	28	Number of CPUs/ cores
RAM	241.66	Amount of memory
Nodes	1	Number of nodes

Simulation parameters

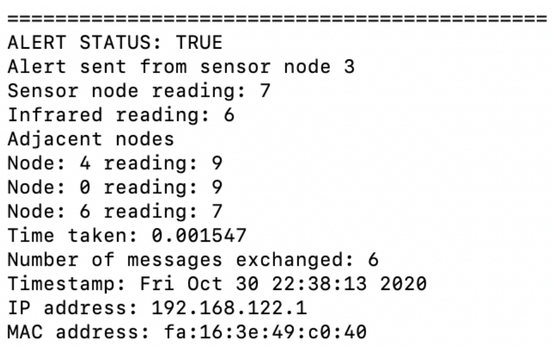
Parameter	Value	Description

Rows	3	Rows in the grid
Columns	3	Columns in the grid
Number of processes	10	Processes used in the simulation
Wait time	1 s	Time interval between iterations
Sensor threshold	5	Trigger event above this value
Tolerance range	3	Tolerance range of sensor reading
Reading range	0 - 10	Generate random numbers in this range

B. Base Station Log File

In each iteration of the simulation, the base station logs the sensor nodes' event details, alert details and the total number of alerts received.

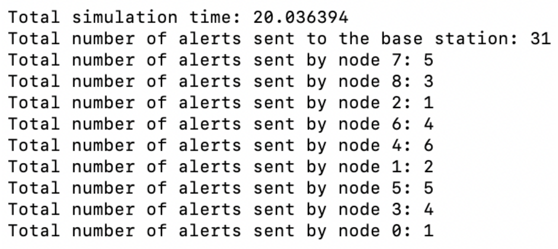
In Figure 5, during iteration 0 the base station received an alert from sensor node 3 where it has a reading of 7. Since the sensor node's top, bottom and right neighbours both obtained readings that are in
sensor node 3's threshold range, this constitutes an alert which must be sent to the base station. The base
station compares its own infrared satellite reading, 6 to the sensor node's reading. Since both of the readings are in the threshold range, it is recorded as a true alert. The base station obtains the event details which includes important information such as the number of messages exchanged as well as the summary of the number of true, false and total alerts per iteration and logs it in a file.



```
=============================================
ALERT STATUS: TRUE
Alert sent from sensor node 3
Sensor node reading: 7
Infrared reading: 6
Adjacent nodes
Node: 4 reading: 9
Node: 0 reading: 9
Node: 6 reading: 7
Time taken: 0.001547
Number of messages exchanged: 6
Timestamp: Fri Oct 30 22:38:13 2020
IP address: 192.168.122.1
MAC address: fa:16:3e:49:c0:40
```

Figure 5: Event details

After the simulation has ended the log file records a summary of the simulation which includes the total simulation time, number of alerts received by the base station and the number of alerts sent by each sensor node.



```
Total simulation time: 20.036394
Total number of alerts sent to the base station: 31
Total number of alerts sent by node 7: 5
Total number of alerts sent by node 8: 3
Total number of alerts sent by node 2: 1
Total number of alerts sent by node 6: 4
Total number of alerts sent by node 4: 6
Total number of alerts sent by node 1: 2
Total number of alerts sent by node 5: 5
Total number of alerts sent by node 3: 4
Total number of alerts sent by node 0: 1
```

Figure 6: Summary of the simulation

C. Communication Time

The results obtained are tabulated and analysed in this section. Since the MacBook Pro only has 4 cores, it is oversubscribed to perform the simulation.

	MacBook Pro	**MonARCH**
Number of events	342	354
Average communica t-ion time (s)	0.004285	0.000108547
Total simulation time (s)	565.28	200.169815

The MonARCH cluster computing setup has outperformed the single computer, MacBook Pro setup significantly. The total simulation time of MonARCH is less than half of the MacBook Pro's simulation time. The average communication time between sensor nodes and the base station in MonARCH is also significantly shorter which can be seen in Figure 7. This is because MonARCH has enough logical cores to perform the simulation while the MacBook Pro has to oversubscribe which is an issue caused by an insufficient number of CPUs. MonARCH also has a higher RAM than my local computer.

The communication time is correlated with the number of events and messages sent in an iteration. The higher the number of events, the longer the communication time is as more messages are exchanged between processes. To prove this, we can compare the upward and downward spikes in Figures 7 and 9. This might be caused by the known issue that MPI_Recv is a form of blocking communication that slows down the overall communication time.

