

*Model Implementation of a Distributed Fire Detection System with MPI and POSIX Threads*

Ethan Nardella

*FIT3143*

*Monash University*

Melbourne, Australia

enar0002@student.monash.edu Philip Chen

*FIT3143*

*Monash University*

Melbourne, Australia

pche64@student.monash.edu

*Abstract*— Wildfires and deforestation is becoming a larger global concern. In order to prepare and combat against these fires, Wireless Sensor Networks (WSN) can be used as a monitoring system to monitor these environment [1]. The primary aim of this investigation is to simulate a WSN model and simulate the interaction between the WSN with a base station and satellite in using POSIX. The secondary aim in this investigation is to provide an analysis of the implementation through examining the reports generated by the algorithm. The design and implementation presents the WSN implementation using OpenMPI and the interaction between the WSN with a base station and satellite in using POSIX. This also includes a discussion regarding design choices. The model is then used to simulate the detection of wildfires and deforestation. Results of these simulations are then presented and discussed. From these results, it can be observed that for this implementation, the performance for nodes to detect wildfires and notify the base station does not appear to decrease over a long period of time.

Keywords—cartesian topology, OpenMPI, MPI, POSIX

# Introduction

Wildfires and deforestation are becoming an increasing problem around the globe. In order to pre-emptively detect and fight these fires, organisations have begun implementing ground based networks of wireless sensors for detecting abnormally high surface temperatures. Upon detecting abnormally high temperatures, these sensors communicate and compare their readings against an expected temperature computed by an infrared satellite orbiting the Earth. Alerts are then raised to relevant authorities to combat fires for if these comparisons reach a pre-defined threshold at a specific time range.

The primary aim of this investigation is to model the above initiative for pre-emptively detecting and fighting forest fires. To do so, a Wireless Sensor Network (WSN) to simulate a network of wireless sensors for detecting abnormal surface temperatures. In addition to the network, a satellite and a base station has also been modelled as part of the simulation. The WSN model is arranged in a two dimensional cartesian topology. Each node in the topology uses is able to send messages to nodes immediately adjacent to it in the cartesian representation and may also communicate with the base station which has also been modelled as a node. Communication is simulated using the Open Message Parsing Interface (Open MPI) library in C. Additionally, the model of this implementation also uses POSIX threads for MPI rank level parallel implementations. MPI allows for Inter Process Communication (IPC) [2], which supports sending and receiving information between otherwise distinct processes.

Each node represented a temperature sensor device, which has the capacity to record the temperature around it and report any anomalies where the temperature is above a specified threshold. The application of this system is to detect fires to combat deforestation by quickly reporting and tracking events. Nodes then communicate with the base station, which compares the reports with information from a satellite equipped with an infrared camera (simulated via an additional thread in the base station node). These reports are documented for analysis.

It was hypothesised that the number of nodes would impact the runtime of the program linearly after a certain number of nodes is present. This is because prior to this point, the base station node is idle, and will not fall behind on serving these nodes. This critical point is determined by the combined factors of the processing capacity of the system running the model (how fast requests can be processed), and the custom temperature threshold set (how many alerts will need to be processed).

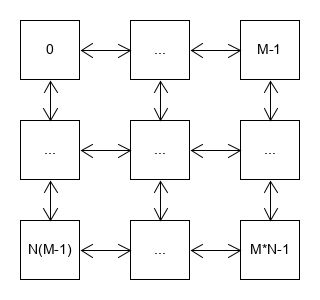
The secondary objective of this research is to provide an analysis of the implementation through examining the reports generated by the algorithm.

# Design and Implementation

## Sensor Node

A number of components were implemented in order to simulate the WSN as listed below:

* Cartesian topology: MPI\_Cart\_create was used to create a cartesian topology, then using the MPI library to manage the cartesian topology (eg. MPI\_Cart\_coords for getting cartesian coordinates of current MPI rank)
* Request and messaging system: a combination of MPI message passing calls was implemented as part of the simulation of communication between sensor nodes. These include calls such as MPI\_Isend, MPI\_Irecv, MPI\_Waitall, and MPI\_Test.



1. Diagram illustrating m \* n cartesian topology for WSN

For this implementation, a dynamic m columns by n rows cartesian topology was simulated as illustrated in Fig1 above. As discussed above, MPI\_Cart\_create was used to create a cartesian topology. As such, each node would be accessed though an MPI rank. For this reason, for all script runs, the number of ranks that will be used will always be m\*n. As shown in Fig1, m\*n-1 ranks will be used to depict the cartesian topology. An additional node is used for the Base Station and Satellite component of the implementation.

