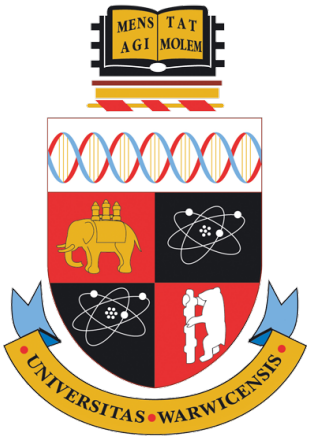
WARWICK
THE UNIVERSITY OF WARWICK



Towards Accurate Predicate Detection in Wireless Sensor Networks

by

Alwaleed Alharbi

Thesis

Submitted to the University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

March 2021

Contents

List of Tables

List of Figures

Acknowledgments

I owe a debt of gratitude to my supervisor, Dr. Arshad Jhumka, for providing me with the opportunity to conduct research at Warwick University and for providing me with invaluable support, feedback, and inspiration during my studies.For me, Arshad has been an excellent supervisor. During the study, he has taught me a lot. This work could not have been finished if it hadn't been for his support.

I would also like to express my gratitude to the majority of the Computer Science department's staff for their assistance and guidance during my studies.

Finally, I would like to extend my gratitude to my parents and family for their extreme love, prayers, motivation, and assistance.

Declarations

I certify that no part of this thesis has been published elsewhere or submitted for a degree at another university.

Alwaleed Alharbi

Abstract

Wireless sensor networks (WSNs) have enabled a large class of novel applications such as environment monitoring and tracking. The size and cost of the WSN devices have made it that such devices can be deployed on a large scale. However, WSNs are often deployed in harsh environments where the network may not be reliable. As such, applications may not run as intended, thereby requiring run-time monitoring to determine when applications are deviating from their intended behaviour. The monitoring of the system takes the form of detecting whether a given predicate holds true during the execution of the system, where the predicate captures a notion of correctness of the system, i.e., there is a need to monitor when a predicate turns true or false.

However, the process of monitoring and debugging wireless sensor networks has many challenges. One of the challenges is that the network may not be reliable. Another challenge is the resource constraints imposed on the devices such as energy. Another one of the major challenges is that overhead and the size of the monitoring system must both be kept to a minimum. The monitoring system should not interfere with any running application that needs to be monitored since the monitoring system needs to run independently. Many monitoring systems implement various methods for debugging their applications; all techniques invariably contain a sensor data collection step at monitors. Typically, it has either been assumed that the network sink is the monitor or every node is a monitor. However, in this thesis, we address the problem of monitor selection and provide a novel formalisation of the problem.

Adopting a different perspective of the predicate detection problem, we refactor the problem into two important problems: (i) monitor selection problem and (ii) predicate detection problem. As such, the main works performed in this research are: (i) we show that the problem of Monitor Selection Problem (MSP) is NP-complete and (ii) to circumvent the inherent complexity associated with monitoring, we develop a number of heuristics to support data collection for debugging. The application that will be used to showcase the framework is a TDMA slot assignment. We define the predicate that captures the correctness for TDMA and use the heuristics to monitor it. Four main contributions are presented. First, a novel formalisation of monitor selection model is presented, allowing for comparison of the existing and different approaches. Second, once the monitor selection problem has been formalized, a heuristic is presented and developed for WSN to circumvent the complexity of selecting monitors. Third, different frameworks are presented with multiple algorithms in WSN that disseminate messages throughout the network. Finally, extensive simulation and real hardware (i.e., testbed) experiments are conducted in order to investigate the overall performance and accuracy of each implemented infrastructure and algorithm. Our results show that our protocols can outperform state-of-the-art protocols for predicate detection at much lower overhead.

Sponsorships and Grants

The research presented in this thesis was made possible by the support of the following benefactors and sources:

- Taif University, Saudi Arabia.
- Saudi Arabian Cultural Bureau in the U.K.

Acronyms

6LoWPAN IPv6-based Low Power Wireless Personal Area Networks.

BS Base Station.

CH Cluster Head.

CP Constraint Programming.

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance.

CTP Collection Tree Protocol.

DAS Data Aggregation Scheme.

GPS Global Positioning System.

ILP Integer Linear Programming.

IoT Internet of Things.

IP Internet Protocol.

MAC Medium Access Control.

MPS Monitor Selection Problem.

RAM Random-access Memory.

RFID Radio-Frequency Identification.

SPIN Sensor Protocols for Information via Negotiation.

TDMA Time-Division Multiple Access.

WSN Wireless Sensor Networks.

Symbols

α	The alpha symbol
ϕ	Predicates
δ	Influence Diameter
V'	Monitors
V	Nodes
u, u'	Communication links between nodes

Chapter 1

Introduction

WSN is a network consisting of hundreds or thousands of sensor nodes which are distributed in a particular area to monitor environmental conditions, such as temperature, sound, pressure and other sensing measurements [8]. There are numerous areas of application of WSNs, including habitat surveillance [102], healthcare[104] and environmental monitoring[114]. WSN nodes have resource constraints imposed on them and the constraints typically are energy, CPU and memory. Each node has a wireless communication interface and thus, WSN nodes communicate with each other and pass information along, from one to another, to connect to at least one base station (or sink).

Given the harsh environments in which WSNs are typically deployed, failures of the network nodes are typical, rather than being the exception due to resource exhaustion, memory corruption due to adverse environmental conditions. To ensure correct operation or conditions of the network, including the applications running on it, the network needs to be monitored to ensure that the requirements of the applications are being met. Monitoring the sensor network while operating ensure failure detection before it occurs, also enable remote user to debug and remotely control the sensor node while application is running.

Typically, these correctness conditions are specified as (*predicates*) that need to be evaluated over the state of the network. Such predicates, when evaluated over the nodes of the network, are called global predicates. On the other hand, when the predicates are evaluated over a small subset of nodes that are co-located, then this type of predicates is called local predicates, as the predicates are evaluated over the states of nodes local to a region. There are two monitor placement alternatives, global placement at the sink node or local at some local cluster within the network. For global placement monitor node

can be far away from where the predicate detection algorithm is operated. It means that monitor nodes need to be positioned at certain locations in the network to correctly detect errors at low cost, Hence, the other alternative is local where the monitor nodes are placed in a subset local nodes within the network. For the global placement of the monitor node there are only unique node monitor which monitor all the system while for the local monitor there are several monitor nodes distributed in the network. In a traditional distributed system, monitors are nodes dedicated to monitoring when these predicates turn true or not. This is achieved by requiring all the nodes in the distributed system to send control messages about their states to these monitors, which then assemble them to evaluate the predicates. However, in a WSN, sending all control messages to a monitor will cause energy to be depleted. Thus, a careful selection of monitors in a WSN is very important to monitor predicates. This is the focus of the Thesis, where we analyse the problem of selecting monitors so that (i) the monitoring accuracy is high and (ii) the overhead is reduced.

1.1 Motivation

Debugging is one of the most important techniques for determining software errors (bugs). Debugging is especially important in wireless sensor networks (WSNs), which are prone to unpredictably changing runtime conditions. As such, WSN programmers use techniques such as simulators, safe code enforcement, and formal testing before deploying an application in the field. However, testing phase of all conditions in the lab is unrealistic since WSNs are implemented in harsh environments whose behaviour cannot be easily replicated in a laboratory. Debugging is frequently done in a cyclical process of continually running a programme and checking for defects. In WSNs, cyclic debugging can be time-consuming and labor-intensive:

- Because nodes are not always immediately accessible, the programmer must rely on low-power wireless networks to capture any data of importance.

- Because there may not be enough information to detect a malfunction right away, the network must be wirelessly rebuilt with code to gather extra debugging information. This can take many minutes, as might waiting for the bug to reappear. Thus, it's critical to have a monitoring system that can detect errors while the system is running.

- Some WSN applications/protocols do not require information from the entire network to evaluate the correctness of their behaviour. Users are

more interested in checking the status of these applications in some part of the network, such as the status of neighbouring nodes.

Because sensor networks have limited resources, the monitoring and detection overhead must be decreased. When debugging techniques are implemented in WSN, the delay factor and message response is another important because such messages can be lost or the monitors do not receive a response. Complex and cross-layer interactions in multi-hop WSNs require a complete stochastic characterization of the delay and ensure the message requested is received. Because of the randomness of wireless communication and the low power nature of the communication links in WSNs, average delay or worst-case delay provides limited insight into the operations of WSNs. Bugs happen by non-deterministic proceedings such as reboot of nodes, duplicates of packet or loss of packet [106]. Thus, a protocol may not execute as intended due to network uncertainty. There are predicates that capture the protocol's correctness. These predicates can be inserted into the programme or evaluated while it is running. If the predicate is false, the algorithm isn't working correctly. WSNs are often used in mission-critical systems, hence they need function properly.

The network monitors are in responsible of collecting and evaluating those conditions. Predicates are usually evaluated by the sink or all nodes when monitoring WSN applications. This resulted in another issue, including the network monitor selection. For instance, the base station of the network may use detection algorithms to diagnose whether a sensor malfunctions by comparing data from surrounding sensors or subset of nodes. The sink is usually considered to be the monitor for the entire network in most WSN applications, meaning that the sink has the overall view of the entire network. This technique is called global evaluation. Although the predicate evaluation is applied to only one node, it necessitates the use of more packets in order to keep the overall view at the sink. The term "global evaluation" refers to a centralised monitoring system that can keep track on the entire network. Sending more data to the sink can result in lost messages or data collisions with the requested data. The other local technique is to designate a subset of nodes as monitors, so that predicate evaluation is processed locally in a subset of nodes, ensuring low latency and accurate evaluation. Local predicate evaluation, on the other hand, requires a large number of messages and overlapping monitoring between neighbour nodes. Also packet collision is most probably occurs due to huge amount of messages. By selecting more than one monitor in the network, the local technique can potentially evaluate the correctness of a subset of nodes with low overhead. This will reduce message redundancy and eliminate the redundant messages in the network. Even if the desired monitors selection is

implemented, appropriate data dissemination is critical for more efficient data propagation. Therefore, a careful selection of monitors in a WSN is important for monitoring predicates as well as dissemination protocols.

1.2 Monitoring Predicates Challenges

In WSN, predicates capture properties that must hold in the network, either at individual nodes or at an aggregate level. Temporal operators provide connections between states that occur at various times. The monitoring of such temporal predicates would require recording the evolution of the environment over time, which would be impossible due to the memory restrictions of normal WSN nodes. Furthermore, a predicate violation happens in the environment when a specific combination of physical quantity values occurs at the same time. In practise, however, these physical values are only available to any software monitoring system when they are detected through some device (e.g., a monitor) and saved as program data. Our objective is to create a solution that detects predicate violations effectively, based on changes in predicate values at several node if they occur.

According to the statement above, monitoring predicate violations is an example of distributed predicate detection[50]. A centralised solution that maintains global information about the system state suffices to solve the problem in a fully synchronised system with unlimited communication and processing capabilities[50]. In the absence of particular assumptions about system latencies, however, (absolute correctness) a system that captures predicates violations if and only if they occur is difficult to achieve. WSNs are non-deterministic systems with limited communication and processing resources. Thus, monitoring WSNs raises a number of challenges, the more important being:

- **Maintaining accurate global information**: Capturing and maintaining accurate global state knowledge is extremely costly, for example in terms of communication energy, because each change of state at any node in the network may need transmission to a central unit. Also, unlike assumptions in distributed systems, links in WSNs are often not reliable, making capturing global state a real challenge.

- **Network faults**: Because WSNs are typically asynchronous in nature, network failures, such as state corruptions, are difficult to predict. This not only risks missing predicate violations, but it also makes it difficult to determine when a predicate has been violated.

- **Detection latency** In addition to the need of accurately capturing

violations, another factor that prevents the system from detecting all violations in a timely manner is latency. This is often affected by the monitor(s) that are selected in the network. Often, the sink is used as a monitor. However, the long distances travelled by state data will impose a latency on the detection process.

1.3 Properties of monitoring system

The system applied for the monitoring should be light weighted and consume low energy. The memory and overhead transmission for the monitoring system should be also very low, with these all requirement the system must not alter the application running. The efficient and robustness implementation of the monitoring system can improve the WSN operation and enhance the life time of the WSN. The proper dissemination of the monitoring system messages reduce message collision and overhead. The monitoring system should be not affected by the network size and must be applicable to adaptive network size.

1.4 Contributions

To detect the predicates in WSN applications, we have identified several challenges.

- To address the challenge of capturing accurate global state information, we conjecture that (i) data needs to travel over short distances only to the monitor and (ii) data may need to be sent either proactively or reactively depending on the requirement of the application. As such, we observe that predicates have "radius of influence" and we identify a novel problem called Monitor Selection Problem (MSP), which we address in Chapter 5. Also, the MSP problem helps address the latency challenge, in that data does not have to travel over long distance when aggregating state information, thereby reducing detection latency.

- Chapter 6 introduces a number of dissemination protocols for data collection at the monitors. They differ in terms of the properties they have and enforce, e.g., one algorithm may send data periodically while another may send data upon request. Those algorithms help address challenges 1-3.

- In Chapter 7, we address issues of link failures (challenge 2) and also of scalability. By executing the algorithms on actualt HW testbed, we allow for the link to vary in a natural way, e.g., due to external interference. Different parameters and settings are evaluated and tested on various

network topology and sizes.

1.5 Thesis Organisation

This chapter describes the contributions that this thesis makes, as well as the thesis's introduction and organization. The rest of the Thesis is as follows:

- Chapter 2 presents a collection of related work and background.
- Chapter 3 presents system structure and simulation setup.
- Chapter 4 discusses the architecture of the network and the monitors selection proposed by describing the MSP in WSN.The chapter also outlines the predicates implemented which defines the predicate types that are targeted for.
- Chapter 5 outlines the model's development as well as the proposed heuristic approach for monitor selection. The techniques and methods for implementing a heuristic algorithm in a wireless sensor network are presented in this chapter including experiments and comparisons of other existing approach.
- In Chapter 6, different distribution and dissemination algorithms are presented.
- Chapter 7 conducts simulations and real-world testbed experiments.
- Chapter 8 discusses the implications of this work.
- Finally, Chapter 9 presents concluding remarks and future works.

Chapter 2

Background and Literature Review

2.1 Background

2.1.1 Wireless Sensor Networks

Wireless sensor network, or WSN, is a collection of hardware devices called motes; these devices capable of computing and communicating with each other through a short range of wireless. These motes have sensor hardware on them to allow them to sense some of the environment around them [64]. These motes are capable of forming a distributed system capable of conducting several algorithms, such as data collection, data transmission and other tasks. Each of these motes is equipped with a radio that allows communication within a limited range between them. In order to sense the surrounding environment, mote may equip one of the embedded sensors, such as light, temperature and others(see Figure 2.1). These devices are equipped with a micro central processing (CPU) where they have collection of programs to monitor the hardware on them (see Figure 2.2). The Processor can also handle the events during the received or sent data. The CPU can control and manage much of the operating system overall.Since the motes are operating on a wireless network, the power supply is limited and a small battery is supplied.Today, there are many applications and systems using this type of wireless platform, such as military applications, to conduct intelligence and information in the battlefield [126] [8] [74] and in the industrial field where factories and companies track their data [28, 55, 153].

