Event detection in a Fully Distributed Wireless Sensor Network (WSN)

Prepared by:
K. Sharsindra Pratheen (25636626)
School of Information Technology, Monash University
kspra3@student.monash.edu

Abstract: Event detection in a fully distributed Wireless Sensor Network (WSN) essentially possesses multiple real world applications and this brings us to improvise this Network System by proposing a method to parallelise this process by using Message Passing Interface and the use of OpenMP. This would theoretically provide a significant speed up that any implementation of a fully distributed Wireless Sensor Network (WSN) would perform flawlessly and have the best interest of its purpose for implementation. In order to simulate random occurrences of events within the WSN, each WSN node may be provisioned with an independent random number generator with the condition that at least ***three*** *adjacent nodes must produce the same random number, at the sampling time, to constitute an event.*

Keywords: Wireless Sensor Network (WSN), Message Passing Interface (MPI), OpenMP, Encryption, Inter Process Communication (IPC).

I. INTRODUCTION

A. Overview and Objectives of the Assignment & Inter Process Communication

The aim of this report is to shine light on the design and implementation of a promising construct of future technology, which is the intelligent Wireless Sensor Network (WSN) [1]. They consist of various nodes, in our scenario we have 20 nodes arranged orderly in a 4x5 grid topology and a base station mimicking the nearest neighbourhood concept. A nearest neighbourhood concept is supported by (Rajasegarar et. al. 2006) "We minimize the communication overhead by clustering the sensor measurements and merging clusters before sending a description of the clusters to the other nodes." Each and every one of these nodes are an MPI Process which means we have 21 MPI Processes in total [2]. Message Passing Interface (MPI) implies a class of parallel programming models which at the core of it is essentially simplified [3] Inter Process Communication (IPC). Messages are sent and received mostly as atomic units in MPI. MPI, designed by a team of researchers to function on a vast variety of parallel programs [4] is standardized and portable when it comes to message passing standards. We implement this Inter Process Communication in a C++ code that reflects the operation of the Wireless Sensor Network (WSN) including the base station in the most immaculate and efficient manner possible. The efficiency [5] of the system is measured by finding an Inter Process Communication scheme that reduces the messages to the base station while satisfying the Wireless Sensor Network (WSN)'s event detection requirements. On the other hand, the messages sent between adjacent nodes and to the base station have been encrypted [6]. Open Multi-Processing (OpenMP) is an Application Programming Interface (API) that provides multi-platform shared memory parallel process programming [7]. All in all, Open Multi-Processing (OpenMP) is used to improve the performance of the encryption algorithm applied.

B. Hypothesis

The Hypothesis essentially is a reiteration of the objectives of the assignment, which gracefully comes down to the idea that **the use of OpenMP here should demonstrate improvements in encryption and decryption processing time by passing as fewer messages possible between adjacent nodes and to the base station.** Which can be supported by the time taken to encrypt and decrypt a message before sending or after receiving, the number of messages passed throughout the network, the number of events occurred throughout the network and details of nodes involved in each of the events which includes the reference nodes and its adjacent nodes. These metrics are written onto a log file at the end of the program to help us evaluate the correctness of the hypothesis.

II. THEORETICAL ANALYSIS & IPC DESIGN

A. Technical illustration and description of IPC architecture.

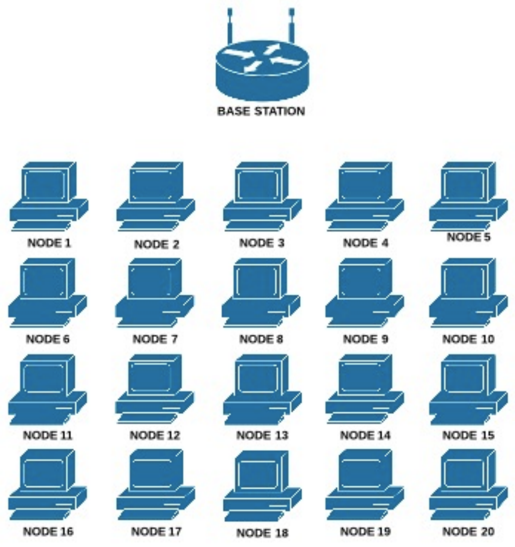


Figure 1. Wireless Sensor Network (WSN)