By considering our hypothesis from section I of this paper and the outcomes of the simulations, it is clear that the higher number of events, the longer the communication time between MPI processes during simulation.

IV. CONCLUSION

In this paper we have demonstrated a simulation of an alert detection system in WSN using MPI processes. By using MPI's cartesian topology, we have found an effective way in implementing a 2D grid for sensor nodes. We have minimized the number of messages exchanged between sensor nodes and the base station by using point-to-point communication. We have also proposed an efficient way in comparing readings in the case of an alert by using a lookup table which is the objective of this paper.

By analysing the results obtained from the simulations conducted, there are many future works that could be done to improve this project. One of the possible improvements is to implement a simpler form of communication between sensor nodes. Currently, sensor nodes communicate using point-to-point communication but other alternatives such as MPI Neighborhood collectives can be explored.

Another possible future work to be explored can also include clock synchronization between the sensor nodes and the base station. This includes a possible scenario where the nodes and base station are located at different geographical locations and time zones.

REFERENCES

[1] R. Sharma, Y. Chaba and Y. Singh, "An IPC key management scheme for Wireless Sensor Network," 2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010), Solan, 2010, pp. 251-255, doi: 10.1109/PDGC.2010.5679906.
[2] Mirtaheri, S. ., Khaneghah, E. ., & Sharifi, M. (2008). A Case for Kernel Level Implementation of Inter Process Communication Mechanisms. 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, 1–7. https://doi.org/10.1109/ICTTA.2008.4530360
[3] Gropp, W. (2018). Using Node Information to Implement MPI Cartesian Topologies. Proceedings of the 25th European MPI Users' Group Meeting, 1–9. https://doi.org/10.1145/3236367.3236377
[4] Sala, K., Teruel, X., Perez, J. M., Peña, A. J., Beltran, V., & Labarta, J. (2019). Integrating blocking and non-blocking MPI primitives with task-based programming models. Parallel Computing, 85, 153–166. https://doi.org/10.1016/j.parco.2018.12.008
[5] Linux Manual Page. (2020). Select(2) – Linux manual page. Retrieved from https://man7.org/linux/man-pages/man2/select.2.html