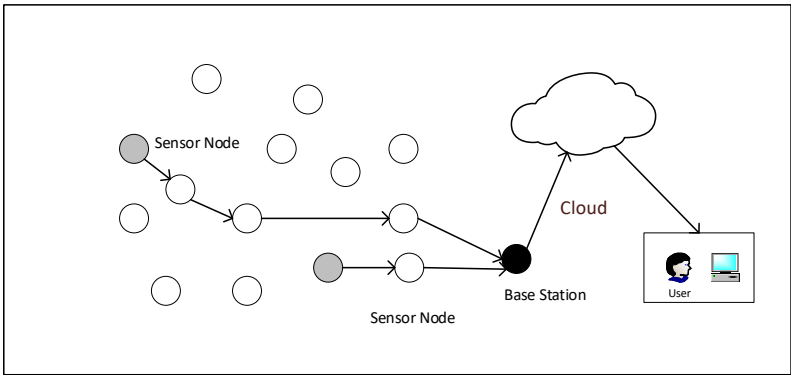


Figure 2.1: Demonstration of wireless sensor networks

Limited and finite energy supply for each node is the most critical aspect of designing an application in wireless sensor network. In wireless sensors, thus, costly broadcast protocols such as IEEE 802.11 are avoided[64], but instead use a lot simpler energy-saving protocol. Wireless protocols such as IEEE 802.15.4 ZigBee, for example, are designed for use of wireless sensor networks and are correlated with lower energy usage [27, 81]. Some applications depend on actions of even lower levels defined by a certain MAC layer [20, 25, 120],These implementations require a trade-off between development time and energy consumption, in which flexibility is often sacrificed for reduced energy use.Unfortunately, using these basic protocols has the drawback that transmissions are susceptible to many kinds of collisions and loses of messages, so it is really critical that the program running on the nodes is built to manage those situations.Having battery consumed means that the design of Wireless Sensor Networks implementations is essentially limited to optimizing the lifetime of the network, so that the greatest usefulness can be obtained from the network [110].

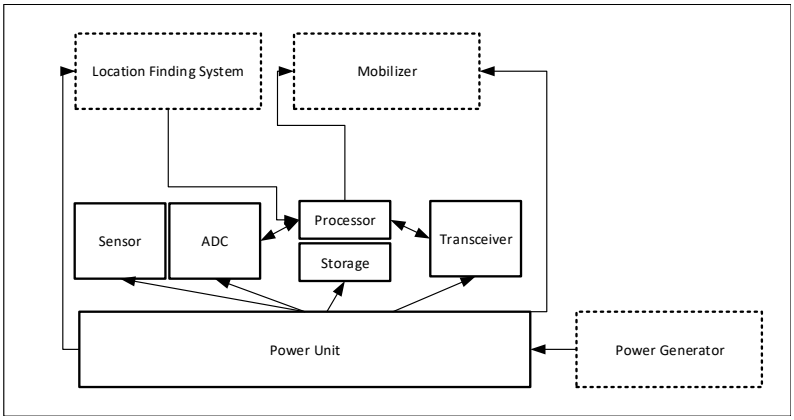


Figure 2.2: The components of a sensor node [8]

As wireless sensor nodes run in rugged and harsh conditions in outdoors platform [134, 144], there is a high risk they will fail. Such faults can range from damage to hardware caused by environmental conditions or interference, program glitches, or just a lack of service triggered by power-out nodes. It is therefore important to develop algorithms and applications to tackle these possible errors, or they face catastrophic failure when they experience these problems.

Wireless sensor nodes are designed to run in remote and usually unattainable locations with no human intervention during their operational life [85]. In this regard, a distinctive feature of a WSN system is that all applications built for it must be self-configuring in nature; that is, the system must be controlled by itself and the network without external input [126].Obviously, solutions to these issues have fall hand and hand with the topic of network robustness and tolerance to faults.

Although the prevailing characteristics of a wireless sensor node are a restricted energy supply, self-configuration and network robustness, there are various other features or problems that can be considered.These nodes may be mobile, for example (examples include an ad-hoc PDA network or motes built into soldiers' helmets) [108], this can contribute to very curious actions in the handling of communication between these nodes.

2.1.2 Communication and data dissemination

Traditionally, TCP / IP sensor networks are considered as unsuitable;Estrin et al. indicated that sensor networks have such different criteria for traditional networks that they need to reconsider the basic configuration of these networks [47]. Some of the IP related problems Estrin et al.in sensor networks are [47]:

- "The sheer numbers of these devices, and their unattended deployment, will preclude reliance on a broadcast communication or the configuration currently needed to deploy and operate networked devices."
- "Unlike traditional networks, a sensor node may not need an identity (e.g., an address)."
- "Traditional networks are designed to accommodate a wide range of applications."

Hui and Culler indicated that these major issues are due in large part to IPv4 and not to the new IPv6, stating "IPv6 is better suited to the needs of WSNs than IPv4 in every dimension"[69]. Further demonstrating that a network

architecture based on IPv6 could be used in sensor networks,and provides a solid basis from which to base further work [42].Simplified versions as well as optimisations with headers have been suggested [37, 38],however there were still problems with their being IPv4 based. Estrin et al.'s implementation of an IPv6-based architecture still presented some issues.Large headers and the dependency on routing tables for example is one of these issues,after all, the architecture has certain advantages with respect to neighbour discovery and routing applications, but little change was given to applications that did not require identities (flooding for instance).

Dunkels et al. designed an architecture that allow more code reuse and protocol interoperability, this includes the Rime communication stack. Applications having to operate over several protocols created the need for this new layer, this is mainly because it does not rely on a certain mechanism of communication (for example, acknowledgments and re-transmissions)[40]. The previous implementation [45, 121] did not specifically discuss and overcome problems with protocols operating on the top lower layer, although the Rime stack used in Contiki was presented as a basis [40].

The solution is presented in the Rime communication stack where the cross-layer information-sharing problem of a layered communication stack separates the logic of the protocol from the packet header details. To address this issue, the packet headers needed to be sufficiently adaptable to cover all types of communications and small enough. They solved this problem in the Rime Communication Stack using the notion of Packet Attributes. To do this, the communication modules generated to migrate the attributes and data packets into the required packets (including headers and payloads) [40].This allows developers the benefits of obtaining information at a lower level without the conflict with layering concepts, at the end of which there are no performance changes.It further facilitates interoperability with all protocols, which abstract the networking layers from the main application layer.As such, Rime Communication stack includes several primitives, for example Single-Hop Broadcasting, in the application layer [40]. The collection of these primitives was focused on review of different use cases in WSNs. Figure 2.3 shows communication primitives in the Rime network stack, whereas tables 2.1 and 2.2 detail the mechanisms and protocols implemented in the thesis.

Name	Header	Address
Network Flooding	netflood.h	http://contiki.sf.net/docs/2.6/a01728.html
Unicast	unicast.h	http://contiki.sf.net/docs/2.6/a01738.html
Broadcast	broadcast.h	http://contiki.sf.net/docs/2.6/a01720.html
Multi-hop Unicast	multihop.h	http://contiki.sf.net/docs/2.6/a01726.html
Polite Broadcast	ipolite.h	http://contiki.sf.net/docs/2.6/a01724.html
Reliable Multi-hop Unicas	rmh.h	http://contiki.sf.net/docs/2.6/a01732.html
Reliable Unicast	runicast.h	http://contiki.sf.net/docs/2.6/a01738.html
Reliable Unicast Bulk Transfer	rucb.h	http://contiki.sf.net/docs/2.6/a00365.html
Anonymous Broadcast	abc.h	http://contiki.sf.net/docs/2.6/a01717.html
Anonymous Polite Broadcast	polite.h	http://contiki.sf.net/docs/2.6/a01730.html
Stubborn Broadcas	stbroadcast.h	http://contiki.sf.net/docs/2.6/a01739.html
Stubborn Unicast	stunicast.h	http://contiki.sf.net/docs/2.6/a01740.html
Neighbor discovery	neighbor-discovery.h	http://contiki.sourceforge.net/docs/2.6/a01727.html
Trickle	trickle.h	http://contiki.sf.net/docs/2.6/a01742.html
Collect	collect.h	http://contiki.sf.net/docs/2.6/a01723.html
Mesh	mesh.h	http://contiki.sf.net/docs/2.6/a01725.html
Net-Flood	netflood.h	http://contiki.sourceforge.net/docs/2.6/a01728.html

Table 2.1: Rime Communication Primitives, documentation location, and headers

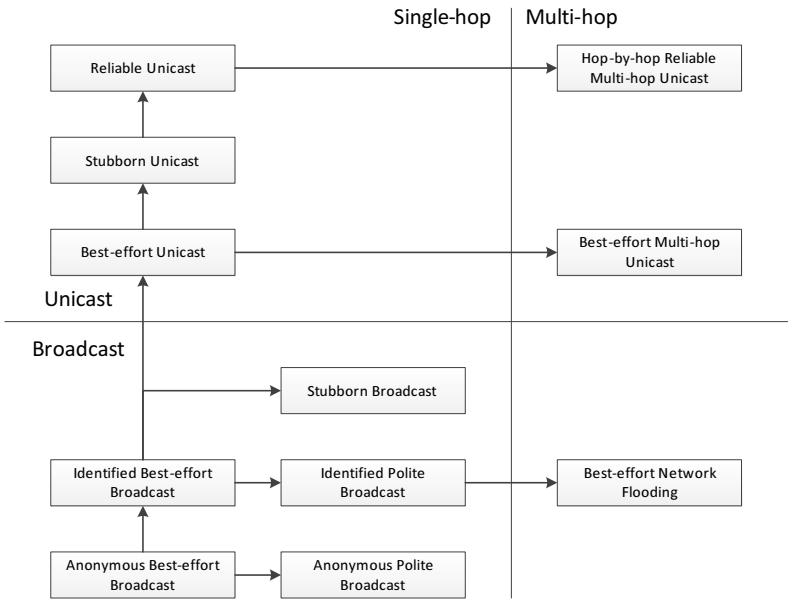


Figure 2.3: This graph demonstrate communication primitives in Rime network stack [40]

Name	Reliable	Target	Sender Known
Anonymous Broadcast	No	1-hop neighbours	No
Broadcast	No	1-hop neighbours	Yes
Stubborn Broadcast	No	1-hop neighbours	No
Anonymous Polite Broadcast	No	1-hop neighbours	No
Polite Broadcast	No	1-hop neighbours	Yes
Unicast	No	destination	Yes
Stubborn Unicast	No	destination	Yes
Reliable Unicast	Yes	destination	Yes
Network Flooding	No	network	Yes
Multi-hop Unicast	No	destination	Yes
Reliable Multi-hop Unicast	Yes	destination	Yes
Mesh	No	destination	Yes
Collect	Yes	destination	Yes
Trickle	Yes	network	No
Net-Flood	Yes	network	Yes

Table 2.2: Rime Communication Primitives Mechanism

Tree Aggregation

Tree aggregation is a hierarchical routing protocol that, unlike clustering, divides a network into a tree structure, with the root node at the base station.The nodes that are the farthest away from the base station (in hops) become the tree's leaves, while nodes in the middle become branches.Using the tree structure, leaves and branches forward messages to the base station and across the network, leading to a lower number of global network messages than other algorithms like the flooding protocol. By adding an aggregation or compression mechanism at each branch node of the network, the message count may be further minimized, compared to clustering.The disadvantages of tree aggregation are strikingly similar to those of clustering: i) The burden on branch nodes is greater than on leaf nodes. ii). If the battery in a branch node runs out and the node shuts off, it will isolate a whole portion of the network from the base station. TAG, MFS, and DCTC are some of the most well-known tree aggregation algorithms [96]. They all take different approaches to organizing the data aggregation mechanism of the branch nodes in order to minimize the number of collisions and therefore messages in the network, despite their similarities. TAG, for instance, employs the concept of a periodic routing protocol separated into time periods for each layer of the tree, ensuring that each branch receives as many messages as possible from its children before moving on to the next layer. It is worth mentioning In this study[56], tree aggregation was implemented, and the study contrasted this data propagation against the Trickle [94]mechanism to determine where the lowest cost and fastest detection when detecting faults in the application.

Flooding and Gossiping Communication

Flooding and Gossiping are two of the most basic routing algorithms accessible to a developer of a wireless sensor network. During flooding, wireless sensor network nodes transmit any messages they receive to all of their neighbors.Messages only come to a halt brodcasting When one of the following occurs:i) the packet is sent to the intended receiver; or ii) the maximum number of node hops for a message has been exceeded. In certain ways, gossip is the same as flooding; the only distinction is that instead of broadcasting a message to any of its neighbors, a node selects one neighbor at random to forward the message to.There is no need for sophisticated routing algorithms or topology management because Flooding and Gossiping are simple [7].Nevertheless, these two routing algorithms have a number of flaws, including: If the same message is delivered to the same receiver twice, it is called *Implosion*. Gossip prevents this, but it induces delays in message transmission across the network.Another problem is *Overlap*, which occurs when two nodes detect the same change in their local area and send identical messages to the same neighbor.Another flaw in these routing protocols is *Resource Blindness*, which occurs when a routing algorithm uses a large number of resources without consideration for performance [7].SPIN (Sensor Protocols for Information via Negotiation) is a set of routing algorithms that aims to solve the problems that standard Flooding and Gossiping protocols have [7, 9].The set algorithms accomplishes this through data negotiation and resource-adaptive algorithms. For instance, SPIN nodes have access to their current battery level and use this data to adjust the protocol based on how much energy is left.Furthermore, the SPIN protocol family employs the idea of meta-data to minimize the amount of redundant data transmitted through the network.

Trickle

With a simple broadcast, information can be disseminated efficiently via a single-hop network. Data propagation gets more difficult as the network topology grows more complex with multihop and non-uniform physical deployment, and the complexity increases considerably with the number of hops in the network, especially for low-power and lossy networks (LLN). Iterative unicast, recursive broadcast (a.k.a. flooding), and the Trickle timer algorithm [94] are the three essential building elements for data propagation over a multihop wireless network. These primitives provide the foundation for many network protocols.Iterative unicast provides the advantage of being able to monitor each node's progress and provide end-to-end reliability. Iterative *unicast* is extensively utilized in many high-end devices because of this. It does not, however, scale well.Iterative *unicast*, for example, takes at least N transmissions

to forward a message to N 1-hop neighbors, whereas broadcast just requires one.Iterative unicast requires $\Omega(NlogN)$ broadcasts for a binary tree-like h-hop multihop network with 2^h nodes, whereas recursive broadcast (described below) requires Ω(N), not to mention substantially higher network-wide delay.

Because of its simplicity, recursive broadcast is extensively utilized in resource-constrained embedded systems. It is ideal that iterative unicast be used for dissemination because it is substantially faster.To minimize broadcast-storm problems, a node normally uses a random temporal jitter within a predetermined range to mitigate synchronized broadcasts and collisions. It also ignores previously-heard packets. Increasing the time jitter range and communicating slowly can reduce collisions and improve reliability, but at the cost of higher latency. While (re)transmitting multiple identical packets can improve reliability, it wastes bandwidth and energy on the channel. As the configuration changes, the recursive broadcast approach makes an explicit trade-off between reliability, latency, and overhead[72].

For multihop networks, the Trickle algorithm[94] is meant to address the challenges mentioned previously. To limit the number of duplicated packets and collisions, it uses two techniques: halving with doubled intervals and suppression. Unless it detects network instability, Trickle increases the transmission interval at the end of each transmission to decrease channel utilization and collision probability. Furthermore, it divides each period in half and selects the transmission time at random from the second half of the interval. It listens for duplicate packets and counts them till the timer goes off.If the duplication counter hits a particular threshold during transmission, Trickle suppresses the transmission. Trickle offers reliable, responsive, and energy-efficient distribution over a multihop network with minimum overhead using these basic techniques. Trickle has been widely used in various standard protocols and IoT applications, including the IPv6 Routing Protocol for LLN (RPL) [79], Multicast Protocol for LLN (MPL) [67], OTA programming for wireless sensor networks (WSN) (e.g. Deluge [68]), and task propagation in WSN (e.g. Maté [92], Tenet[116].