In the diagram above (see Figure 1), I have clearly illustrated the Wireless Sensor Network [1] that we will be utilising. The structure consists of 20 Nodes and a Base Station. These nodes are arranged in an orderly manner for the sake of simplifying the task at hand, in a 4x5 rectangular-shaped grid. The grid topology, or more commonly, the nearest neighbourhood method is used here due to its ability to "scale very well" and "tasks can be executed in parallel speeding performance. Grid environments are extremely well suited to run jobs that can be split into smaller chunks and run concurrently on many nodes. Using things like MPI will allow message passing to occur among compute resources." (Vassilios, T. 2007) [8]. In our Wireless Sensor Network (WSN), each node is an Message Passing Interface (MPI) Process which means that we have altogether 21 MPI Processes set up. Message Passing Interface (MPI) is the most suitable Inter Process Communication (IPC) that we can use for the Nodes to communicate between each other and from the Nodes to the Base Station because The MPI interface is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes (that have been mapped to nodes/servers/computer instances) in a language-independent way [4]. For every Node within the Grid, a generator assigns them a random number. This falls well within the criterion specified by the assignment specification mentioning that at any Node, if there are at least three or more nodes with the similar assigned random number, then an event is triggered. And this information will then be sent to the Base Station. Once the Base Station receives this information, it is written into a log text file. The Nodes are only able to communicate between *adjacent* Nodes amongst themselves and all the immediate neighbours in top-bottom and left-right directions are *adjacent* Nodes. Each node in the grid is both a reference node and an adjacent node at some point. Looking at Figure 1, if for instance we take Node '8' as our Reference Node, then simply the adjacent Nodes will be Nodes '3', '13', '7' and '9'.

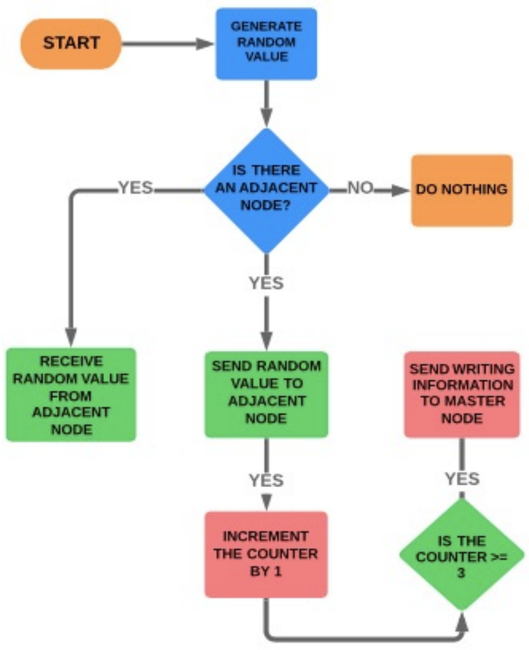

START
GENERATE RANDOM VALUE
IS THERE AN ADJACENT NODE?
YES
NO
DO NOTHING
YES
RECEIVE RANDOM VALUE FROM ADJACENT NODE
SEND RANDOM VALUE TO ADJACENT NODE
SEND WRITING INFORMATION TO MASTER NODE
YES
YES
INCREMENT THE COUNTER BY 1
IS THE COUNTER >= 3

Figure 2. Flowchart of IPC at Nodes.

In the flowchart above (see Figure 2), a brief illustration of the Inter Process Communication that a Node carries out is presented. To begin with, the Node is assigned a random generated value, which in our case we use 'rand()%5' to narrow down the range and therefore simultaneously increasing the chances of 3 or more adjacent nodes having the same random number which will trigger an event. After generating the number, we check to see with an 'if' and 'else' condition to see if there are any neighbouring Node in all four directions. If there is an adjacent Node present to the Reference Node, the Reference Node sends the random value to any and all neighbouring Node(s) using the MPI_Send() function and/or command and *simultaneously* increments the counter for each send request. This is very important to know if the number of adjacent Nodes are more than or equal to 3 which is the condition for us to then send the writing information such as 'communication time' and 'Timestamp' to the Base Station to be written into a log text file. The Reference Node also while performing all these tasks receives the random value from all adjacent Nodes via the MPI_Recv() function and/or command.