Using these components, the WSN behaviour were simulated using the cartesian topology component of the Sensor Node implementation. Each sensor node runs for a fixed number of runs, specified by the user. The nodes continuously generate a random temperature value and upon detecting a temperature above the predefined 80.0f threshold, the node sends a non-blocking message to adjacent cartesian nodes as a request for their temperatures. Comparisons are made to neighbouring nodes. When the difference between two of these messages responses and the message response fall within the tolerance threshold, a message is sent from the node to the Base Station rank which will be described in the following section.

## Base Station and Satellite

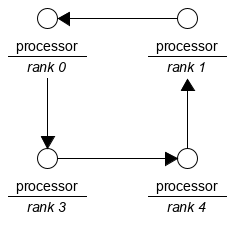
The Base Station and Satellite were implemented on one MPI process, using a separate POSIX thread [3]. The POSIX thread allows multiple programs to be executed concurrently, saving time by executing one process while the other process is idle. In this instance the authors elected to have the main thread waiting to serve the main node network, and the additional POSIX thread to emulate the satellite by randomly generating detected temperatures within the same range as the sensor nodes.

## Script algorithms

Appendix 1 Fig 2, is a flow chart illustrating the code logic for WSN, base station, and satellite implementation. Certain algorithms were used as part of the code and have been depicted in the flow chart. A more detailed discussion fo these algorithsm can be found below.

|  |
| --- |
| **Algorithm 1** Simulating nonblocking requests and responses |
| 1. Initialise MPI\_Requset for request sent to neighbour nodes 2. Iterate through neighbour node ranks 3. If rank is valid, use MPI\_Isend to send request 4. Sleep script for an interval of time 5. Check using MPI\_Test the request status 6. If request was unsuccessfully received, cancel request and free resources |

Algoirthm 1 presented above summarises the process of simulating nonblocking reqests send by WSN nodes when the node detects that its temperature is greather than the threshold. Nonblocking MPI\_Isend has been used in this process as a measure of deadlock avoidance. As shown in Fig1 in the section A of Design and Implementation, all nodes can communicate in both directions (to and from) when communicating with neighbouring nodes. Fig2 below illustrates a scenario where four processors enter a deadlock. A scenario like this may occur in a three by three cartesian topology. In this scenario, rank 4 is requesting for neighbouring process rank 1 to provide its temperature. Likewise, rank 1 is requesting for rank 0, rank 0 is requesting for rank 1, and rank 1 is requesting for rank 4. If the request and message passing system in this implementation was using block MPI\_Send and MPI\_Recv, the allocation of resources would result in a deadlock indicated in Fig2 below. Thus, for the purpose of deadlock avoidance, Algorithm 1 has been implemented in the code to simulate nonblocking requests when nodes request for temperatures.



1. Resource allocation diagram in the scenario of a deadlock for a three by three cartesian topology WSN

|  |
| --- |
| **Algorithm 2** WSN node logic and algorithm |
| 1. Check using MPI\_Irecv for whether the base station has sent a STOP command (this usually occurs at the end of all iterations) 2. If so, set the flag vairable to stop 3. Generate a random float to represent node temperature 4. Compare the generated temperature against the threshold temperature 5. If the temperature is greater than the threshold temperature, request temperature from each neighbour node using *Algorithm 1* 6. Check for responses using MPI\_Irecv 7. If the difference between two or more of these responses and the current temperature is less than the threshold, generate and send report to base station 8. Listen for requests for temperature from neighbour nodes, if there are requests, send temperature to rank of neighbour node 9. Check if stop flag as beeen set |

Algorithm 2 presented above describes the logic that has been implemented in the WSN nodes. This logic does not describe all checks and invocations made by each node, Algorithm 2 represents the logic a WSN node employs in the do-while clause after the environment variables have been initialized in an MPI thread for a node. In this algorithm, the node uses nonblocking MPI\_Irecv to receive messages from neighbouring nodes and also from the base station. At the beginning and end of this algorithm, the node will check and confirm whether the do-while loops requires termination based on messages passed from the base station node. The logic and checks that hve been implementted in this algorithm include the geneartion of node temperature, a comparison with the threshold, and a comparison with the tolerance. In the event where the threshold is met, Algorithm 1 is executed to issue nonblocking request to adjacent nodes. When the difference between the current node temperature and two or more adjacent node temperatures is less than the tolerance, reports are genearted, store dina buffer and sent to the base station. Note that nonblocking MPI\_Irecv has been used in this algorithm as part of the deadlock avoidance strategy.

## Script invocation

As mentioned in the Introduction, Design and Implementation sections, the script uses OpenMPI and POSIX threads as part of the implementation. Additionally, since random floats are also generated to simulate the node and satellite detected temperatures, the code also uses the math.h library. Thus, as shown in the makefile, the following commands are used to build and execute the model.