2.1.3 Simulators

A WSN consists of a large number of autonomous sensing nodes. An analysis of a WSN generally leads to over simplified assessment with restricted confidence. On other hand, deploying testbeds requires a vast effort. Therefore, a simulation tool is useful for studying a WSN environment and its characteristics. However, results of simulation are based on a specific scenario undera different study and topology, hardware and physical layer assumptions, which are not

generally accurate enough to explain the complex behavior of a WSN. This will lead to reduced accuracy of simulation results. In order to have good simulator, there should be a correct balance between scalability and accuracy factors of a WSN. According to Guo and Shuo [57] simulation can be classified into general simulation packages and a specific WSN framework environment.

We will now discuss some of the simulators which are provided in a general simulation package. Network Simulator version 2 (NS-2) [103] is one of the most popular non-specific network simulators and it is an open source simulator, supporting a wide range of protocols in all layers. Objective Modular Network Testbed in C++ (OMNET++) [140] is open source and it provides a powerful GUI library for animation, tracing and debugging support as well. It doesn't support a larger protocols library as compared to NS2. Global Mobile Information System Simulator (GloMoSim) [15] is a simulation environment for wireless networks designed using the parallel discrete event simulation capability provided by Parsec. Egea-Lopez et al. [44] have listed the most important and popular simulation tools for WSNs.

We will now discuss some of the simulators which are provided under specific WSN framework simulation packages. EmStar [52] is an event driven system consisting of a library that implements message-passing IPC primitives and frameworks that enable fault tolerance and field preparation. Mannasim [105] is a WSN simulation environment comprised of two solutions: the Mannasim Framework and the Script Generator Tool. TOSSIM [93] is a bit-level discrete event simulator and emulator of TinyOS [91]. TOSSIM is a library whereby you must write a program that configures a simulation and runs it. TOSSIM supports two programming interfaces: Python and C++.

2.1.4 Emulators

WWSN application involves a sensor node, its drivers, operating systems and networking protocols. The performance of a WSN application is measured based on these matrixes as well its implementation. Emulator is a special type of simulator; it might be hardware or software. The aims of Emulator are to allow realistic performance evaluation for WSN applications. Emulators are good choice since it can be run directly for testing, debugging and performance evaluation WSN applications. We will now mention some of Emulators. ATEMU [122] is a software emulator for AVR processor-based systems. It also includes support for other peripheral devices on the MICA2 sensor node platform such as the radio. ATEMU is intended to cover the gap between

actual sensor network deployments and sensor network simulations. MSPSim [71] is a Java-based level emulator of the MSP430 series microprocessor and emulation of some sensor networking platforms. MSPSim supports loading of IHEX and ELF firmware files. MEADOWS [137] is a software framework built at HKUST (The Hong Kong University of Science and Technology) for modeling, emulation and analysis of data of WSNs.

2.1.5 Testbeds

Other tools used for evaluating large sensor networks are called testbeds. These can offer understanding of resource limitations, communication loss and energy constraints at scale. Testbeds are an environment that provides support for measuring a number of physical parameters in a controlled and reliable environment. This environment includes the hardware, instrumentations, simulators and various software to conduct an experiment. In general, testbeds allow for accurate results, along with frank and replicable experiments. Here we will review some of the testbeds used in WSN applications. Motelab [143] is a public testbed and allows users to upload executables, receive execution results via the Internet and the creation and scheduling of experiments. MoteLab is a set of software tools for managing a testbed of Ethernet connected sensor network nodes. Kansei [46] provides a testbed infrastructure to conduct experiments on various wireless platforms, including 802.11, 802.15.4 and 900 MHz Chipcon CC1000 radios, as well as diverse sensor node platforms, including XSM, TelosB, Imote2 and Stargates.

2.1.6 Debugging Distributed Systems

The architecture of a distributed system is perceived to be an especially difficult task, rather than a conventional one.There are a number of explanations for this. First, several processes must be carried out in parallel within a distributed system.As a result, variables can be updated independently or in response to other processes that could lead to a myriad of synchronization and timing issues that the developer must address.Second, traditional programming languages are not suitable for the development of distributed programmes [107, 141].There's the opportunity for errors in any system built by humans, software or otherwise. Errors may be innocuous or cause unwanted actions and system failures. An extremely critical aspect of any toolchain is to build software to identify these *bugs* and alert the developer so they can be patched or fixed. The GNU toolchain for instance has toolchain such as gdb [132],this helps developers to put breakpoints in code to interrupt the execution of the program in that state, so that it can be tested and checked.There are also several methods that search into memory problems (such as valgrind [128]) [23, 111], security defects

[48, 142] and a variety of other types of faults.

It is a difficult challenge to implement distributed systems; furthermore, debugging these systems can be much more challenging [141].It's difficult to simply set a breakpoint or checkpoint to check and test the system at a certain point when you're developing a distributed system. This debugging approach is non-trivial, since the problems resulting from distributed networks are not deterministic because of the existence of their message communications[49, 87], what it was that the message began broadcasting, how long it took, whether it succeeded or in what order the transmissions took place. Each time a distributed program is executed, due to the different execution order, it is possible to achieve a different outcome and result from the previous run. This runs contrary to one of the normal principles in debugging typical applications where one execution with a set of inputs is believed to operate precisely the same with the same number of inputs [89, Chapter 10].

Although the execution can be different in a distributed system each time, it is not acceptable to wait for an error to appear, and then attempt to figure out where it is. Instead, the system has to self-assess its situation when it implements the distributed code. If a problem is found then the problem will be identified by the debugging tools and methods. One approach to do that is to test whether the system meets any global predicate, in which place a lot of work has been undertaken to examine and verify the various classes of these predicates[50, 51, 56, 141].

2.1.7 Sensor Network Data Fault

Sensor networks open the way for many methods of monitoring the environment. Researchers have become more able to get data about environmental, animal and weather behaviors. Sensor devices are susceptible to environmental conditions and they may fail during deployment time. This will produce faulty data and there are many types of this [112]. Therefore, it is necessary to ensure data integrity when it has the potential to be faulty. Figure 2.4 shows the classification of data faults in sensor networks.

First: The data-centric approach shows faults that concern the data readings. In particular, a data-centric approach does not have an explanation for the basic cause for each fault that occurred in the data so it is easy to identify by sensor reading behavior. Examples of data-centric approaches are:

- Outlier: sensor unexpectedly distant from models .
- Spike: Multiple data points with a much greater than expected rate of

change .

- Stuck at: Sensor values experience zero variation for an unexpected length of time.
- Noise: Sensor values experience unexpectedly high variation or noise.

Second: The system-centric approach mainly concerns faults related to the malfunction of the sensor. Examples of system-centric approaches are:

- Calibration: Sensor reports values that are offset from the ground truth.
- Connection or Hardware: A malfunction in the sensor hardware that causes inaccurate data reporting.
- Low battery: Battery voltage drops to the point where the sensor can no longer confidently report data .
- Environment out of range: The environment exceeds the sensitivity range of the transducer.
- Clipping: The sensor maxes out at the limits of the ADC.

Third: A fault-tolerant distributed approach is classified based on the behavior of the failed component. Examples of fault-tolerant distributed approaches are:

- Crash: A crash faulty sensor node loses its internal state and cannot participate in in-network activities.
- Omission: A sensor node that does not respond to the sink node on time, fails to send a required message on time, or fails to relay the received message to its neighbour is exhibiting an omission fault.
- Timing: A timing fault causes the sensor node to respond with the expected value but either too soon, or too late.
- Incorrect Computation: This refers to the fault that occurs when a sensor node fails to send the true measurement even though the sensing element of the sensor node perceived the true data.
- Fail stop: The fail-stop fault occurs when a sensor node ceases operation due to depletion of battery and alerts its one-hop neighbors of this fault.

Fourth: The duration approach classifies the fault types regarding their duration. Examples of duration approach are:

- Permanent: software or hardware faults that always produce errors when they are fully exercised.
- Intermittent: Temporary internal faults.
- Transient: Temporary external faults.

We have explained in more detail the sensor network data fault. It is important to explain that regarding the application of sensor networks there are many applications like habitat monitoring, healthcare monitoring, transportation, manufacturing and search and rescue. A sensor network application is divided into two basic categories: first; environmental monitoring and second, event detection. In environmental monitoring, data is constantly collected and utilized in a scientific way. However, in event detection, one is only interested in detecting the occurrence as specific. Here we will focus on environmental monitoring and the data collection. Habitat and environmental monitoring [101] is its most important application of the sensor network, which represents a class of sensor network applications with enormous potential benefits for scientific communities.

Mainwaring et al. [101] designed and developed a complete WSN system for habitat monitoring using a MICA sensor for a small island using 32 nodes. The sensors used are humidity, temperature, barometric pressure and surrounding light from which they determine breeding habits. The aim of this work is to understand the behavior of wild animals, especially in islands where the presence of humans can disturb the breeding. They monitored a sea-bird nesting environment. System architecture could serve numerous uses for habitat monitoring services, such as sensor sampling, data collection, routing and communication, health monitoring and network re-tasking.

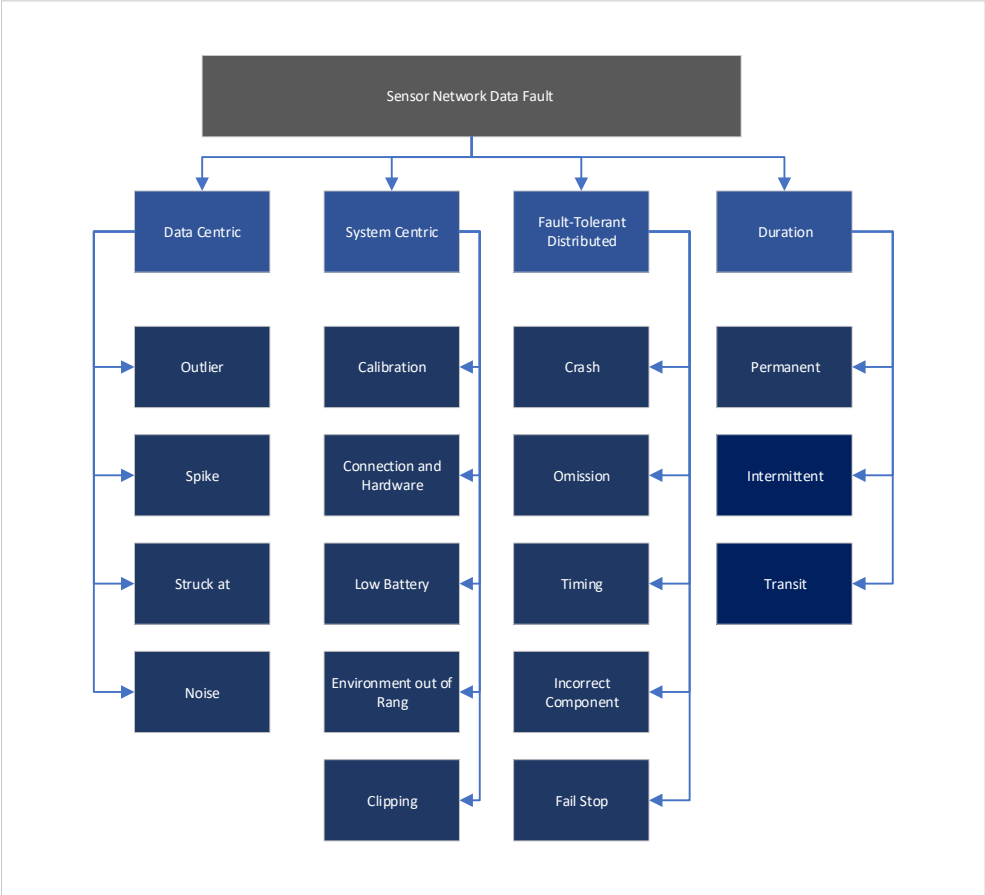


Figure 2.4: Overview of Data Fault in Wireless Sensor Network

2.1.8 Fault-Error-Failure Cycle

Types of predicates are important to implement a predicate checking mechanism. Nevertheless, types of errors that these predicate algorithms can detect are also important. It is necessary to consider errors and their origin. The fault-error-failure cycle can be examined for this reason. This can be done by looking into the cycle of failure-error-failure cycle. This procedure specifies that a *fault* triggered by either any external influence (e.g., radiation contributing to 4 bit-flips in memory [98]) or internal effect (e.g. code bugs) may or may not result in an *error*; this is the *activation* stage. An *error* is the fault representation (e.g., memory carrying incorrect values owing to bit-flips).If an *error* occurs outside the system, this failure can cause further errors through the process called *propagation*, or if it becomes a *failure* (e.g., requiring the opening of doors that should remain closed), the system failure is an observable deviation from the system specification. It is not always the case that faults cause errors or errors cause failures; often several faults or errors can cause a

single error or failure[13].

We need to consider the stage of our fault-failure cycle that we monitor and evaluate in our predicated language; we have three options: failures, errors, and faults. Faults would be a bad choice for two reasons for our predicated language: i) faults do not automatically activate and propagate into failures to determine several false positives by any system that detects faults; ii)fault protection typically is accomplished by strict quality control procedures used during hardware and software design and manufacturing rather than by predicate software control[2]. This ignores errors and failures. Failures are negligible because they are user-detectable errors by definition. However, the simple detection makes it essentially pointless for a developer to detect them because the harm has already been done, which leaves errors. If a dedicated system checks conditions and compares them to the predicted state, errors can only be evaluated. For example, an error correcting code, e.g., Hamming code, can be used to detect and correct an error (in this case, a bit-flip) after a fault occurred in memory (such as a voltage surge)[59].

Most of the debate included transitory faults such as those triggered by environmental conditions. There are various faults that are much easier to fix—faults triggered by software bugs. Such faults can lead programs to the wrong state and provide incorrect results. There have been few studies to detect distributed system faults in wireless sensor networks (such as deadlock [4, 119]). However, little research has been done to provide tools to help debug system.We are monitoring predicates caused by software or application bugs in our scenario. A TDMA protocol, for example, must assign different time slots within two hops without conflicting in order to run in a network.

2.2 Literature Review

Several debugging methods can address and explore how to detect bugs and faults in WSN. Techniques and methods are used to detect bugs and behavior of WSN application [56]. Few studies examined network behavior, routing protocols, and WSN clustering [63].

Few methods used *run-time* techniques to validate the evaluation of predicates in run time and tackle any problem that may arise when the system is running [43, 99]. While other *post-deployment* methods are used, file logs of predicates found are checked to evaluate and correct errors after the system completes working. New codes or patches can be uploaded to the wireless network after glitches are corrected [57].

There are several WSN methods in which WSN environments are simulated to check algorithms or applications before they are compiled on the hardware [16, 91, 103]. Contiki[115] is an operating system and an open source project that connects and compiles low-energy devices and applications used in the Internet of Things. Moreover, it can be used as a traditional development platform. Cooja is a simulator for Contiki, which controls and analyzes each simulated mote by a simulator, which fulfils and executes the Contiki program.

The previous study analyzed wired system *run-time* detection, which did not directly apply to sensor networks[78]. Our implementation is considered as *runtime* in which the system can detect faults once deployed. This project aims at understanding how to monitor and debug a wireless network system in which the monitoring will detect and monitor faults in *runtime*.Some of the methodologies and research used to monitor and debug WSNs are listed in this chapter.

2.2.1 Debugging and monitoring WSN in Deployment-time

Deployment-time tools validate system functionality at the time of deployment to lower the risk of early failure. The main advantage of Deployment-time tools is to minimize the requirement of revisiting deployment contexts which are composed of environmental states that are difficult to replicate. Deployment of a WSN is a stringent step because theoretical models and assumptions are not factual. These systems are often located in non-interest areas for humans because it is hard to reach for certain periods of time and it is necessary to verify the functionality of the system at the time of the deployment; this is why minimizing the expense of revisiting the site in the near future for re-deployment, maintenance, or repairs, is important.Other than deployment-time, there are several techniques for debugging and monitoring WSN, such as post-deployment monitoring [57][75][137][124][77][131][19][70][22][147].Post-deployment investigating devices are utilized for troubleshooting WSNs post implementation. This instrument is used to gather and break down information from programs at runtime like bundles.This section will survey much of the existing work performed on Deployment-time tools for WSN debugging.