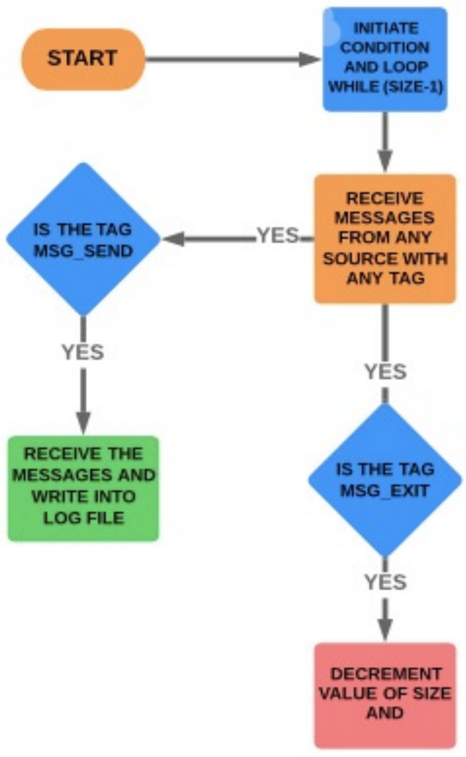

START
INITIATE CONDITION AND LOOP WHILE (SIZE-1)
RECEIVE MESSAGES FROM ANY SOURCE WITH ANY TAG
YES
IS THE TAG MSG_SEND
YES
RECEIVE THE MESSAGES AND WRITE INTO LOG FILE
YES
IS THE TAG MSG_EXIT
YES
DECREMENT VALUE OF SIZE AND

Figure 3. Flowchart of Base Station Procedure.

The Base Station is essentially an Universal Receiver. Whenever the criterion of event is met for a Reference Node, the relevant information required such as the number of reported messages, communication time, timestamp and adjacent Nodes are broadcasted to the Base Station to be written into a log file. This is made possible by using the MPI_Recv() function and/or command. This whole ecosystem is otherwise called the Master-Slave Concept. The Base Station acts as the Master whose tasks are to Receive and Write while the Nodes are to Send messages to the Base Station having met the criteria. The Base Station receives any information from any source with any tag, and a 'switch' is used to decide if to end the process or to write it into a log file. The messages are in the form of integers and double values.

B. Description of the Encryption/Decryption algorithm (OpenMP).

The encryption algorithm that we have adapted and implemented is the XOR Encryption. An XOR Cipher is a type of an additive cipher [9]. This makes it robust against brute-force based attacks. The concept of implementation is to first define XOR Encryption key and then to perform XOR operation of the characters in the String with this key which you want to encrypt. To decrypt the encrypted characters we have to perform XOR operation again with the defined key (see Figure 4).

```
// The same function is used to encrypt and
// decrypt
void encryptDecrypt(char inpString[])
{
    // Define XOR key
    // Any character value will work
    char xorKey = 'P';

    // calculate length of input string
    int len = strlen(inpString);

    // perform XOR operation of key
    // with every caracter in string
    for (int i = 0; i < len; i++)
    {
        inpString[i] = inpString[i] ^ xorKey;
        printf("%c",inpString[i]);
    }
}
```

Figure 4. XOR Encryption algorithm.

"The XOR operator is extremely common as a component in more complex ciphers. By itself, if the content of any message can be guessed or otherwise known then the key can be revealed" (Churchhouse, Robert. 2002). Its primary merit is that it is simple to implement, and that the XOR operation is computationally inexpensive. If the key is random and is at least as long as the message, the XOR cipher is much more secure than when there is key repetition within a message [10].