APPENDIX

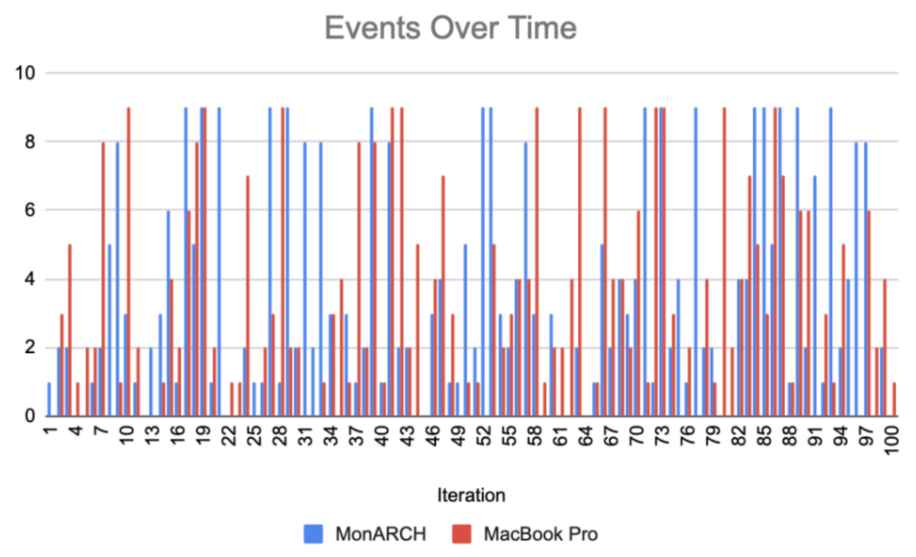


Figure 7: Chart of triggered events over time

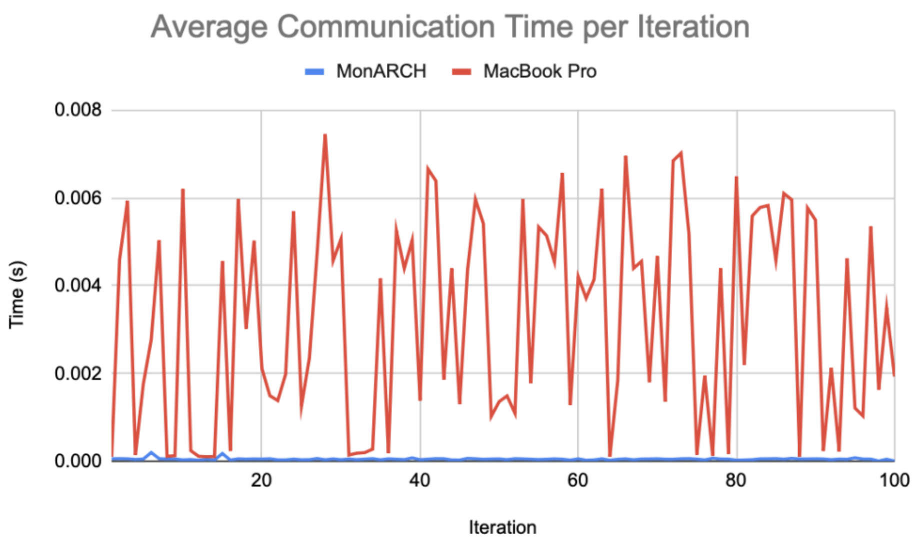


Figure 8: Chart of average communication time per iteration

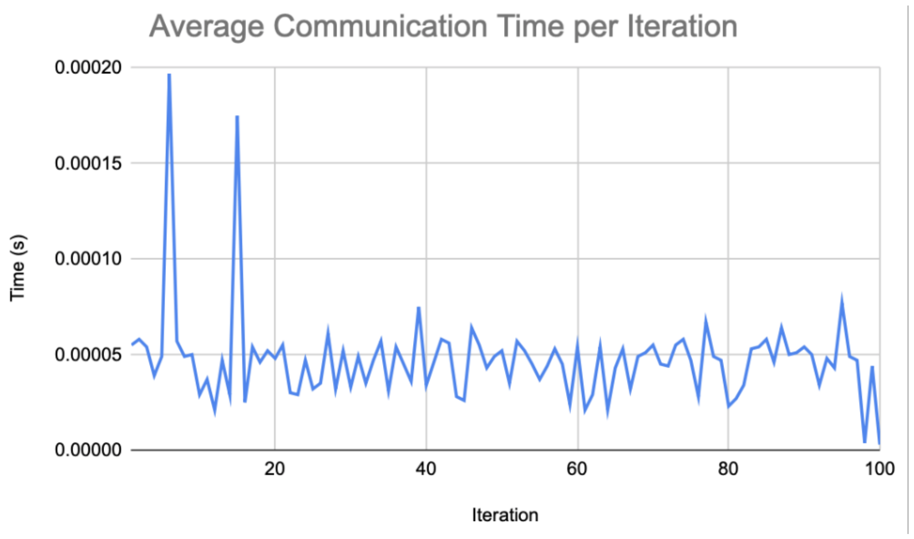
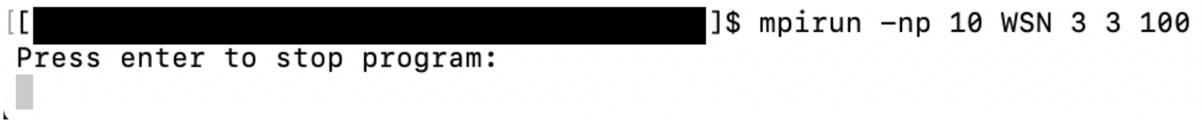
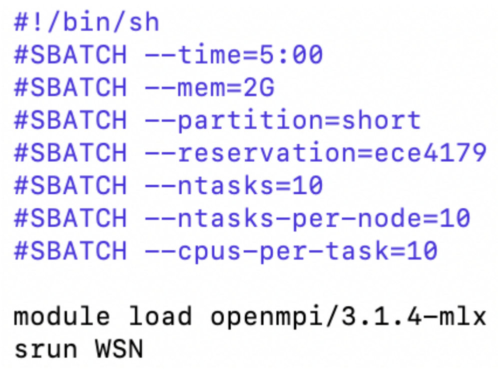


Figure 9: More detailed chart of average communication time per iteration using MonARCH



```
[[redacted]]$ mpirun -np 10 WSN 3 3 100
 Press enter to stop program:
```