A Deployment Support Network (DSN) [43] has been proposed by Dyer et al. as a stopgap, used for WSN in several environments for the development, testing, deployment and validation. A DSN allows for deploying and testing big numbers of equipment in a real scenario. The DSN is transparent, scalable and can be deployed quickly. Due to its nature as an overlay network and

autonomous operation, it does not disturb the target sensor network anymore than the traditional approach. The DSN nodes are connected to sensor devices using a programming and debugging cable and form an autonomous network. The sensor nodes can then be accessed through serial-port tunnels for online monitoring, debugging and uploading software updates, operated over the DSN. The DSN is working in an online environment, so the node logs data and events from the target nodes and delivers them to the server in push or pull mode. It receives and executes commands from the client and finally safely reprograms target nodes. DSN Nodes performed the following tasks: they configure the DSN network and provide status info, Adaptive topology control, Reliable multi-hop connectivity and Time synchronization. The DSSN consists of the following components: the DSN Backbone Network, DSN Nodes and the DSN Server and Client. In term of data collection the DSN uses a server-logging database, where it collects the log-messages from all the DSN nodes. DSN uses multiple nodes: the DSN node and the target node. The DSN has a lower throughput for debugging and control information so the researcher must be aware of this and choose the rate of generated pushed messages accordingly or change to pull mode if possible. The efficiency of the DSN means it has less interference since debugging services and the sensor-network application are clearly separated and do not share the same computing and radio resources. Running a data dissemination service on the target-nodes would require additional memory that is large enough for a whole code image. Expensive extra memory that is only used for development is no feasible option for industrial products.

SeeDTV Deployment-Time Validation for Wireless Sensor Networks [99] has been proposed by Liu et al. as a deployment time validation framework that consists of techniques and procedures for WSN status assessment and verification. SeeDTV is supported by a portable, lightweight, and low power in-situ user interface device called SeeMote. SeeDTV has demonstrated the potential for the early detection of problems at three levels of WSN in-situ validation: sensor node devices, a wireless network's physical and logical integrity, and connectivity to the back-end such as a data server over the Internet. SeeDTV is working in an online environment; a DTV application that runs on the SeeMote. The focus at this stage is a single node. By default the application assumes that the sensor nodes are switched on and ready for communication. Therefore SeeMote starts by periodically broadcasting a query message that requests the node configuration and health status, including their local ID, RSSI value, remaining battery voltage (in percentage), and ADC readings (channel 0 to 7 for MICAz motes). Once the SeeMote starts receiving the

requested information, it displays the status on the LCD screen. The navigation between the modes is conducted by the user through the user interface as follows: The Statistics screen shows all the nodes currently present in the network that are reachable by SeeMote. The information displayed includes the number of packets received from the nodes and their latest RSSI value. A user can navigate among the nodes using the SeeMote buttons and select one for closer examination, which brings them to the Options Screen. The Options Screen presents several options: Check Status to query the specific sensor node, Status Screen to display the current status, Restart to initialize the number of packets received from the node, Retimesync to re-synchronize the time on the network, and Reset to Reset the whole network system. The Status Screen displays the current parameters of a previously selected node, reflecting its status and health. The available information on this interface is the node ID, the number of packets received from this node, the RSSI value between this node and the base station, the remaining battery charge as a percentage, and a general health status as a text message. A push of a button leads to the ADC screen providing more information. The ADC Screen is a complement to the Status Screen allowing one to observe the actual Analog to Digital Converter (ADC) channel values as they are sampled at each request. These values represent the sensor data, for example, the light intensity received at the location. Thus we can observe the change of the sensory data in real time on the SeeMote. In addition, a moving average of the ADC values over time is displayed to show the trend or average of the sampled signal. It can be used to estimate the presence of noise. The higher the noise, the more the current value will differ from the average value. Data is collected by the system periodically, reported to the mainland and stored in a database connected to the Internet accessed over an amplified wireless gateway. Focusing on single node, the validation process is designed to easily detect and prevent sensor node failures while they are in the field. SeeDTV is efficient since it can be successfully applied during the design and debugging phases allowing early detection of the faulty components.

H-SEND (Hierachichal Sensor Network Debugging) [63] is a framework for detecting faults in sensor networks. It was designed to minimize energy consumption and be capable of handling very large networks. As part of the implementation the developer must specify invariants within their code using a custom language. These invariants are then semi-automatically inserted during compilation. If an invariant is violated at runtime actions are taken (such as increased logging frequency, or an error message to the base station). Using these responses developers can use the information to fix the software,

which possibly includes uploading a patched version of the firmware. There are typically three different invariants that can be specified: Local vs. Multi-node, Stateless vs. Stateful and Compile-time vs. Run-time. The first indicates whether the predicate needs information about the node it is being evaluated on (Local) or other nodes in the network (Multi). The second is if the invariant depends on the node's execution state (Stateful) or if it doesn't (Stateless). The third indicates whether the invariant involves values that are fixed at the compilation time (such as integer constants), or if it compares against values obtained during execution (such as neighboring states or previous states). H-SEND is optimized for WSNs in a variety of ways. For example, it minimizes the overhead by buffering messages it needs to send, and piggybacking on existing network traffic. Due to the hierarchical nature of the protocol, multi-node invariants can be checked efficiently at the closest parent node with all the required information. Data can be collected by the programmer by specifying the invariants and the variables to be observed using multiple nodes, observing data trends that are determined only at run-time. It is efficient to use H-SEND since it reduces overheads in terms of code size and network traffic.

Sympathy [124] is a method for identifying and localizing failures. Sympathy is intended to be run either in pre- or post-deployment environments where it collects data from distributed nodes at a sink. When insufficient data is received it is taken to imply that a problem exists (insufficient data is component-defined). The idea is that by monitoring data between components (both actively and passively) the system can identify what kind of failure occurred in a certain area of the network, both of which are very useful when trying to debug a failure. It does, however, have some downsides. The first is that there is assumed to be no traffic and thus no application traffic or network congestion. These are real issues especially when applying this kind of debugging to a high-throughput sensor network. There are also a number of spurious failure notifications, which the authors are working on reducing, by applying a Bayes engine. Sympathy considers three possible sources of failures for a node: self, path and sink. Sympathy monitors regular network traffic which is assumed to be frequently generated by each healthy node: sensor readings, synchronization beacons and routing updates. Sympathy treats the absence of monitored traffic as an indication of faults. It uses metrics traffic generated at the nodes to localize the failure. These metrics include connectivity metrics (like routing table or neighbor list), flow metrics (like packets transmitted and received per node and per sink) and node metrics (like uptime). The measurements expire if they are not updated for a certain period of time. Sympathy determines whether the cause of failure is in node health,

bad connectivity/connection, or at the sink by using an empirical decision tree. Gathering data at a centralized sink involves locationNodes periodically sending metrics back, which combines this information with passively gathered metrics to detect failures and determine their causes. The target is multiple nodes and the traffic conditions are: (1) no application traffic, just routing traffic (no-app); (2) one 10-byte application packet sent every 60 seconds (traffic60); (3) one 10-byte application packet sent every 30 seconds (traffic30); (4) one 10-byte application packet sent every 10 seconds (traffic10). In each case, thresholds were set accordingly; in the no-app case, Sympathy tracked Sympathy metrics instead of application data. It is efficient since it accurately detects injected failures with a relatively low network overhead and low latency. Sympathy is able to reduce excessive failure notifications .

NodeMD [84] an alternative to debugging compared to either specifying a predicate or an invariant, or using machine learning to learn what to check for, is to instead look for the faults that arise after a failure has occurred. NodeMD also has support for a number of useful features to aid with the debugging. The first is a debug mode that is entered when a fault is detected. The debug mode freezes critical parts of the system to prevent the fault from leading to errors. This prevents events such as a context switch after a stack overflow that would end up being performed incorrectly. The debug mode also resets certain OS components to a safe state, so that some components (such as the radio) are usable to report the fault that was detected. The other two useful features are support for remote debugging and an implementation of dynamic reprogramming algorithms to update the firmware across the network. The remote debugging feature allows a human to access all the available fault information of a sensor node. Parameters can be changed to expose more information when the node is queried and the potentially useful feature of telling the node to restart is also available. Overall NodeMD provides lots of insight into the low level failures in wireless sensor networks. Its data is read from four different sensors into a packet buffer and sent over the radio every 1 second. NodeMD allows a user to define their own application-specific conditions for triggering the detection of a fault and the subsequent halting of the node. It is efficient since it detects stack overflow. NodeMD helped us to pinpoint the location of an actual legacy bug in MOS.

DICE [56] is a protocol used for WSN-based distributed monitoring of global invariants in physical processes to improve the network lifetime. DICE provides a declarative language to specify invariants and a run-time support enabling efficient monitoring of their violations. The run-time can be configured to use either the structureless FLAT protocol or the TREE protocol, which

instead relies on an overlay. FLAT provides increased fault tolerance by allowing any node to detect a violation, at the expense of increased overhead in scenarios with a high rate of changes in the monitored application state. TREE provides improved performance in this scenario by structuring and optimizing the dissemination of relevant state changes, but this very structure makes the approach more complex, as it requires additional mechanisms to preserve structure in the presence of failures, and limits the ability to detect violation only to a subset of the nodes. DICE invariant is expressed by combining predicates defined over the state of multiple WSN nodes; single node (local-predicate) and multiple node (distributed-predicate) DICE invariants are combinations of atomic predicates over variables whose value is a function of network nodes. DICE supports three kinds of attributes: constant attributes, attribute representing the type of node and attribute for a CO2 reading. DICE provides a declarative language to specify invariants and a run-time support enabling efficient monitoring of their violations. Invariant violations are detected in a timely and energy-efficient manner.

From the above literature review, we can see some of the disadvantages where studies and researches are lacking and need to be addressed, as well as others that are not mentioned in this section[125][76][147][135].Most of these research implemented fast detection techniques and discovered new techniques in WSN, but most of them had a high overhead rate when the implementations took place or didn't consider how much these methods would cost. External hardware such as a multicolor-graphical LCD display and detachable storage are required in some of the methods, which adds to the cost, weight, and power consumption of the motes.Despite the fact that some of the methods are sclabale and have flexible invariants that can be applied in the debugging system, the tool lacks the diagnostic framework required to determine the root cause of the error detected. Very few studies have been conducted to monitor low-level system components rather than high-level application requirements.Some of the solutions require a significant amount of human effort from application developers in order to determine whether or not an error has occurred. Some technologies do not provide user interfaces, making it easier to construct debuggers for specific programs.

2.2.2 Selecting Monitors and Clustering in WSN

Various routing protocols uses models based on clustering. Clustering is a type of hierarchical routing with a strong history of improving network efficiency and is typically used to determine an efficient placement in digital circuits for logic gates[86]. The basic principle of clustering remains the same for these early and more recent applications, grouping (in a logical sense) network nodes

in accordance with the (typically geographical) localities[96]. Every node in one of these clusters is connected to the rest of the network, which is a particular node in each cluster known as the cluster head (CH) in a wireless network. The advantage of clustering a network is that it eliminates global network traffic. In a clustered network, each node can route messages to its CH rather than to transmit it all the way to its destination. In several applications, CH may perform data aggregation or use a compression feature on messages forwarded to reduce the number of messages between clusters; the essence of these functions is often application-specific[34][138]. Nevertheless, the existence of clustering means that the CH has a considerably higher overhead than any other node in its cluster, because it must listen to every message sent from the cluster and likely forward just as many communications to the other CHs in the system. This leads to an increase in the energy consumption of CHs in environments that have no permanent power source (e.g., WSNs), allowing their batteries to drain faster[33]. This results in the reliance or inspiration of modern cluster algorithms(e.g., [73],[82]) on the low energy adaptive clustering hierarchy (LEACH)[60].

Normally, CH sensors can communicate with the base station in such protocols. LEACH is a highly described protocol using this approach [60]. CHs are elected by random rotation in LEACH, and members join within 1-hop. LEACH does not guarantee that each member belongs to a cluster, and the distance between CHs is 1 hop, which does not meet our objectives.

Another common routing protocol is collection tree protocol (CTP)[53], which constructs trees that are routed at the sink. To reach that sink, data from a source travel up the tree to decrease data traffic. Moreover, CTP requires more transmission networks to keep topological information up-to-date, which can be energy-consuming if not optimized. Duplicate messages may cause messages to be lost in certain nodes in the network.

The authors present a fast local clustering service in [32], which partitions a cluster of multihops. To build clusters, the algorithm uses a solid disk. The cluster has a CH so that all the nodes within the unit distance and few nodes within the CH range m belong to the cluster. We initially inspired our clustering algorithm based on this study; however, the deployment process did not meet our proposed goals. Key disadvantage is that the clustering implementation to form clusters is based on the RSSI calculation and timers. While our study is based on the method of RSSI and the number of hops involved in creating the clusters. The study considers the number of messages and the number of hops that the data need to travel to reach the CH and cover the distance within m unit of each CH.

2.2.3 Distributed Predicates

It is necessary, in the first place, to understand what predicates are applicable to distributed systems.First, there is a difference between global and local predicates; global predicates involve taking a clear global snapshot of the whole system and verifying whether the snapshot satisfies the global predicate[50]. While local predicates that instead operate with a subset of the network[51]. Such predicates also have a concept of consistency – a stable predicate must remain true once it has become true (e.g. termination),while unstable predicates can turn from true to false. Lastly, there is a distinction between weak and strong predicates, where there is a weak predicate if there is an observation that the predicate is true and a strong predicate holds if it is true for all observers of the distributed computation [29, 50]. Understanding what types of predicates there are is important since, when testing some properties of the method, a certain type of predicate will be necessary and therefore some specification will be needed to ensure that the predicate is checked correctly. An example of this is the fact that the same application may not detect unstable predicates while running an algorithm using global snapshots to detect stable predicate. The predicate may shift to false, and then return to true before the next snapshot. We address the issue of predicate detection in distributed systems in our work. In this field, similar work investigates the detection of stable predicates, whose truth value changes only once in the system's lifetime [26].So we don't concentrate on unstable predicates whose truth value can switch repeatedly [50].

Chapter 3

System Model and Experiment setup

This chapter discusses the abstract models we assume in this work that underpin the solutions we provide. It also discusses the environment used to experimentally evaluate the proposed solutions in order to analyze their respective efficiencies. Simulation and implementation on real-world hardware (i.e., deployment) are two key experimental approaches. Both methods have advantages and disadvantages. For example, simulation helps test various scenarios and larger networks, while deployment on real hardware allows for the execution of the program in real life but at the expense of scalability. As both methods have their own merits, the developed algorithms will be evaluated using both techniques. The architecture of the system is also outlined, as well as the components of the system. Finally, the experiments setting and methodology for implementing the system are described in this chapter.

3.1 Models

A wireless sensor network contains network nodes (or motes) that communicate among themselves via their respective radio interfaces. Every node has limited computational resources, i.e., low memory, weak CPU and others, and has finite amount of energy supplied by batteries. Two nodes m, n in the network are called neighbors, where m can directly communicate with n and vice versa. The network has a dedicated node known as sink, which is responsible for data collection and provides a link to the base station.