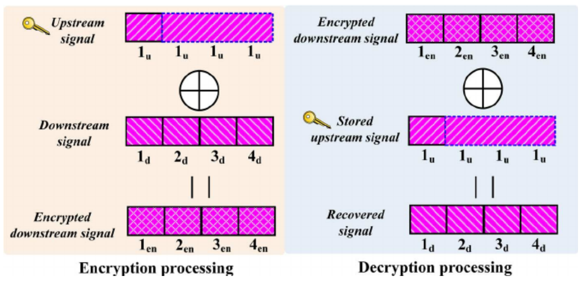
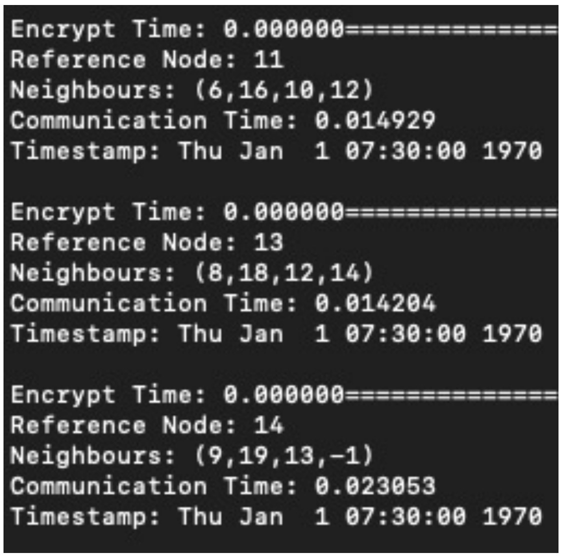


Figure 5. XOR Encryption/Decryption.

We carry out a parallel implementation of this cryptography algorithm onto multicore architectures by making use of OpenMP API. OpenMP is a programming interface that provides a scalable and portable model for shared memory parallel applications. Open MP uses a fork-join model for the execution of any program. All OpenMP program begins execution as a single thread of execution called the master thread. The master thread will execute in a single region until the first parallel construct is encountered. When a parallel construct is encountered then the master thread creates a team of parallel threads. The statements that are enclosed by the parallel region construct are then executed in parallel among the various team threads. After the execution of all the statements within the parallel region, team threads will terminate and leaving only the master thread.

III. RESULTS AND DISCUSSIONS

The Makefile containing all the code files were run twice. Once without encrypting the messages between adjacent nodes and the base station and once again with the Encryption/Decryption using Open Multi-processing (OpenMP). The results of the last 3 readings in the "log.txt" file were screenshot and presented in Figure 6 and Figure 7 below. The information that it contains are the Encryption Time (if there was Encryption involved), the Reference Node, the Adjacent Nodes, the Communication Time taken and the Timestamp.

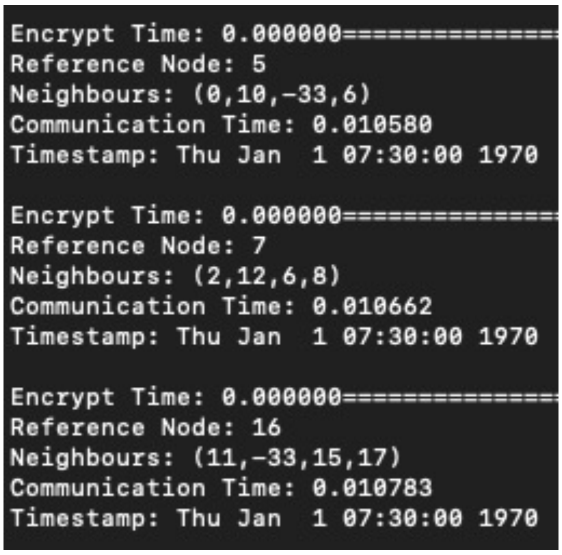


```
Encrypt Time: 0.000000==============
Reference Node: 11
Neighbours: (6,16,10,12)
Communication Time: 0.014929
Timestamp: Thu Jan  1 07:30:00 1970

Encrypt Time: 0.000000==============
Reference Node: 13
Neighbours: (8,18,12,14)
Communication Time: 0.014204
Timestamp: Thu Jan  1 07:30:00 1970

Encrypt Time: 0.000000==============
Reference Node: 14
Neighbours: (9,19,13,-1)
Communication Time: 0.023053
Timestamp: Thu Jan  1 07:30:00 1970
```

Figure 6. Last three entries in the log.txt file before Encryption of random values.



```
Encrypt Time: 0.000000==============
Reference Node: 5
Neighbours: (0,10,-33,6)
Communication Time: 0.010580
Timestamp: Thu Jan  1 07:30:00 1970

Encrypt Time: 0.000000==============
Reference Node: 7
Neighbours: (2,12,6,8)
Communication Time: 0.010662
Timestamp: Thu Jan  1 07:30:00 1970

Encrypt Time: 0.000000==============
Reference Node: 16
Neighbours: (11,-33,15,17)
Communication Time: 0.010783
Timestamp: Thu Jan  1 07:30:00 1970
```

Figure 6. Last three entries in the log.txt file after Encryption of random values.