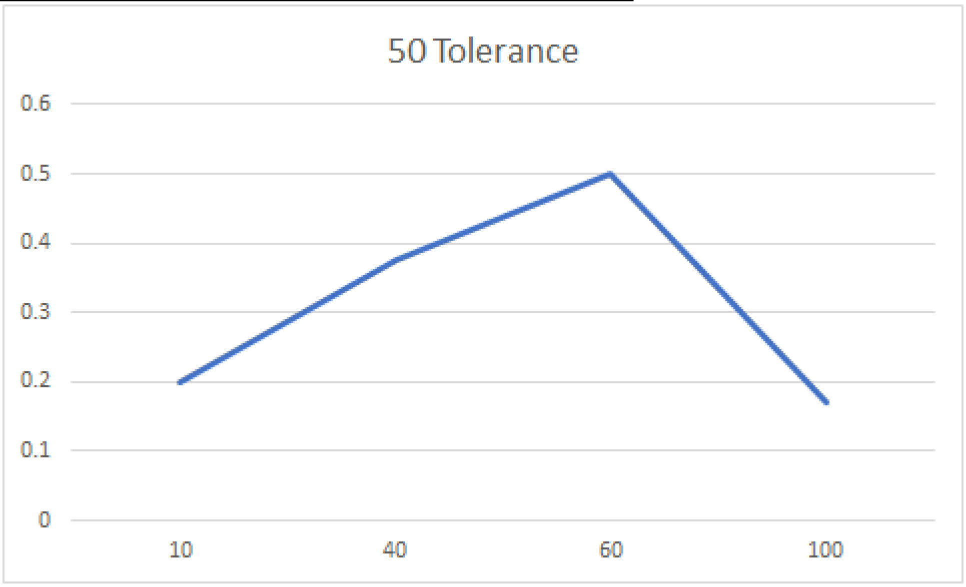
Figure 10: Screenshot of the running program. Rows, columns and number of intervals are entered as command line arguments. The user is also presented with an option to enter the sentinel value `enter` to stop the program.



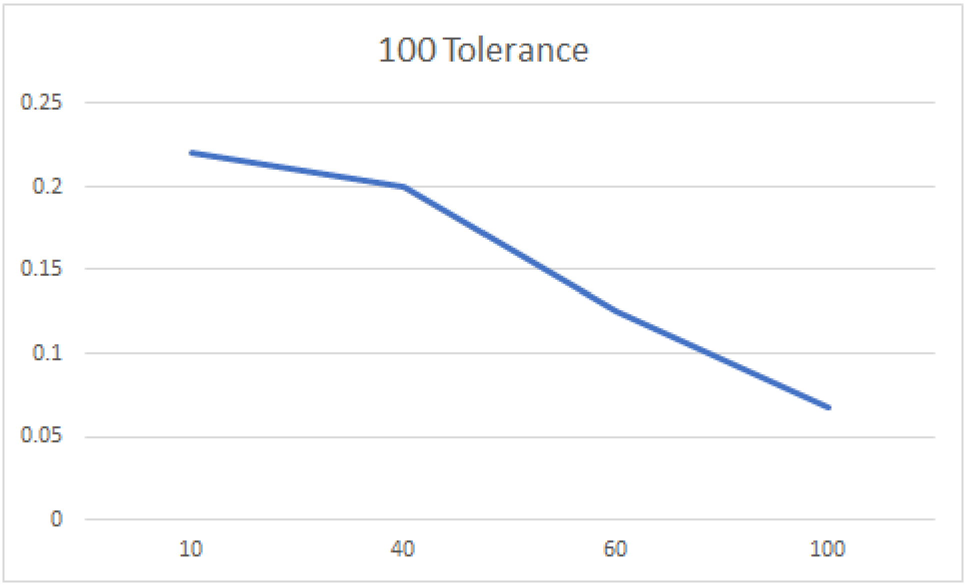
```
#!/bin/sh
#SBATCH --time=5:00
#SBATCH --mem=2G
#SBATCH --partition=short
#SBATCH --reservation=ece4179
#SBATCH --ntasks=10
#SBATCH --ntasks-per-node=10
#SBATCH --cpus-per-task=10

module load openmpi/3.1.4-mlx
srun WSN
```