Thus, a wireless sensor network can be modelled as a graph $G = (V, E)$, where V represents the set of nodes and E is the set of edges, i.e., direct

communication links between neighbors. Edges between two neighbors do not necessarily persist in the network owing to issues such as node failures, link failures, and others. However, we assume that communication between two neighbors is reliable whenever such a link exists. In such a case, we say that these two nodes are in each other's i-band. Otherwise, when two nodes can directly communicate with each other, but communication is unreliable, these two nodes are in each other's o-band. Such a nature of communication has been observed in [31]

Definition 1 m-Hop Neighbourhood*: Given a node n the m-hop neighbourhood of that node is the set of nodes that are within m hops of node n. When we refer to this neighbourhood it will not contain the node n.*

This ensures that, when implementing simulations and physical hardware first, no assumption is made that such communications or links are reliable and stable in wireless networking. Specific explanations will be provided in the next chapters about how the messages reach their targets along with data dissemination methods, but this is not always guaranteed to reach their target (a wireless sensor network disadvantage). A further assumption is that all the nodes possess the same hardware capabilities in the network. This results in all the nodes having the same capacities or properties (transmission range and power level). Another factor is that when this network design is implemented, interference from other networks, e.g., electromagnetic interference, will not be considered.

Definition 2 P_{single}*: The set of predicates that are evaluated by a single node in the network.*

Definition 3 P_{all}*: The set of predicates that are evaluated by all nodes in the network.*

Definition 4 P_{sets}*: The set of predicates that are evaluated by a sets of nodes in the network.*

It is the main contribution to this project and the key concept to explain the difference in output between each method. Please note that the above definitions are defined at the position where the predicates will be evaluated, not when they will be evaluated. When it comes to determining when the predicates will be evaluated, the location or selection of monitors in the network is taken into account first. For example, (P_{single}) is a single node that evaluates predicate messages in the network, typically the sink node. For most wireless sensor network debugging tools, this scenario, which is a simple and straightforward technique, is implemented. Practically and technically, this

scenario is not optimal in terms of efficiency and messages lost owing to the nature of the wireless sensor network. Therefore, several messages collected in one node result in a message loss and an increase in power consumption. This scenario would result in a lot of messages as the number of nodes in the network increases. However, this scenario is considered to be implemented and tested in the project because it is not desirable to compare it with other structures.

The other scenario is P_{all} where all the nodes in the network can evaluate predicates. This has been called in the wireless sensor network ($flat$) architecture where all the nodes apply and act with the same functions and properties. This form of decentralized architecture would potentially consume more data and energy. However, technically, the flat architecture would be more desirable in terms of performance. This form of networking architecture requires a more specific dissemination method when the information is collected by default to reach a certain node or all the nodes. In the next sections/chapters, the data dissemination process and the overall performance will be explained in detail.

The predicates in the ($Psets$) scenario can be evaluated by specific nodes or a subset of nodes in the network. In WSNs, this is referred to as clustering. Cluster heads will be selected and designated in order to forward data to the sink or conduct other functions in the network. However, a similar approach is intended here for selecting monitors; these CHs will operate as monitors and evaluate the correctness of the application's behaviour. In other words, the network will select a subset of nodes to act as monitors in order to collect and evaluate data.

Definition 5 Diameter δ*: The maximum distance between any two nodes in the network. This refers primarily to the distance of the monitors to their nodes that fall within their interference.*

Definition 6 *distance*(n, m)*: The length of the shortest path between the nodes* n *and* m*.*

The δ represents the distance between monitors and their nodes that come under its interference. In this process, communication needs to determine the path of predicates to travel through the network by number of hops. The monitors should acknowledge the number of nodes need to travel to reach them. The major outcome of the communication between the monitors and their nodes that evaluate messages that come under interference is the selection monitoring method, and how the information is disseminated to the monitors where the distance dissemination manager must first be applied.

3.2 System Architecture

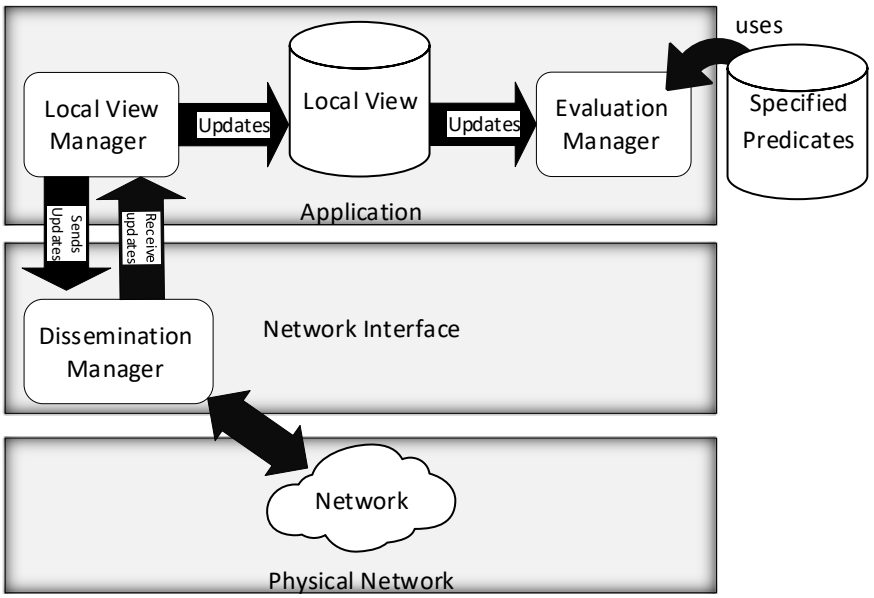


Figure 3.1: The architecture of the system components at runtime

Figure 3.1 shows the system's main components at runtime. A data framework implementing the *local view* as described in Section 4.3 is the core of the system. Each *local view* entry *(value, source, timestamp)* is related to a tuple. The value entry is applied in this scenario to determine the actual data needed to evaluate from the application, such as the time-slot or temperature that originated from the source node. The ($nodeID$) value representing the node linked to the local entry is reported by the source entry. While the time stamp field is used by the dissemination manager to ensure that *local view* notifications are properly processed, as defined in Section 6.2.

The manager of predicates or the evaluation manager is responsible for checking the *local view* if the predefined predicates are violated. Relevant changes to the *local view* are defined by the *local view* manager, based either on value changes to local attributes or on notifications obtained from the network.However, this is greatly influenced by the distributed TDMA algorithm (explained in section 4.3.5) that keeps modifying the values of the predicates across the network.

3.2.1 Evaluation Manager

A *local view* change causes the evaluation manager to evaluate whether all of the predicates being monitored are violated. This violation will be reported immediately after the evaluation manager checks the violated predicates , these reports and notifications are explained in Section 6.1.5. Otherwise, the evaluation manager begins a timer with various durations, the durations variation is introduced to provide the research further analysis where the system able to change its self between high cost or high accuracy depends on the natural of the application. Furthermore, the evaluation manager is able to record and monitor the predicates independently where redundant predicates or checked predicates are recorded and able to remove already checked predicates. Because the space resources available to store data on WSNs are limited, we developed a history buffer on the monitors to record as much predicate data as possible. This implies that when the data has been evaluated, monitors will free up space. Section 7.4 discusses into further detail about how monitors can improve the correctness of evaluated predicates by adding the history buffer.

3.2.2 Dissemination Manager

The Dissemination Manager is responsible for the dissemination of predicate data over the network. The dissemination manager delivers a control message to monitors regarding the values/source. The statuses of various nodes will then be received. The predicates will subsequently be evaluated. Several network algorithms are introduced and implemented in Section 6. These network algorithms are motivated by different scenarios, which are primarily based on the selection of monitors from the network. Scenarios and network algorithms are explained in section 6 . Network algorithms are primarily divided into two parts; one part uses a selection monitor algorithm where predicates are evaluated by subset of nodes in the network, and the other algorithms evaluate predicates from all nodes in the network. As this research works for wireless sensor networks, efficiency and cost are mainly influenced by the predicate algorithms used. The Dissemination Manager uses different communication networks where it operates independently from other managers such as the *local view* manager.

3.2.3 Local view manager

In the *local view* manager, any changes will immediately update the *local view* entries, several of which are executed by the algorithm in Section 4.3.5. The main changes originate from the distributed TDMA algorithm mentioned in Section 4.3.5, where the distributed algorithm proceeds to distribute $time-slot$

values until processing is complete.These changes and updates, in turn, needed more dissemination across the network where the dissemination manager will deal with this. Once the *local view* changes the values where predicates need to be checked, the *local view* automatically updates the *local view* updates where the system can check multiple predicates independently of the *local view* manager through this procedure.This enables the system to check the predicates independently of the *local view* manager and the *local view* manager to disseminate its own updates through the network.

3.2.4 System Runtime

In order to evaluate predicates from these monitors, the system initially needed to select monitors which explained in Section 5. Eventually, the heuristic algorithm will complete its process until further processes are triggered by the system. The heuristic algorithm has a duration of 5 minutes where it is required to process random topology and large nodes, while it will take less than 1.5 minutes in cases where smaller nodes and grid topology applied.

If the system has completed the selection of monitors, the TDMA protocol will start running immediately, and as soon as the TDMA begins assigning time slots, the monitoring system will start evaluating predicates.Eventually, the TDMA protocol will complete its own network time slots allocation before the TDMA communications are synchronized and established. It depends on the efficiency of the TDMA algorithm and other factors, such as network topologies and large network nodes, to complete the time slot allocation.Section 4.3.5 describes the TDMA protocol.

The monitoring system will continue to monitor the predicates, as stated above and in Figure 3.2, even though the TDMA has completed its own allocation. The explanation for this is that even such bugs are triggered by multiple variables such as software errors or environmental bugs after the completeness of both algorithms. However, the monitor system mainly focused on the behavior of the TDMA execution where the monitoring system expected the TDMA algorithm to be able to correct itself when there is the same time slot for any 1-hop neighbors.

3.3 Simulation Setup and Methodology

Implementations run on Contiki operating system [36]—an event-driven open source operating system for wireless sensor networks. Here, programs are written in plain C language using purpose-built libraries, which provide access to core features of Contiki, e.g., the Rime communication stack and proto-

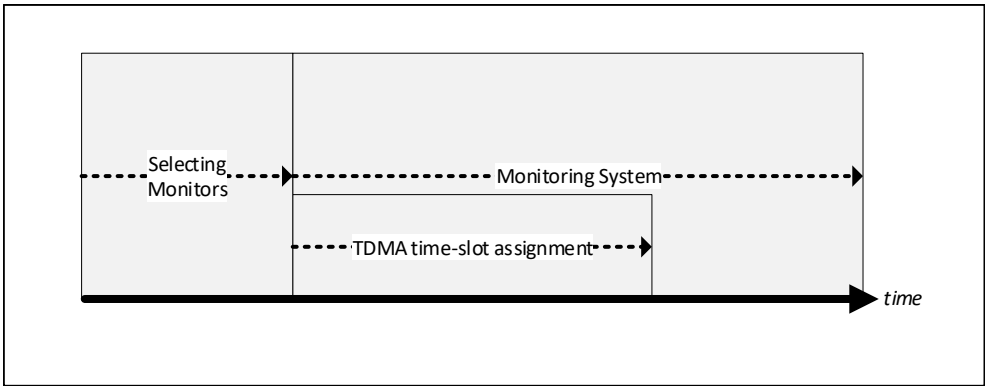


Figure 3.2: The system Runtime

Parameters	Values
Simulator	Contiki 3.0 Operating System with Cooja
Radio Medium Model	Unit Disk Graph Medium (UDGM) with Distance loss
Node Communication Range	Tx/Rx/Interference Range – 50m/50m/100m
Node Distance	North,East,South,West -20m
Node Type	Z1[1] and msp430[30]
Network Layer	Contiki Rime
Number of Nodes	25 ,50 ,90 ,121 ,169 ,225
Topology	Grid topology
Duration of Simulation	70 minutes per simulation
Number of Runs	20 Times
RSSI parameters	i-band larger than-55 ,o-band smaller than -55
Measurement and calculation the set of values	Standard Deviation and average

Table 3.1: Simulation Settings

threads[3]. Contiki is built as an open source platform to connect low-power battery-operated devices to the Internet of Things[12]. Simulations run on Cooja, which is a simulator for the Contiki OS platform, where each simulated node or device is an actual Contiki system that is managed and analyzed by the Cooja simulator [115].

We evaluated algorithms and predicates in different sizes in our simulations and positioned them in a grid topology. We also ran simulations with topologies that were generated at random. They are, however, much less insightful due to the numerous sources of randomness (topology, value distribution, timers, etc.). Configurations of the evaluation network range from 25 to 225 nodes. The nodes are arranged in such a manner that each node had four neighbors, one to the north, one to the south, one to the east, and one to the west, with the exception of nodes in the edge with three neighbors and nodes in the corner with two neighbors. In the configuration, the sink is placed in the left corner of the network, and the address assigned is ”1.0” for clustering set up for all

implementations. The predicate algorithms begin after the sink informed that the clustering process and the setup phase have finished selecting monitors in the network. The cluster setup has been provided 10 min to finish its selection of monitors before the evaluation of predicates. The simulation was carried out for 60 min, and then, it was stopped to gather data. The radio medium was set to be "UDGM: Distance Loss" when setting up the Cooja simulator to run simulations, the mote startup delay was set at 1000 ms, and a random seed was set to be used on each reload.More information about the simulation experiment settings can be found in table 3.1.

The predicate was evaluated every 2 or 4 minutes, depending on the parameters, after the predicate checking algorithm was set up.The explanation for this is that each monitor would have the time to obtain the necessary information from the neighboring nodes to compare and evaluate them.This will allow each monitor time to collect data from distant nodes, especially when the 2-hop parameter is implemented.

A Contiki library called rimestats [1] was used to collect metrics on the energy usage of the motes,in the MAC layer, this library is integrated and records deep statistics.Contiki comes with a library to calculate energy consumption[41], but to keep it straightforward and concentrate on the number of messages that are sent and received.In Graph 3.3, a screenshot of the print logs is shown from which data such as Tx and Rx can be obtained.

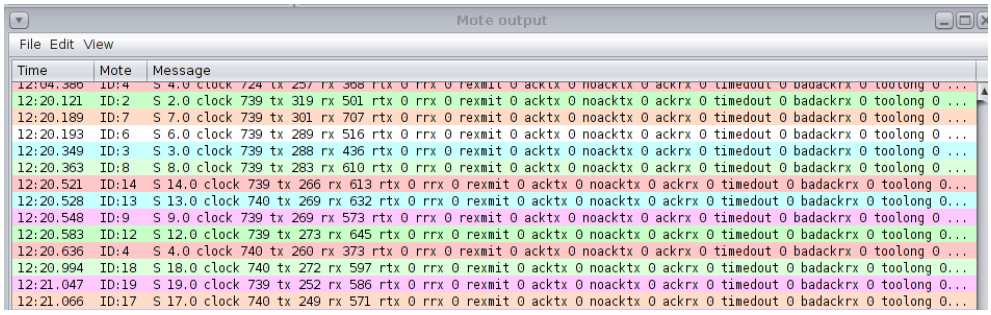


Figure 3.3: The log files that generate the Tx and Rx calculations and other useful information for each node.

The implementation provided with sent and received counters to measure how many messages the TDMA algorithm generated, which were increased when a message was sent or received, to determine the number of messages. There is a one-to-one correspondence between the number of total broadcasts and the number of transmissions made at the MAC layer, since the TDMA protocol was implemented with basic broadcasts, the same applies to receiving a message.The

[1] `http://contiki.sourceforge.net/docs/2.6/a00357_source.html`

difference between the total and the TDMA messages was avoided to measure the message cost of the predicate evaluation algorithm.This also applied to clustering set up messages when the cost of messages was calculating. In other words, only the predicate evaluation messages are measured in order to provide a fair measurement where the number of messages increases by the number of nodes and with each run, the TDMA does not have a constant number of messages.

The TDMA algorithm printed any changes in the allocated slot as well as the time the slot was changed to verify that a predicate was correctly evaluated later. When a predicate is evaluated by the monitor, the result will be printed with time and each node will record the time when the slot value has changed. The predicate was then evaluated by analysis scripts using the latest slot value from before or until the predicate was evaluated to evaluate the predicate itself.In other words, in order to measure the number of missing violations when a node has changed, the script analysis will later compare the time that each node has changed with the time the monitor result is printed. The time-slot and other details will continue to be tracked and checked in Graph 3.4 log files to be used later in the scripts to evaluate missed violations and latency.