The data from these screenshots were obtained and then tabulated into a table which consists of the serial time and parallel time (with OpenMP) and three readings were averaged to obtain a more reliable reading.

	Serial	Parallel	Speed Up (Serial/Parallel)
1st Run	0.014929	0.01058	1.4111
2nd Run	0.014204	0.010662	1.3322
3nd Run	0.023053	0.010783	2.1379
Average	0.017395	0.010675	1.5371

Figure 7. Table of Speed Up

The above table illustrates that when run in parallel with Encryption using Open MP, there is a significant Speed Up in the Communication time between the Nodes and the Base Station. This is credited to the reason that fewer messages are being passed between the adjacent nodes and the base station.

The reason for this assignment was to test the use of OpenMP here, which should demonstrate improvements in encryption and decryption processing time by passing as fewer messages possible between adjacent nodes and to the base station. This was realised in this report by the above results and the evidences and explanation that followed. Therefore, it is with a happy heart that I would like to confirm that the Hypothesis stands and has been proven to be valid.

IV. CONCLUSION AND FUTURE WORK

Event detection in a fully distributed Wireless Sensor Network (WSN) essentially possesses multiple real world applications and this brings us to improvise this Network System by proposing a method to parallelise this process by using Message Passing Interface and the use of OpenMP. This would theoretically provide a significant speed up that any implementation of a fully distributed Wireless Sensor Network (WSN) would perform flawlessly and have the best interest of its purpose for implementation. This idea was realised in this paper. The reason for this assignment was to test the use of OpenMP here, which should demonstrate improvements in encryption and decryption processing time by passing as fewer messages possible between adjacent nodes and to the base station. This was realised in this report by the above results and the evidences and explanation that followed. Therefore, it is with a happy heart that I would like to confirm that the Hypothesis stands and has been proven to be valid. This opens up the possibility of future works as due to the limitations in computing power the full potential of using OpenMP could be improvised even further.

REFERENCES

[1] L.F.Akyildiz,W.Su,Y.Sankarasubramaniam,andE.Cayirci,“ Asurvey on sensor networks,” *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114. vol. 25, no. 4, pp. 114–124, 2002.

[2] "Windows/WSL Interop with AF_UNIX". Microsoft Corporation. Retrieved 25 May2018.

[3] Crovella, M. Bianchini, R. LeBlanc, T. Markatos, E. Wisniewski, R. Using communication-to-computation ratio in parallel program designand performance prediction 1–4 December 1992.

[4] Walker DW (August 1992). Standards for message-passing in a distributed memory environment (Report). Oak Ridge National Lab., TN (United States), Center for Research on Parallel Computing (CRPC).

[5] Dargie, W. and Poellabauer, C. (2010). *Fundamentals of wireless sensor networks: theory and practice*. John Wiley and Sons. pp. 168–183, 191–192.

[6] Bellare, Mihir. "Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements." Springer Berlin Heidelberg, 2000.

[7] Costa, J.J.; et al. (May 2006). "Running OpenMP applications efficiently on an everything-shared SDSM". *Journal of Parallel and Distributed Computing*. 66 (5): 647–658.

[8] Vassilios, T. (2007). “Grid Computing – The Advantages.” https://it.toolbox.com/blogs/outervillage/grid-computing-advantages-and-disadvantages-040908

[9] Churchhouse, Robert (2002), Codes and Ciphers: Julius Caesar, the Enigma and the Internet, Cambridge: Cambridge University Press.

[10] Tutte, W. T. (19 June 1998), Fish and I (PDF), retrieved 7 October 2010 Transcript of a lecture given by Prof. Tutte at the University of Waterloo.