* Compiling script: since OpenMPI is being used as part of the implementation, the script compilation command will use *mpicc*. Additionally due ot the use of the math.h library, the script compilation will also use *-lm.* A sample command for the script compliation is as follows:

*$ mpicc ass2.c -o ass2\_out -lm*

* Running script: since OpenMPI is being used as part of the implementation, the compiled script execution command will use *mpirun* OpenMPI simulates a rank for each node in cartesian topology. Each rank is allocated a processor. However, since logical processors are an expensive component of computer hardware, it is unreasonable to expect a one to one ratio of nodes to physical logical processors. For this reason, -*overscubscribe* will be used as part of the command for invoking the compiled script to emulated additional processors being used to execute the script. Additionally, as mentioend in the Introduction, and indicated in Appendix 1 Fig6, the script is written to read in two additional parameters representing the number of columns and number of rows of nodes in the modelled cartesian topology. The script also reads in an additional parameter indicating the number of iterative rounds. For these many reasons, the compiled script execution command is typically structured as follows:

*$ mpirun -oversubscribe -np 21 ass2\_out 4 5 1000*In the line above, the invocation command uses *21* to represent the number of MPI threads that will be used. The *4* and *5* indicate the number of columns and the number of rows in the cartesian topology. Since each node will be represented by its own rank, it will use its own MPI thread. Thus, 4\*5+1 threads will be used in this invocation, where an additional thread is included to take the base station MPI thread into acocunt. Hence, the *21* MPI threads as part of the invocation command. The final parameter in the presented invocation aommand is the *1000* representing 1000 iterations of temperature generation and comparison in the WSN before termination.

# Results and Discussion

## Results

|  |
| --- |
| Number of nodes  Fig. 3. Chart showing time taken to complete 10000 iterations |

The results of the experiment show that as the nodes increase, the time of the algorithm remains relatively stable. This is demonstrated in the Fig3.

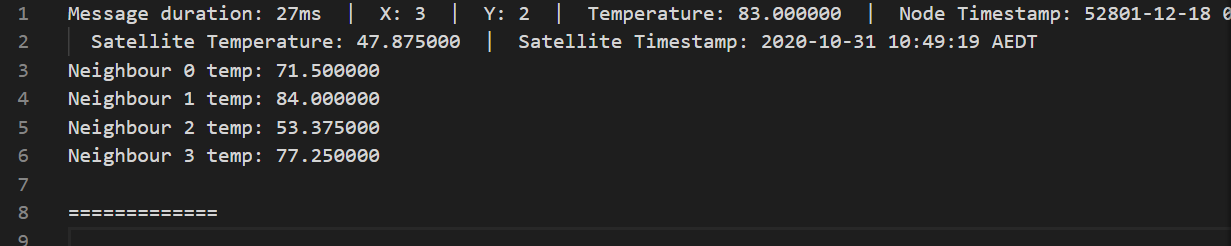
|  |
| --- |
| Number of iterations  Fig. 4. Plot showing time taken to complete |

The Fig4 shows how as the number of iterations increases, the time increases with it in a linear fashion. A linear trend can be observed from this plot where   
*time(ms) = 10.436 \* iterations + 4845.8* with a correlation factor of 0.9999. This recording ignores the small amount of time taken to initialise the nodes, which does scale with node size, but is eclipsed by the actual operation time on significant iteration counts.

## Discussion:

## The results do not support our hypothesis. The graphs in the results section indicate that the ability of the server node to serve requests is magnitudes beyond the number of successful requests a node will made. As such, as the nodes scale, for any physical implementation in a real situation, it will be unlikely the server node will be a bottleneck.

## As expected, the amount of time taken scales linearly with the input size. This shows that the nodes themselves have no impact on the linear runtime of the algorithm, and that it will thus be fit for the purpose discussed in the beginning.



1. Extract showing reports sent to base station

An extract of the report text file can be seen above. As part of this report, the following information is provided:

* Message duration
* Coordinates
* Temperature
* Node timestamp – note that this model assumes that all node clocks are in sync
* Satellite temperature
* Satellite timestamp
* Neighbor temperature