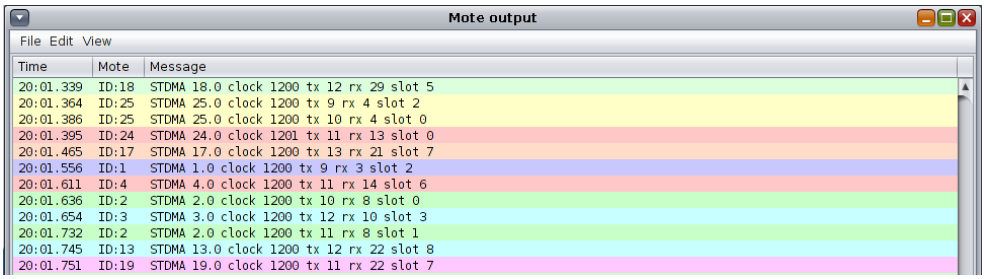


Figure 3.4: A TDMA information screenshot, such as the tx, rx and time slot for each node.

3.3.1 Clustering settings

Several implementations have been carried out in the clustering algorithm to ensure that results are expected to meet the clustering algorithm requirements. Sixteen CHs are selected from a 10x10 network size, as shown in Fig. 3.5. Several implementations with various applications, e.g., aggregation of temperatures, humidity and light data from the member, and aggregation to the sink are implemented before the evaluation monitoring is integrated to ensure hierarchical clustering works and local clustering heads collecting from members as expected.The RSSI is set up such that (i-band) is larger than -55 and (o-band) is less than -55.Table 3.1 shows the communication range, node

distance, and other settings.

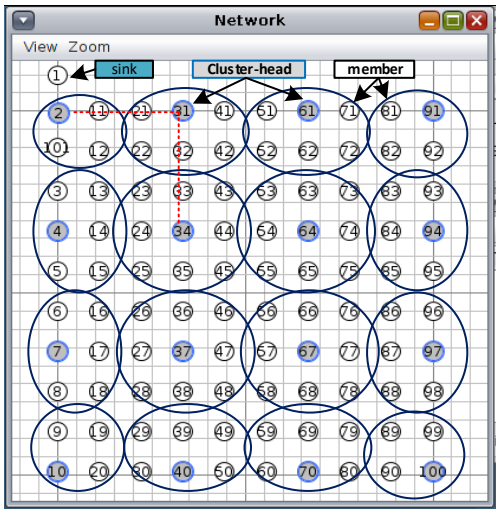


Figure 3.5: 10x10 Network Clustering

3.3.2 Firmware Size

Table 3.2 lists the program memory occupation based on each component of the system. The TDMA protocol, which is implemented to evaluate its predicates, is considered based on the local view manager memory size. In all the implemented algorithms, the size of the evaluation manager remains at 19KB, which includes the size of the predicate code, which defines the specifications of the invariants. As each algorithm differs in the way each network library is implemented, the code size in the dissemination manager can vary from one algorithm to another. The code size of the cluster algorithms will be larger than flat algorithms since the MSP algorithm is implemented as a clustering routing, and this was also implemented alongside the dissemination manager.

Component	Cluster-Periodic	Cluster-Event	Cluster-Polite Gossiping	Flat-Event	Flat-Polite Gossiping
Total	**51 KB**	**51 KB**	**50 KB**	**48 KB**	**28 KB**

Table 3.2: Program memory occupation of the system components.

3.3.3 Polite-Gossiping settings

When the experiments use polite gossiping and are deployed to evaluate predicates, the distribution structure is initially modified from one-to-many to many-to-many communications. The configuration of polite gossiping is as follows: ti= 200 ms and th= 4 s represent the lower and upper bounds of the transmission duration, respectively, and n= 3 represents the number of neighbor transmissions that causes a message suppression.

3.3.4 Dissemination protocols set up

The time to evaluate predicates in cluster-periodic is the time when the monitor sends a request message to member nodes. In a 1-hop and 2-min scenario, the monitor will send the request data to member nodes every 2- min and wait for the data to arrive at the monitor to evaluate them. This would allow most protocols the ability to disseminate or request their data at the requested destination, e.g., the time in the checking algorithm that checks the time-slot of each node and floods if the time-slot modified in cluster-event has been set to zero to achieve the best output. To observe the algorithm that can match with changes occurring throughout the TDMA assignment slot, most parameters are run with different numbers, and their analyses are described in the following subsections.We ran simulations in (1-2 hops) to have a clear picture of how to apply different applications that might use a larger checking hop parameter.In an ideal scenario, TDMA checking predicates on a (1-hop) would fit perfectly into its structure, but this demonstrates (i)the ability to expand the hop parameter from the dissemination manager and (ii)provides more experiment analysis for future work.Furthermore, the time required to evaluate predicates (Event-based architectures) and the time requested (Request-based architectures) are set at (2-4 minutes), respectively, to provide more clarity when the system is able to evaluate predicates.

3.3.5 TDMA protocol settings

The simulation runs the TDMA algorithm with normal settings at the start of the evaluation, which is mentioned in section 4.3.5. As the monitoring system is able to detect the TDMA protocol with normal settings and timers, such as timers for each node, to adjust its time-slot if a node has not received any response from neighboring nodes or the periodic time for each node to determine when to send its current time-slot value, this determination is important to evaluate. In TDMA algorithm protocols, time-slot values change frequently, whereas monitoring temperature values change less frequently. Graphs 7.1,7.2,7.1 7.3,7.4,7.5,7.5,7.6,7.7 and 7.8 demonstrate these evaluations while the TDMA operates as normal without interfering with its own configurations or modifying the slot-changes according to various analytical and evaluation methods. As stated, in terms of number of changes, these graphs which vary from other graphs with other settings as the TDMA protocol will produce a large number of changes that will affect the overall result.These large number of changes are generated and manipulated artificially in order to generate more changes than normal.Section 7.3 explains the modifications to the TDMA default changes.

3.4 Testbed experiments setup - FIT IoT-Lab@Lille

Testing methods focused solely on simulations have become inadequate as wireless sensor networks have evolved as a technology. As a consequence, it's crucial to bring recently designed methods to the test on testbeds, which are large-scale implementations of actual hardware.Testbeds come with a range of features, including hardware, software measurements, supported sensor network OSes, and how jobs are submitted to such testbeds. There are a range of test beds that will provide certain facilities to test the compiled system on actual hardware at the beginning. Furthermore, deciding whether these hardware systems will support the operating system that the project will compile is challenging. The biggest concern arose at the beginning when the old Contiki operating system version was old to be compiled into the newer hardware. Thus, the project needed more time to translate and transfer the library files to a newer version of the Contiki operating system.Another issue was that the size of the compiled system components was taking up more space, despite the fact that the mote type in both the Contiki operating system and the available tesbeds services was small and different. Thus, the FIT/IoT-LAB [5] was chosen to conduct experiments.

In particular, the mote type is implemented and compiled into the IoT-LAB M3 board, which was created specifically for the IoT-LAB testbed.It is based on a STM32 (ARM Cortex M3) micro-controller with an ATMEL radio interface in 2.4 GHz and 4 sensors.It has a form factor (i.e. the CAD design of the embedded system) of 4 cm wide and 5 cm long in terms of dimensions. The Cortex M3 from ARM is the CPU on which the IoT-LAB M3 board is based.The Cortex M3 is a 32-bit processor with a maximum speed of 72 MHz. The board has 64 kilobytes of RAM and 256 kilobytes of ROM. The AT86RF231 radio chip on board communicates over the 2.4 GHz ISM frequency band. This radio chip was developed to implement the IEEE 802.15.4 MAC layer, which is used by the Zigbee communication protocol and others. The maximum bandwidth is 256 kbits/s. A small, 9 mm long ceramic antenna is welded onto the circuit.This type of radio technology is known as "short-range," with maximum indoor distances of 40/50 meters and outdoor distances of up to 100 meters.

As soon as the node type is introduced and deployed, it is necessary to choose which topology and server the provider has to run the system. Some of the servers in IoT-LAB are ready and available to compile the system, but some of these servers are not completely accessible or have very little time to execute algorithms. Furthermore, because of the wide range of node types and network topologies, it may be difficult to run the algorithms. This eventually led to the Lille server, which was selected and chosen as the tesbed topology and

server.The nodes are installed in the two buildings at Inria Lille–Nord Europe on the Lille site [2].The majority of it is dispersed throughout the first building labeled by (A building), making it ideal for large network and multi-hop experiments. In the B building, an additional portion is deployed in a structure called Le Cube, which contains a variety of boards. The dedicated room for the tebeds is a 225 m2 space with a hallway dividing a large room and five offices (respectively on left and right in topology figure 3.6). In this room, nodes are distributed across the ceiling and along the perimeter of the large room with wooden posts.

When running the tesbed in such a topology, the selection monitors algorithm must first be considered.In general, the experiments used a 3x6 network topology, which was similar to the one used in the simulation experiments with different node sizes. Some nodes did not participate in the experiments for the sake of simplicity and demonstration.

[2] `https://www.iot-lab.info/docs/deployment/lille/`

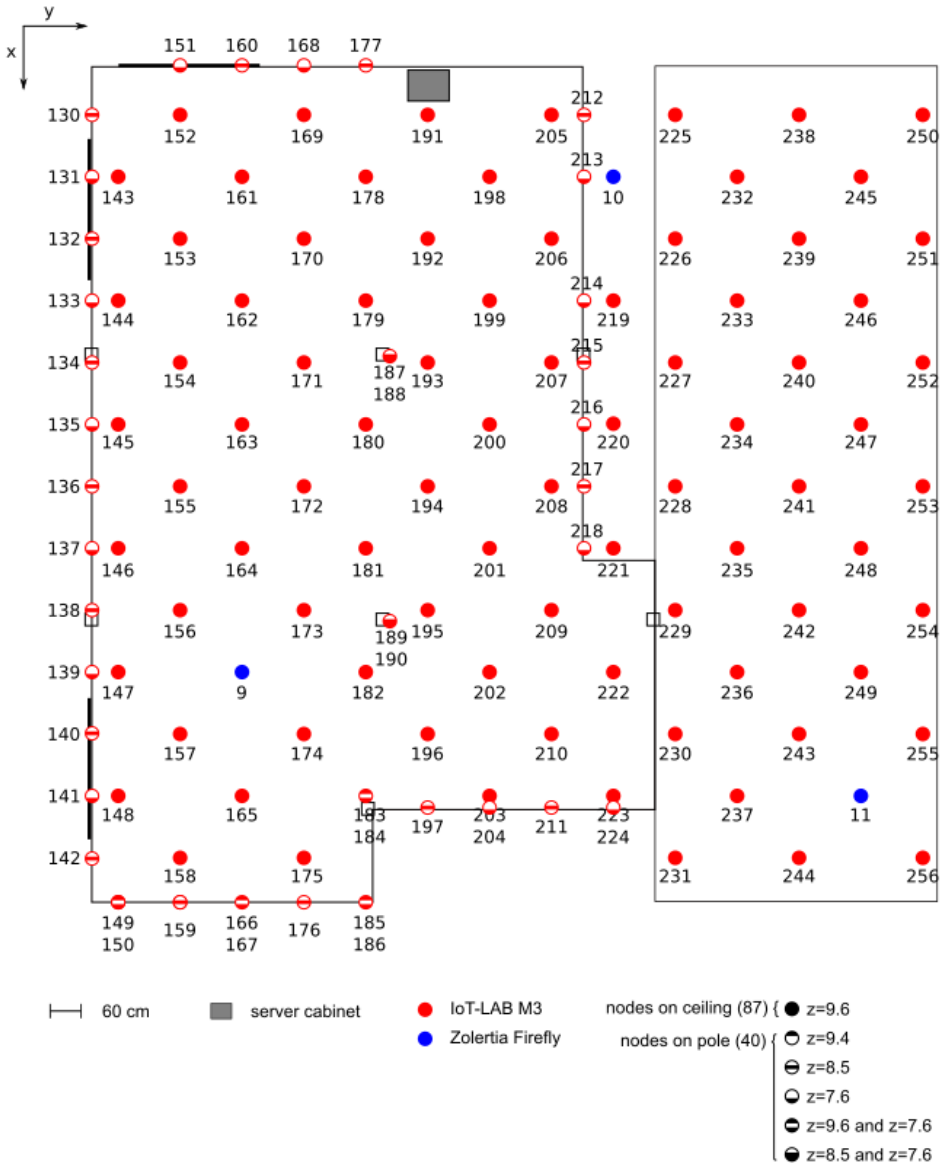


Figure 3.6: figure shows the Iot-Lab Lille server topology ; shows also the node type and the node IDs placed the network

Chapter 4

Problem Formalization and System Structure

The problem must be formally stated to explore ways to model monitor selection problem (MSP), build techniques to provide MSP solutions, and design the architecture of the monitoring system. The formal description of the MSP problem will be presented and modeled in distributed systems in this section. This definition offers a high-level overview of the problem and details a list of components. The selection monitor model must address communications and natural characteristics of wireless sensor networks. Furthermore, the debugging system should demonstrate which predicates are being targeted and which application or network protocol will be monitored on the top of the model. There are two options for monitor placement: global at the sink node or local at a local cluster within the network. The monitor node for global placement might be located remote from the predicate detection process. It means that monitor nodes must be placed in certain locations across the network in order to accurately detect errors at a low cost. Thus, the second option is local, in which monitor nodes are placed in a subset of local nodes within the network. Sending more data to the sink can result in lost messages or data collisions with the requested data. What is important is that some WSN applications/protocols do not require information from the entire network to evaluate the correctness of their behaviour. Users are more interested in checking the status of these applications in some part of the network, such as the status of neighbouring nodes. Therefore, few research have been conducted to formalize/study selection monitors in WSNs, and this chapter formalises the problem.

Various decisions would need to be considered to simulate and deploy the

algorithms. There are four primary options:(i) what is the ideal way to model the problem in a wireless sensor network infrastructure; (ii) what type of predicate and protocol will be monitored; (iii) which operating system should be used as a platform for algorithms; (iv) when these algorithms are tested, and how are they configured; (v) which simulator will be implemented; and (vi) on what kinds of testbeds will the algorithms be implemented? Therefore, this chapter will go through the aforementioned queries in detail and the experimental setup and methods.Other sections address some of the answers, such as the predicates evaluation and configuration settings.

4.1 Objectives

To ensure that the application executing on a wireless sensor network platform satisfies its requirements, runtime checks in the form of global invariants are used to track correct execution. When a violation occurs, an exception is raised. The decision to raise an exception is performed in-network, where all the state information is stored. In a sense, one or more nodes in the network act as monitors, where the network state information is collected and invariants are stored. In the presence of the network information, monitors continually evaluate the invariants to detect any execution deviation.

Deciding on the nodes to act as monitors is the problem we are focusing on in this study. The MSP is critical in a wireless sensor network. There are three main parameters to consider when selecting in-network monitors: (i) network traffic and (ii) predicate detection latency (ii) accuracy. For network traffic, at one extreme, there exists only a single monitor (similar to the case of traditional distributed systems), and all nodes route their individual state to the monitor (typically the sink). This collection process can be optimized using standard collection protocols, e.g., CTP [53, 56, 70]. The sink (i.e., monitor) aggregates the information and evaluates the predicate(s). The issue with this scheme is that there is a large amount of traffic that is routed in the network toward the sink, and nodes closer to the sink use up a lot of their energy budget for this activity. Moreover, owing to the data collection latency, the predicate detection latency is large as well. Overall, the network traffic has $O(N)$ complexity, where N is the number of nodes in the network, and the detection latency is $O(\Delta)$, where Δ is the network diameter.