Figure 11: Screenshot of the MonARCH job

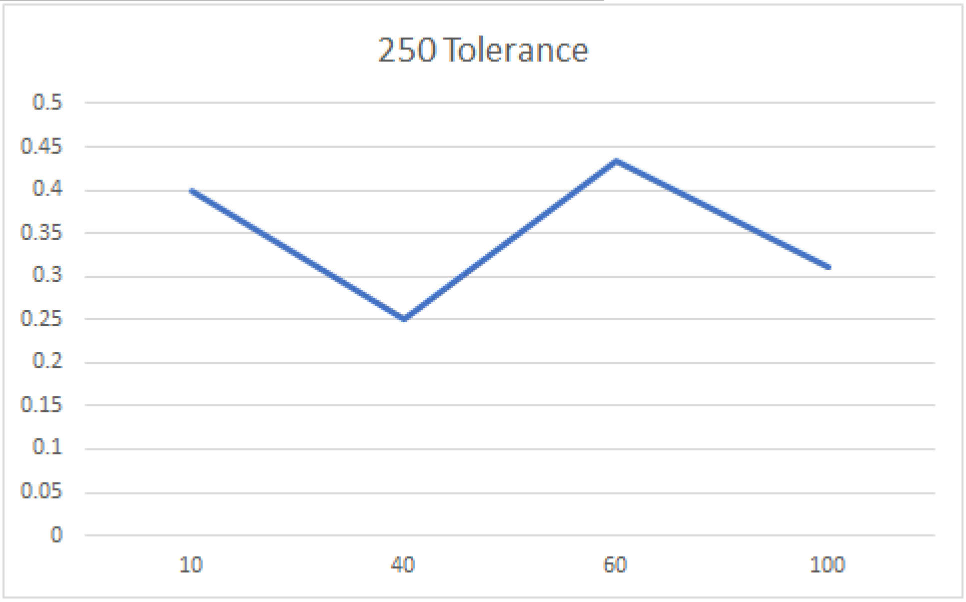
Tolerance	Total Alerts	True alerts	Correct ratio
50	100	17	0.17
50	10	2	0.2
50	40	15	0.375
50	60	30	0.5



Tolerance	Total Alerts	True alerts	Correct ratio
100	100	22	0.22
100	10	2	0.2
100	40	5	0.125
100	60	4	0.067

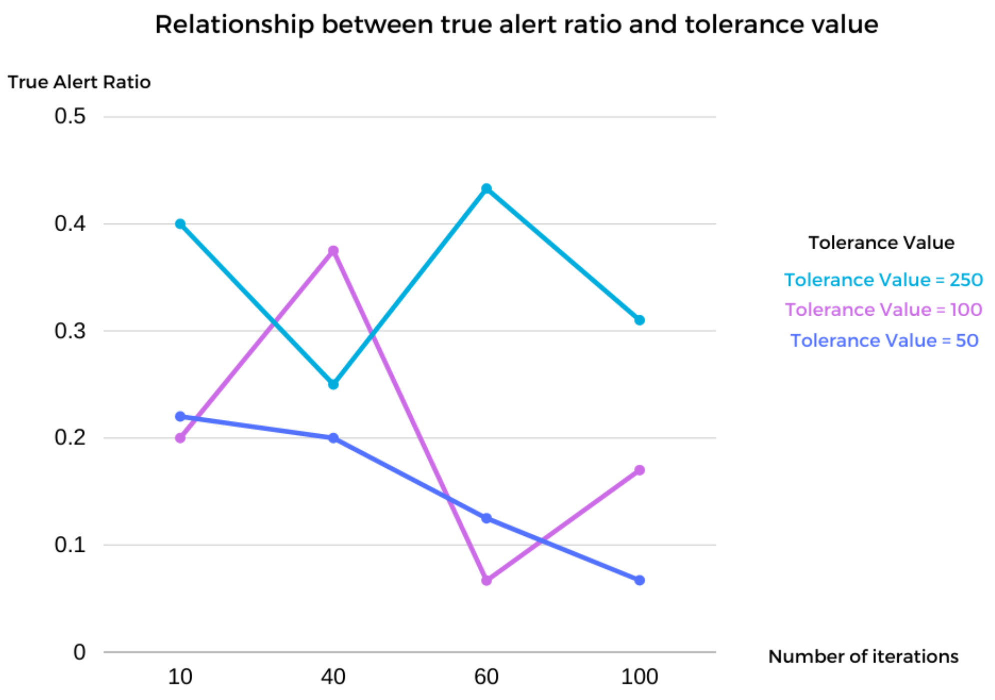


Tolerance	Total Alerts	True alerts	Correct ratio
250	100	31	0.31
250	10	4	0.4
250	40	10	0.25
250	60	26	0.433