## Future Research:

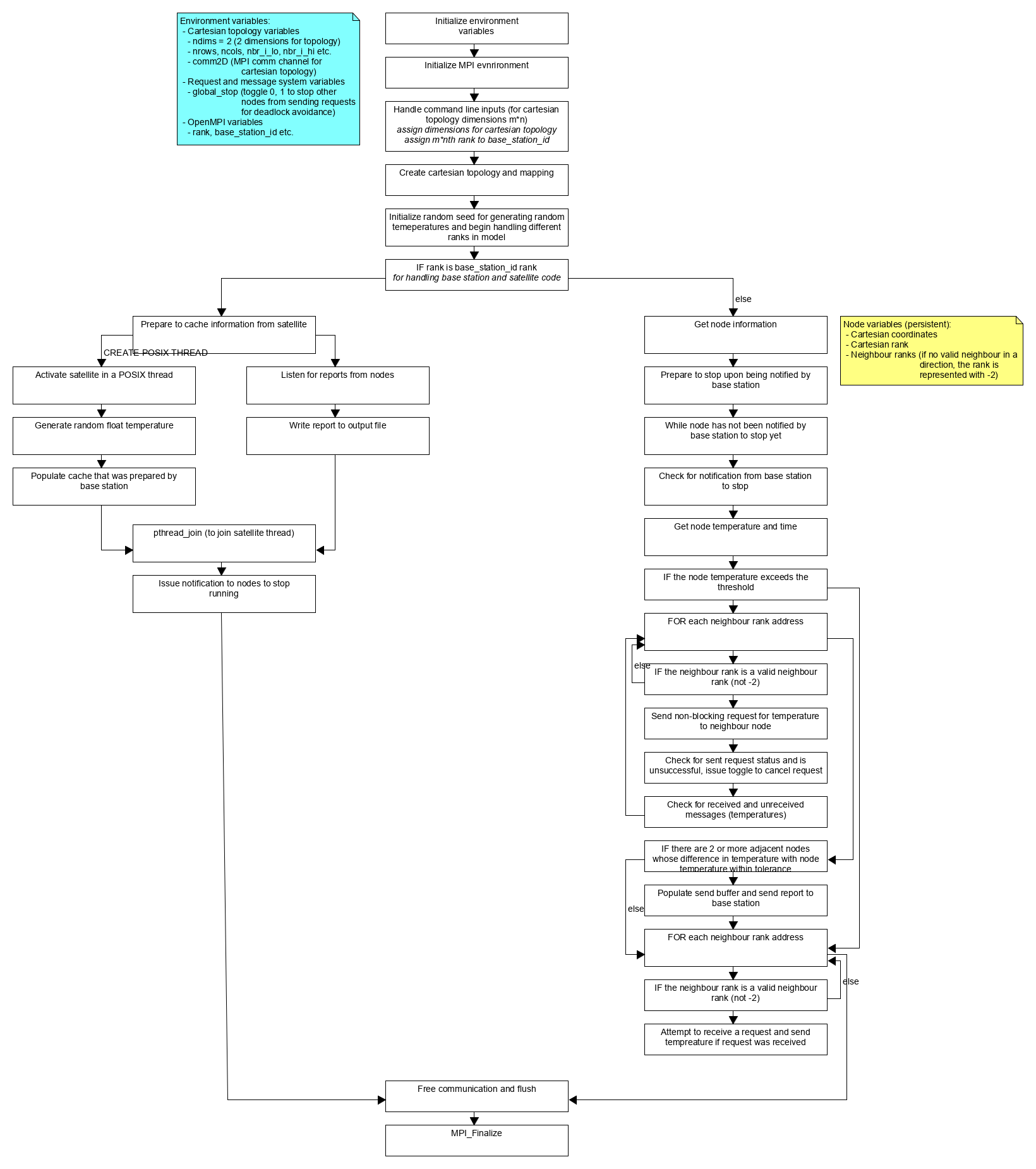
Room for future research is available given the inherent limitations of this implementation and simulation. As shown in Fig3 and Fig 4, the measured time from these simulations do not appear to provide realistic features as it will be unlikely the server node will be a bottleneck. Further exploration with improving the simulation design to realistically represent this could be conducted. Furthermore, further investigation can also be done for a large number of iterations to properly test the load of this model.

# Conclusion

As shown in this investigation, a WSN was modelled and could be used to conduct simulated. It appears that for this implementation, the performance for nodes to detect wildfires and notify the base station does not appear to decrease over a long period of time. It should be noted that these observations were based of a model with many limitations. For a more accurate representation, it would be valuable to further explore with improving the simulation design to realistically represent this could be conducted.

##### References

1. M. F. Othman and K. Shazali, “Wireless sensor network applications: A study in environment monitoring system,” Procedia Engineering, vol. 41, pp. 1204–1210, 2012
2. https://lms.monash.edu/mod/resource/view.php?id=7743767
3. Lamport, L., 1986. On interprocess communication. Distributed computing, 1(2), pp.86-101.
4. Barney, B., 2009. POSIX threads programming. National Laboratory. Disponível em:< https://computing. llnl. gov/tutorials/pthreads/> Acesso em, 5, p.46.



1. Diagram illustrating flow of code logic for WSN, base station, and satellite implementation