All the nodes in the network can act as monitors, and any one of them can raise an exception when a system deviation is detected. Here, owing to the proximity of the monitors to network traffic, the detection latency is minimized. However, the main issue is that there is an enormous amount of traffic as all the nodes need to disseminate their state to all the other nodes. However, energy

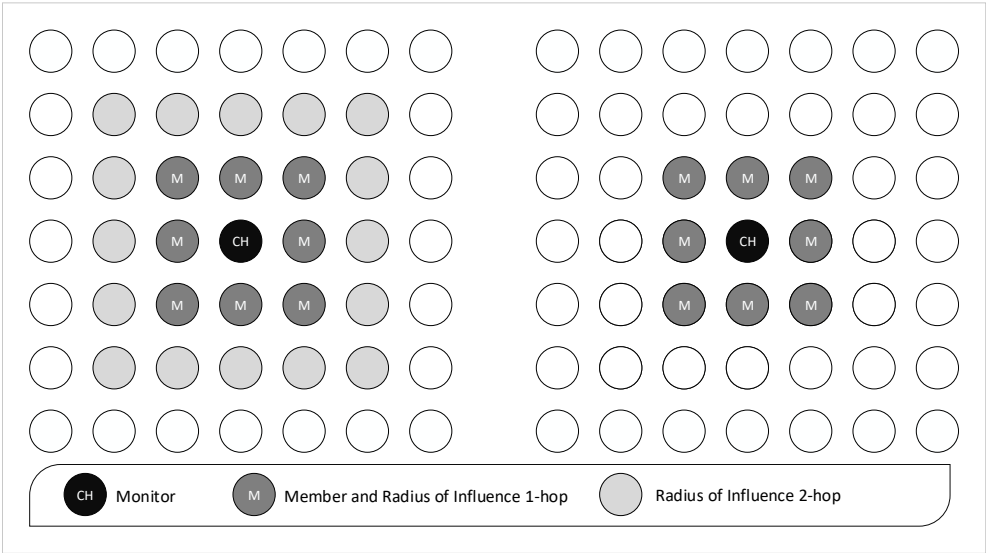


Figure 4.1: Overview of the radius of influence principle by the implementation of the clustering technique to select monitors in the network.

consumption is expected to be more balanced than that in the previous case. In this monitor selection scheme, the network traffic has O(N^2) complexity, and the predicate detection latency is O(1).

Monitors are required such that (i) the network traffic is localized (rather than network-wide); i.e., network traffic complexity is better than O(N), and (ii) the detection latency is better than O(Δ) see figure 4.1.

4.2 Complexity and Proof

Before we formalize the MSP, we provide the following definition:

Given a network $G = (V, E)$, a predicate ϕ defined over G, a monitor $m \in V$ that monitors ϕ over a set of nodes M, we say that ϕ has an *influence diameter* of δ if the value of ϕ at m is such that M defines a δ-hop neighborhood.

For example, monitoring that a TDMA protocol is working properly will involve collecting data over a two-hop neighborhood. Thus, the correctness condition (i.e., collision freedom) for TDMA has an influence diameter of 2.

Now, we define the MSP.

[MSP] Given a network $G = (V, E)$, a predicate ϕ of influence diameter δ that needs to be evaluated, and an integer $\mathcal{M}$, the MSP is as follows:

Select a set of nodes (i.e., monitors) V' such that

1. $V' \subseteq V$
2. $|V'| \leq \mathcal{M}$
3. $\forall u, u' \in V' \cdot \text{distance}(u, u') \geq \delta$
4. $\forall u \notin V', \exists u' \in V' \cdot \text{distance}(u, u') < \delta$

Informally, the MSP specifies that to evaluate a predicate of influence δ, then control messages do not need to travel more than δ hops to reach a monitor (condition 4). Furthermore, to reduce monitor redundancy (i.e., control traffic), monitors should not be close to each other (condition 3). [NP membership] MSP is in NP. To prove this, we need to show the correctness of V' in polynomial time. So, given an instance of MSP as described, and a solution set V', the verification is performed as follows: (i) The first two conditions are trivially verified. (ii) For the third condition, we need to show that the distance between any pair of vertices $(u, u'), u, u' \in V'$ is more than δ. We verify this by constructing a spanning tree of depth δ, rooted at u, and by doing a depth-first traversal on G. Then, for every path from u to any leaf vertex, we need to check if u' is on the path. If this is positive, V' is not a solution. Else, the process is repeated for every other vertex pair. This verification is achieved in $O(|V|^2)$.

Finally, the last condition is verified in a similar way to the third condition by constructing a spanning tree of depth δ, rooted at a vertex $u \notin V'$. If there exists no path from u to a leaf node that contains a vertex $u' \in V'$, V' is not a solution to MSP. Else, the process is repeated for all other $u \notin V'$. This is achieved in $O(|V|^2)$. [NP hard] MSP is NP-hard. To prove this, we reduce the minimum dominating set (MDS) problem to MSP. We present the MDS problem. [MDS] Given a graph $G = (U, \hat{E})$ and a positive integer K, the MDS problem is to find set U' such that

- $|U'| \leq K$
- $\forall u \notin U', \exists u' \in U' \cdot (u, u') \in \hat{E}$

With this definition of MDS, the mapping between MDS and MSP is as follows:

- $U \mapsto V, U' \mapsto V', \hat{E} \mapsto E$
- $\delta \mapsto 2$
- $\mathcal{K} \mapsto \mathcal{M}$

Now, we need to prove that a solution for MSP exists if and only if a solution to MDS exists.

- $\Rightarrow$ Let U' be the solution to MDS with a graph $G = (U, \hat{E})$, and V' be the solution to the MSP problem for a graph $G = (V, E)$, under the mapping defined previously such that $V' = U'$ with $\delta = 1$. We need to show that V' is a valid solution for MSP. As U' is a solution of MDS, $U' \subseteq U$ and $|U'| \leq \mathcal{K}$, then, under mapping, $V' \subseteq V$ and $|V'| \leq \mathcal{M}$. Moreover, as U' is a solution to MDS, $\forall u \in U \setminus U', \exists u' \in U' \cdot \text{distance}(u, u') = 1 \leq \delta$, which implies that V' satisfies the third condition too. Finally, consider an edge $(v, v') \in E, v, v' \notin U'$. Now, consider the case where $\exists m, n \in U', (m, v), (n, v') \in E, (m, v'), (n, v) \notin E$. Then, $\text{distance}(m, n) = 3 > \delta$. Thus, under mapping, V' is a valid solution to MSP.

- $\Leftarrow$ Let V' be the solution to MSP with a graph $G = (V, E)$ and U' be a solution of the MDS problem for graph $G = (U, \hat{E})$. Using the previous mapping, we have $U' = V'$, with $\delta = 1$. Now, we show that U' is a valid solution to MDS. As V' is a solution of MSP, $V' \subseteq V$ and $|V'| \leq \mathcal{M}$, then, under mapping, $U' \subseteq U$ and $|U'| \leq \mathcal{K}$. As the distance between a monitor node m and a non-monitor node n is at most δ, $\text{distance}(m, n) < \delta = 1$. Finally, as the minimum distance between two monitor nodes is at least $\delta + 1$, U' is a valid solution for MDS.

[NP-completeness] The MSP (Definition 1) is NP-complete.

This follows from Lemmas 1 and 2.

One typical approach to construct this in WSN, i.e., heuristic, to solve this MSP problem is to execute a clustering algorithm, where CHs act as monitors.What is important to observe here is that the diameter of the clusters formed will be of the order of δ the influence diameter of the predicate to be monitored.Furthermore, given the application's objective was to detect a TDMA time-slot collision within a 2-hop range, the parameter was specified to δ=2 in order to build a clustering that fulfilled the requirements (condition 3&4).More importantly, after the clustering is complete, a number of dissemination algorithms or protocols can be implemented on top of the clustering layer to expand the monitor's data collection range.A monitor can request 1-hop data instead of 2-hop data if the user prefers it, as shown and detailed in the results and implementation.Figure 4.2 illustrates the monitor selection based on the distance factor =2 and the above requirements.

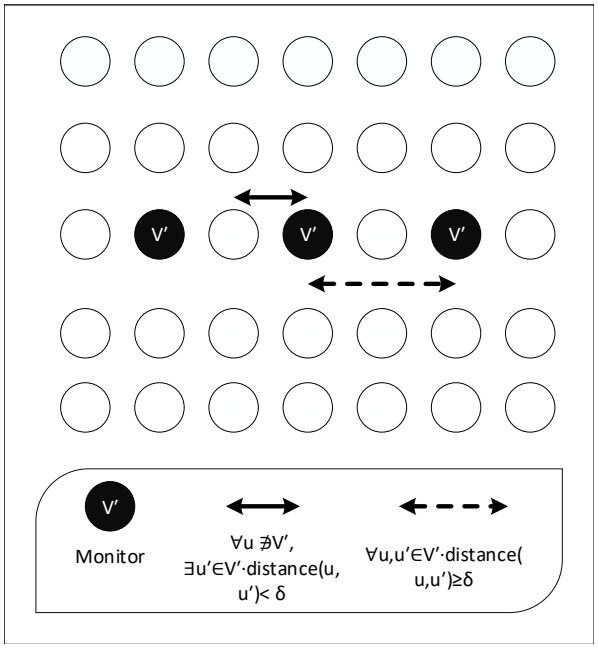


Figure 4.2: Overview of the network (Selection of monitors based on the distance factor δ =2)

4.2.1 Predicate Detection Requirements

Overall, the problem of predicate contains the following key steps:

1. Predicate specification.
2. Monitor selection.
3. Data dissemination to monitors.
4. Predicate evaluation.
5. Result notification.

After the monitors have been selected, before the application program has started, the predicates to be monitored are stored at the monitors. At specific times, nodes will disseminate predicate-relevant information to their respective monitors. Thus, a key component of the predicate detection problem is data dissemination. Once the relevant data have been captured by the monitors, the evaluation of the predicate(s) occurs. If a violation occurs, it is then reported to the sink or monitor.

In the next sections, we preview a case study that we have developed to showcase the applicability of our framework. The case study focuses on the TDMA assignment problem.

The model detection monitors suit the type of predicates and can locally evaluate low-cost predicates with specific distances. We developed a heuristic algorithm to implement our evaluation and solve the problem. A heuristic algorithm is constructed by compensating on optimality, reliability, accuracy, and completeness for efficiency to solve a problem faster and more efficiently than traditional methods. Implementation was conducted by including WSN libraries and resources such as Contiki[3] operating system developing our selection of a subset of monitors in WSN networks. The intended monitor selection for a grid topology network is demonstrated in Figure 4.3 .

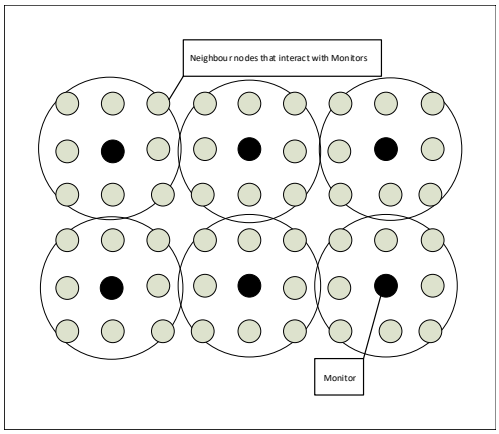


Figure 4.3: 9x6 network that forms the proposed selection monitor algorithm. Each node interacts directly with the corresponding monitor.

4.3 Predicates

There are several aspects that must be considered when deciding how to evaluate predicates. Any predicate evaluation architectures are going to be better than other implementations based on what the purpose is. In addition, the requirements of sensor networks (such as the minimization of energy use) need to be taken into account when designing with predicate evaluation considerations such as accuracy. This section will briefly illustrate the predicates and define the types of predicates that the system will evaluate.

4.3.1 Predicates types

Several aspects must be considered to decide on how to evaluate predicates. Any predicate evaluation architecture is going to be better than other implementations based on what the purpose is. In addition, requirements of sensor networks (such as the minimization of energy use) need to be considered while designing predicate evaluation system. Moreover, Accuracy is in important factor to detect these violations. This section will illustrate predicates and

their types that the system will evaluate.

Most of the study focuses on global predicates that are perhaps not as beneficial for WSNs but typically useful for distributed systems [50, 51, 141]. Initially, a sensor network can experience local problems. By local, it implies that only a single subset of the network information is available to a node in the network. In this case, the project concentrate on the surrounding neighbors of the predicate evaluating node. It implies that examining the evaluation of that predicate locally is excluded when requiring a local problem to be evaluated globally. This is complicated because when a predicate is locally evaluated, energy saving can be an issue. However, the first big decision is that local predicates are the priority instead of relying on global predicates to explore any possible energy saving option.

Definition 7 Global Predicate*: A (P) predicate that acts on some S global state where S is a mapping to some data on that node from a node id.*

Definition 8 Local Predicate*: A (P) predicate evaluated on the j node where the* $N(j, n) \subseteq S$ *accessible state includes information about any n-hop j neighborhood. Where* $N(j, n)$ *is a function returning the condition of all nodes in the n hops of j.*

The second determination is to focus on the stability of the detected predicate. There is a difference between stable predicates that stay true and unstable predicates whose true value will differ, as discussed earlier. This analysis is crucial because it will impact the structure of the algorithm that operates to check stable predicates. An instance of this is taking global state snapshots and evaluating the predicate against what is predicted. However, detailed recording of the system's traces is necessary for unstable predicates [18]. The project concentrates on the simplified issue of stable predicates due to the limited resources of sensor nodes. This is mostly because complicated programs seem to need more instructions to do more tasks and the firmware on the motes is small in size [6].

Definition 9 Application Predicate*:A predicate which is evaluated in the state in which an application is present. This may include collecting sensor data or software variables and evaluating them.*

Definition 10 Network Predicate*: A predicate that is evaluated over network connections. Examples include finding violations or collisions when none could have occurred, or ensuring that multi-hop communications loops are not present.*

Traditional predicate properties have been concentrated so far. This study was

intended to add two new classes of predicates. The authors discussed program traces in the abstract in the previous study; these traces are merely occurrences that contribute to any eventual state traces. Challenges that might occur when designing sensor network applications are (i) hooks to detect traces and (ii) capturing them with minimal available resources. The results can be minimized by understanding what happens in the application independently from the network. Network predicates would include checking for network events of the low-level MAC layer, and additional data for packets sent from the node would also be required. Application predicates can mean that the available data can be checked instantaneously. This study focuses solely on them because of the simplicity, but the good results that application predicates can provide.

In ideal worlds, all the previous studies exist where assumptions including such "no messages are lost" [50] are made. Unfortunately, this is not the case in the real world, although without energy-costing protocols such as 802.11 using a MAC protocol based on CSMA/CA [54], the MAC layer and the application layer will do a lot to reduce packet loss[25], and it is not possible to guarantee a certain degree of reliability. Thus, it is important to remember that it would have a certain degree of accuracy each time a predicate is evaluated because information may have been missing or incorrect data were used on the way to its target. Accuracy is a critical aspect for accurate predicate evaluation, which is not typically obtained and would notify the system user rather than getting messages on a regular basis. Also, in a flawless WSN environment, inaccurate predicate evaluation results may exist.

In summary, this study focuses on the evaluation of stable and local predicates. These predicates often deal solely with data about the mote-based application and not the communication with which it is associated. There should be an emphasis on evaluating predicates as accurately as possible with a low amount of energy. Moreover, delay must be addressed, so that the user can alert as soon as possible when a violation occurs.