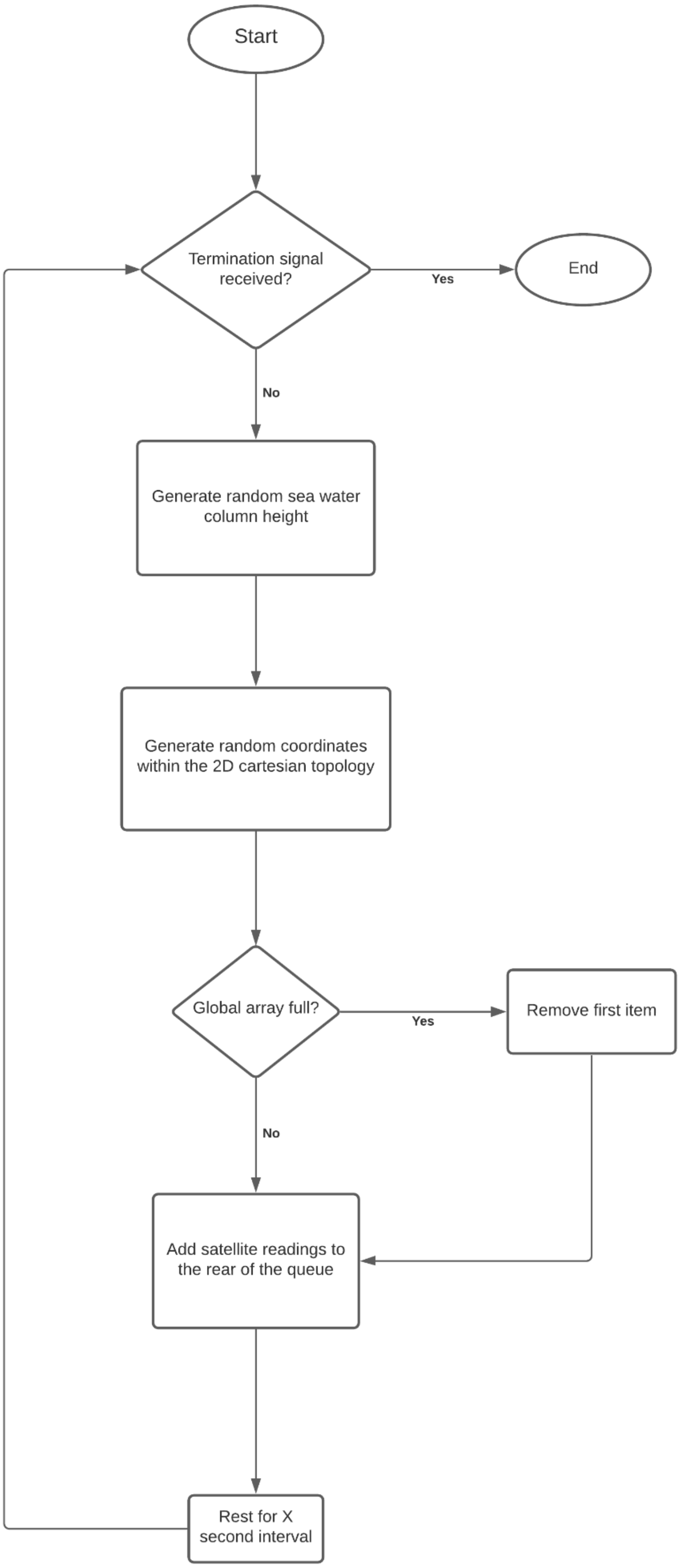


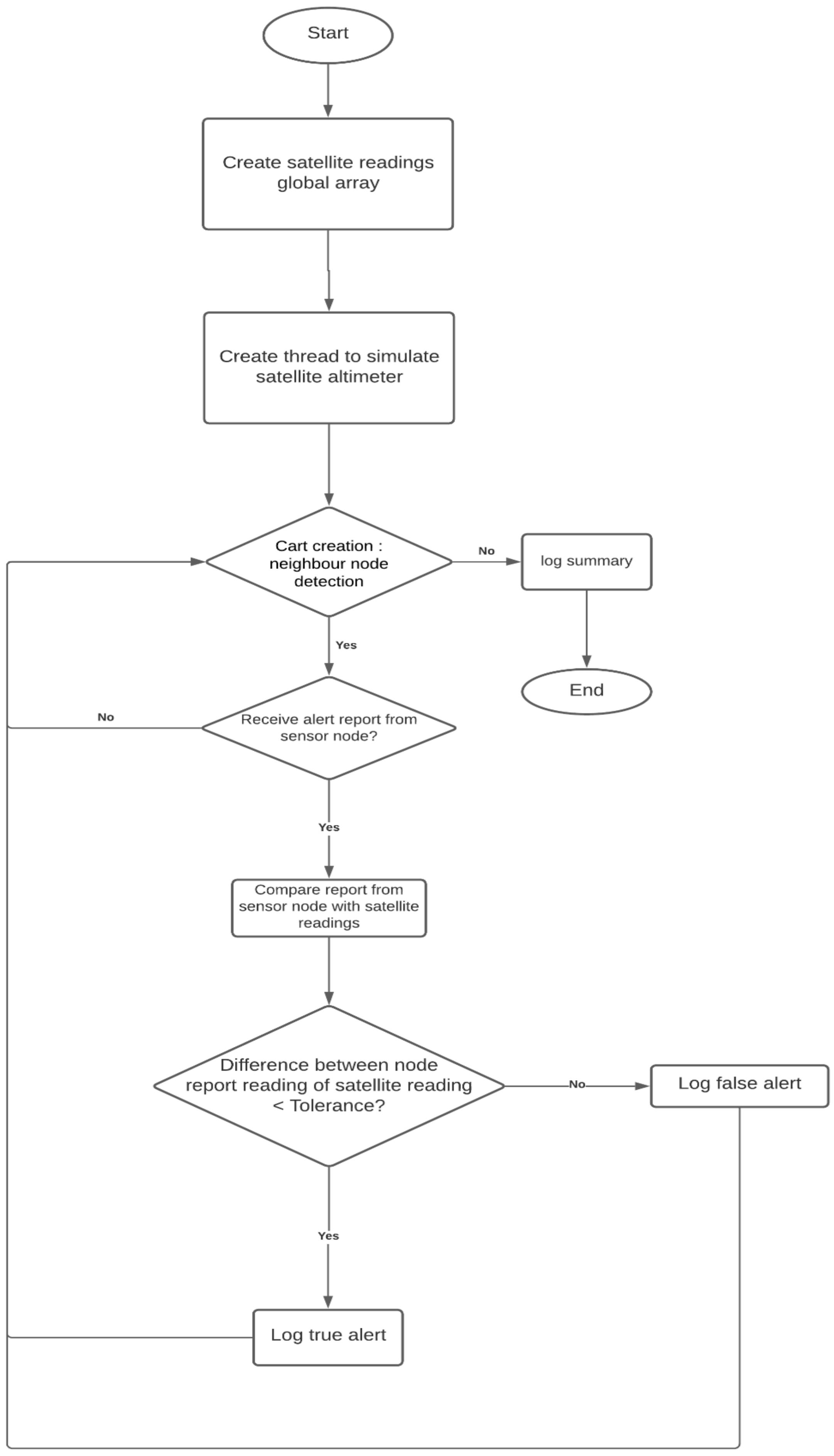
Tolerance	Iterations	Ratio
50	10	0.2
50	40	0.375
50	60	0.5

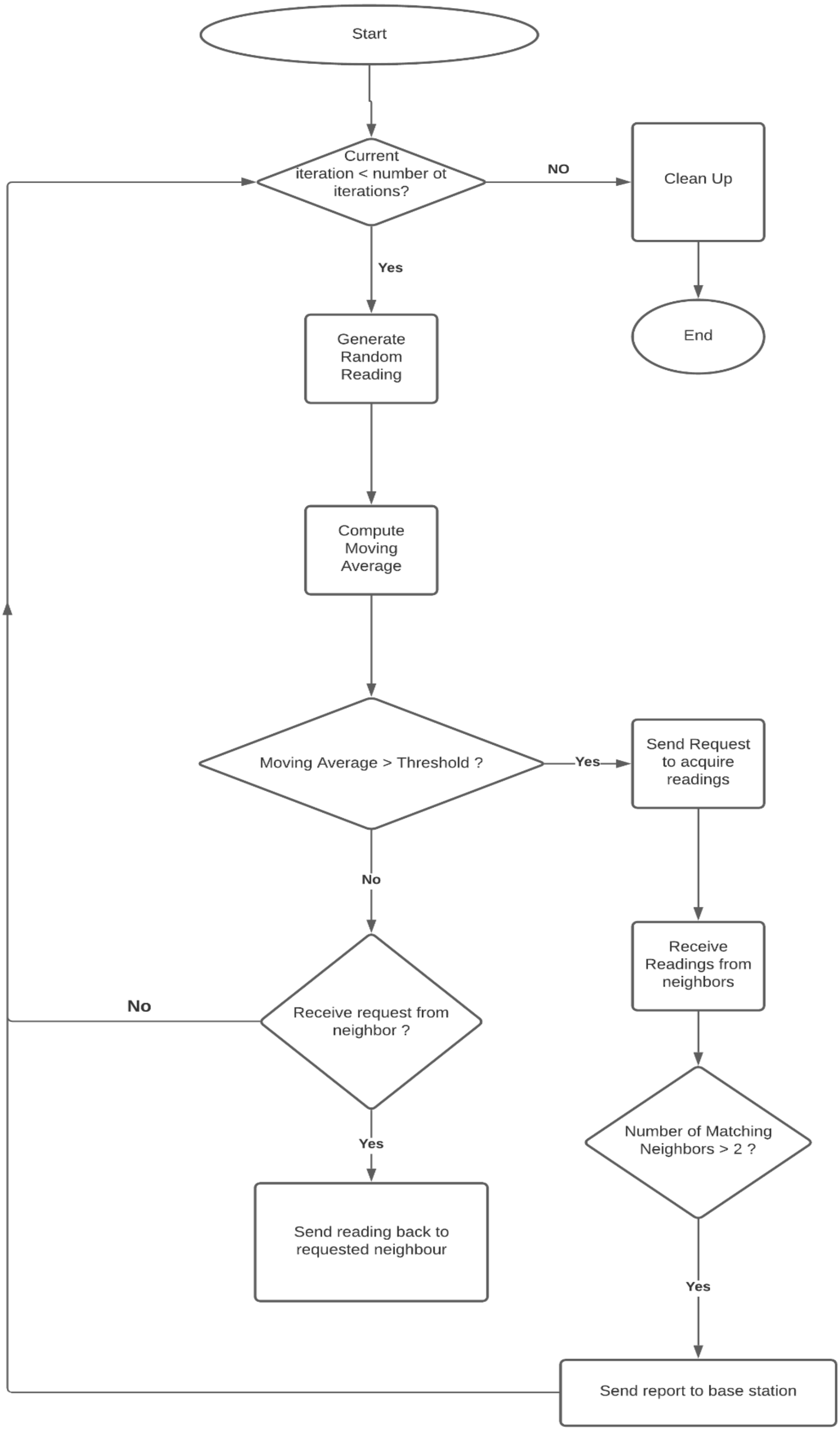
50	100	0.17
100	10	0.22
100	40	0.2
100	60	0.125
100	100	0.067
250	10	0.4
250	40	0.25
250	60	0.433
250	100	0.31



The correct ratio represents the ratio of true alerts divided by the total number of iterations.When you look at a certain number of iterations (for example, 10 iterations) and the correct ratio for 10 iterations at different tolerances, the correct ratio increases as the tolerance increases (0.4 at tolerance of 250, while its only 0.2 at a tolerance of 50). This makes sense as the larger the acceptable tolerance the more values will be accepted and not rejected as more values will fit within the tolerance window above the threshold. However the tolerance should not be set too high, as too high a tolerance will lead to a significant increase in correct ratio that may include several very low values as well. This may lead to “true” alerts which are in fact false, as no tsunami was present.


Start
Termination signal received?
Yes
End
No
Generate random sea water column height
Generate random coordinates within the 2D cartesian topology
Global array full?
Yes
Remove first item
No
Add satellite readings to the rear of the queue
Rest for X second interval


Start
Create satellite readings global array
Create thread to simulate satellite altimeter
Cart creation : neighbour node detection
No
log summary
End
Yes
No
Receive alert report from sensor node?
Yes
Compare report from sensor node with satellite readings
Difference between node report reading of satellite reading < Tolerance?
No
Log false alert
Yes
Log true alert


Start
Current iteration < number ot iterations?
NO
Clean Up
End
Yes
Generate Random Reading
Compute Moving Average
Moving Average > Threshold ?
Yes
Send Request to acquire readings
No
Receive Readings from neighbors
No
Receive request from neighbor ?
Yes
Send reading back to requested neighbour
Number of Matching Neighbors > 2 ?
Yes
Send report to base station