4.3.2 Predicates Language

There were two possible solutions considered when determining how would evaluate predicates once data was obtained. The first option was for the system developer to hardcode the checking into the code and provide a response library, the second step was to create a predicate language that was easier to read and write for the system's user. Using hard checks written in C would have been a more effective way of checking the predicate, similar to the approach used by HSend[63], which parsed C source code and generated C code in a special comment block for the predicates. A failure of the predicate evaluation to

hardcode may lead to a large inflation in the firmware and if system developers want to change, add or remove a predicate, the whole firmware version in the network needs to be updated [39, 85].

Another advantage of writing and developing a new script or language is that firmware will have a size less than the first approach. This is not the case, however, because both can be optimized to obtain less firmware space. Since the development of a new language or script takes more time and more effort to achieve the main goals, this project's system thus uses the first method to write predicates by parsing C source code in order to write the predicates.

4.3.3 Predicates Considerations and Focus

The TDMA MAC protocol ensures that no node is within 2-hop of any node with the same slot. In this case, this project applied the predicates to evaluate these violations. In this case, there are many ways to evaluate predicates, and below steps are one of the scenarios. Figure 4.4 shows the overall requirement of predicates that need to be evaluated and implemented in WSN as such:

1. A message from the monitor to its member nodes to that asking to check slot collisions.(This is a case when a monitor is required to ask for information needed to evaluate predicates)

2. A message from the member node to each of its 1-hop neighbors asking the slot information. If the dissemination manager requires 2 hops of data to evaluate predicates, the member node will reply with the necessary information to the monitor.

3. Each 1-hop neighbor needs to send a message back to the member node that requested the data if the dissemination manager set to 2-hop.

4. The monitor wait to gather all the information from its all members and then evaluate predicates. This would imply that the monitor node would need to know who is in its two-hop neighborhood, including the monitor. To set up members and CHs in the network, the main part of this study is to implement a selection of monitor algorithms with dissemination algorithm to propagate the data.

5. The monitor (CH) will report the result of the predicate.

$$
\begin{array}{l}
\forall n1, n2 \\
\quad n2 \in 2hop(n1) \wedge \\
\quad n1 \in 2hop(n2) \Rightarrow \\
\qquad slot(n1) \neq slot(n2) \\
\\
\text{OR} \\
\forall n1, n2 \\
\quad slot(n1) \neq slot(n2) \Rightarrow \\
\qquad n1 \in 2hopN(n2) \wedge \\
\qquad n2 \in 2hop(n1) \\
\\
\text{OR} \\
\forall n \in Monitors \\
\quad n' \in Neighbours(n, 2) \\
\qquad slot(n) \neq slot(n')
\end{array}
$$

Figure 4.4: Check that no two neighbors have the same slot (2-hop information on Clustering).

4.3.4 Predicate Structure and Messages Optimisation

Optimizations attributable to the design of the predicate should be taken into consideration when monitoring predicates in the network to minimize the amount of data requested.In this project, a monitor expected multiple data to be collected across the network to monitor the predicates. In this work a monitor expected to have multiple data gathered a crossed the network in order to monitor the predicates. The form of the P predicate, in other words, is $P = P1 \wedge P2 \wedge \ldots \wedge Pn$. If one P_i is false, the consequence of the predicate P is false. This indicates that the statements could theoretically be evaluated in an order that first evaluates the Pi that needs the least resources and last evaluates the Pj that requires the most resources.

In order to eliminate energy consumption, the structure of predicates should mainly use minimal messages.Any of the monitoring systems collect predicates at the sink that will not be of much value as the information is sent to the sink in an event-based protocol or periodically requested. However, where the monitors are numerous in the network(Flat architcutre and Selected monitors) and locally gather data, the optimization concept will work perfectly.

It will make logical sense to enforce such an optimization for certain predicates, such as if there is a conjunctive predicate that requires 1-hop data and 15-hop data. For example, a 1-hop predicate is mostly violated, which would then reduce the amount of data needed to monitor 15-hops in the network. Even then, this is not always needed by any monitoring system where the predicates

differ mostly depending on the predicates that target a particular network data. In this work, the distance needed by the case study to gather data and monitor predicates is supposed to be one or two hops and a maximum of 3 hopes.

When monitoring the network, other factors may be considered or applied to this optimization, such as the global knowledge of the network against local predicates. A csae, for instance, where a predicate $P = Plocal \wedge Pnetwork$ where *Plocal* does not need network knowledge, has the potential to optimize the network request if *Plocal* return false. After all, implementing this function will raise the size of the firmware by adding further predicates to optimize the predicates structure. Furthermore, the existence of the predicates that this work attempts to have such a feature is not important.

4.3.5 TDMA Protocol : Case study

Time division multiple access (TDMA) has been chosen as a case study owing to being a non-trivial problem. The protocol chosen for monitoring is a receiver-based channel allocation protocol [127]. The receiver-based channel allocation aimed to minimize the number of channels to eliminate interference. The authors of [127] defined the concept of *receiver interference* to reduce the chances of interference during message transmission. Then, to eliminate interference, every receiver must be assigned a channel that is different from all of its interfering receivers' channels.

However, the algorithm implemented as an understanding of how the allocation of time-slots performs, which means that the algorithm implemented on nodes as senders is for simplicity in our case. This study does not improve or enhance the performance of the TDMA algorithm as it only tries to detect the accrued violations during the allocation of assignment slots. The case study implemented in different channels of radio communications did not conflict with other channels, such as the selection of monitors proposed or the dissemination protocols[1] . There are several TDMA algorithms proposed by other researchers aimed at enhancing and improving the performance of channel assignment [129].

Different TDMA algorithms can affect the detection performance as this affects the total number of allocation changes that occurred. This implies that few TDMA algorithms contributed to the faster allocation of time-slots while others focused on how the completeness was optimally implemented [129].

The case study was carried out separately as an application level to see the TDMA channel assignments perfectly distributed. The algorithm is mainly

[1] `http://contiki.sourceforge.net/docs/2.6/a00018.html`

used to assign node channels to ensure that no node has the same slot before; this is the form of predicates to detect slot collision within two hops of any node.

For completeness, we provide the main steps of the algorithm and they are as follows:

- Assign the smallest numbered channel to each node at first.
- A broadcast with its ID and channel currently assigned is transmitted every round by every node.
- The neighbor array is modified when a message is received, and the node receiving the message assigns its channel as the lowest channel not assigned to any neighbor. In the same round, two neighbors are not permitted to change their channel; therefore, a tie breaker is done, and the lower ID node is allowed to change their channel.
- The node broadcasts its ID and channel after choosing a channel.
- The process is repeated until each node can select a smaller channel than their current channel.

When the algorithm was first introduced and implemented, it meant that the node did not have the same channel as any of its 1-hop neighbors. This did not guarantee that any node's 1-hop neighbors would have different channels (as TDMA require to ensure that no collisions happen). This is the kind of error that would have helped detect in our contribution by running predicates. To ensure that the channel allocation was sufficient for TDMA, instead of nodes, simply send their own channel allocation to their neighbors. It also included the assignment of one of the hop neighbors that it knows about in that message. This allowed the nodes to receive information on their 2-hop neighborhoods, which they based their decision on which channel to assign to themselves. In other words, it is important to evaluate the information for each node with 1-hop in the network to assign or change the time-slot. To detect these errors locally and efficiently, the network protocol should be implemented as explained earlier in section 4.2.

4.3.6 Monitoring Ventilation System

Consider a demand-driven ventilation system for a building. A Heating, Ventilation, and Air-conditioning Controller (HVAC) receives periodic CO2 ,temperature and pressure measurements from sensors. However, there may be additional limits in monitoring proper HVAC operation. A non-uniform

temperature level in the building, for example, could indicate a faulty HVAC system. This condition can be examined as follows: **Any temperature differential between two sampling locations must be kept below a certain threshold.**

The transportation of perishable goods is another typical case[65]. RFID tags in use today can only track the position of objects at specific points in the supply chain. WSN nodes could allow for continuous, fine-grained monitoring of storage conditions in warehouses and throughout transport. When packages are in a container, for example, a nonuniform weight can cause stability issues during transportation. Similar to our scenario, WSN nodes can be placed to ensure that the difference in load between any two sampling points in a container stays below a certain threshold. The predicates in the previous examples are i) global, i.e., they cannot be evaluated solely on the state of a node, and ii) they may change over time. Several application scenarios, such as factory automation and health-care, are expected to have similar requirements for distributed monitoring of global predicates[133]. Furthermore, as WSNs encourage decentralized and increasingly independent applications, the mechanisms that ensure their proper operation in every situation must also become more distributed. Rather than the TDMA protocol, this section investigates our contribution to another application that requires monitoring. Instead of monitoring network predicates such time-slot values between nodes, we monitor temperature values between nodes. In terms of gathering the necessary data between nodes at a lower cost, both of these applications have similar requirements.

As previously demonstrated, monitoring local predicates can be accomplished just by disseminating the truth value of the constituting predicates. Because each predicate is a function of only one node variable, we may determine this value locally. This is not valid for distributed predicates, which have a (single) predicate that includes several node variables. Therefor, the predicate's truth value can no longer be established locally, and actual attribute values must be disseminated. As in the previous case, we begin by presenting an example to help the reader understand the rationale behind our strategy and how we build the structure (Clustering) protocol. It is worth noting that the previous statements are correct when we evaluate predicates from the TDMA protocol. Figure 4.5 shows the predicates that check temperatures on two nodes.

$$
\begin{aligned}
&\forall n1, n2 \\
&\quad n2 \in 2hop(n1) \wedge \\
&\quad n1 \in 2hop(n2) \Rightarrow \\
&\qquad temperature(n1) - temperature(n2) < T
\end{aligned}
$$

Figure 4.5: Check that no two neighbors have the same temperature (2-hop information in the network).

To detect violations, all combinations of temperature readings at any two nodes should be gathered and considered. If the worst-case scenario, i.e. the two nodes with the highest temperature difference, is identified, this step is unnecessary. If this value is less than the threshold, the invariant is satisfied, as any other pair of nodes has a smaller temperature differential. The invariant is violated otherwise. The highest temperature difference is determined by the two nodes sensing the maximum and minimum temperature in this predicate.

The number of system-wide maximum and minimum values required to establish the worst-case combination must therefore be included in the monitor for distributed predicates. If these values do not result in a violation, the predicate must be applied. The fundamental concept is not new: in some ways, comparing the worst-case combination of maximum and minimum values at n nodes to a threshold resembles applying discrepancy functions to $(n + 1)$-dimensional spaces'[139]. But, while the latter tries to analytically describe how closely a structural model matches the actual data, we employ a similar idea in the energy-constrained situation of WSNs to capture global invariant violations with a smaller amount of data. There are similar monitoring models that evaluate invariants that are similar to the concepts stated above[56], and this section compares the state-of-the-art models with our contribution.

Chapter 5

Monitor Selection Implementations

5.1 Introduction

Large scale ad hoc wireless networks pose self-configuration and management difficulties. Allowing all the nodes in a large network to transmit their data to a centralized base station would rapidly reduce resources of nodes owing to the long-distance and multi-hop design of the transmission and result in a network conflict.

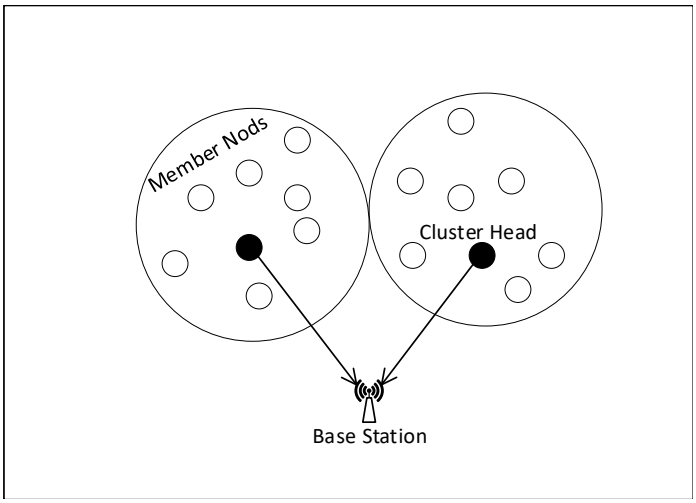


Figure 5.1: Demonstration of wireless sensor networks clustering

Clustering is a common technique to achieve effective and flexible network management to easily distribute network control [10, 17, 21, 32, 61, 109].By allowing localization of communications, clustering saves resources and elimin-

ates network contention. These data are aggregated by CHs into a manageable set of information (Fig. 5.1). Only CHs, not all nodes, will communicate over long distances with the base station; hierarchical clustering, i.e., clustering recursively over the CHs of a lower level, will reduce this burden further.

A clustering solution can combine various properties to allow effective and scalable network control. Moreover, the network can generate clusters of roughly equal size with minimal overlapping between them. Equally sized clusters are advantageous because they allow for an equal distribution of resources (e.g., data collection, aggregation, and storage load) between CHs, ensuring that no CH is overloaded or underutilized. As a node that operates in several clusters absorbs more resources owing to transmission to multiple CHs, there should be as little overlapping between clusters as possible for energy efficiency and lost messages.

To evaluate predicates of the TDMA protocol, this section introduces a clustering mechanism that suits the proposed solution. It begins by outlining problems that may arise while developing any cluster in WSN and requirements that must be met to form a cluster. Initially, the model for implementing such requirements that are required to form a clustering algorithm was defined and clarified. The remainder of the section delves into specifics of the heuristic algorithm implemented.What is crucial to note about this research is that it is not intended to build a new clustering algorithm to the state-of-the-art or compare it to other clustering algorithms; instead, it is largely focused on solving the MSP outlined in section 4.2.

5.2 Related Work

Several protocols for the clustering of wireless networks have recently been proposed [10, 17, 21, 32, 61, 109]. The max-min D-cluster [10] splits networks into clusters of d-hop. The algorithm does not guarantee non-overlapping clustering after forming, and, in the worst case scenario, the number of clusters created may be the same as the number of nodes in the network (for a connected network).
Clubs [109] form one-hop clusters: if two CHs are within the range of one hop, all clusters will drop and the process of selecting CHs based on random timeouts will be performed. Clubs do not meet hop distance and non-overlapping clustering properties: CHs can communicate only to their 1-hop members, and members can have many CHs.
LEACH [61]generates 1-hop clusters as well. By integrating the randomized rotation of high-energy CH locations among the nodes, the energy load of becoming a CH is evenly distributed among the nodes. Based on this prob-

abilistic rotation function, nodes nominate themselves as CHs and broadcast their actions. Each non-cluster-head node selects a CH that considers the least amount of communication resources. As a result, the algorithm fails to meet the criterion for a solution in which any node will collapse to another CH at any time, causing predicates to become unstable and require continuous evaluation.

To solve this problem, the authors[61] suggest the use of separate code division multiple access (CDMA) spreading codes for each cluster; nevertheless, the CDMA mechanism is not available on most sensor network platforms (e.g., Mica2).

First, the method in[17] determines the rooted spanning tree of the network and then creates ideal clusters from the subtrees. It establishes a limit on the number of clusters that can be built, and the convergence time is based on the order of the diameter of the network. It is fault-tolerant to node failures/joins on a local level, although in certain pathological situations, it can cause the whole network to re-cluster.

The algorithm mentioned in [21] produces clusters, in which all the nodes within R = 2 hops of a CH belong to that CH, and the farthest distance of each node from its CH is 3.5R hops, for a given value of R.

The closest work that comes close to this implementation is FLOC[31], in which the algorithm uses the same approach as LEACH. The method ensures that nodes do not interact with other clusters. The algorithm, on the other hand, is unlikely to have the same requirements in terms of CH distance and CH to member distance. Members of the CH can maintain distances of 1 and 2 hops at the same time. Similar to CHs, which may be more than 3 hops apart from other CHs, where the distance between members and CHs in this study must meet the required distance that was mentioned earlier. FLOCK relies on timers to construct the cluster, which considers some time to complete and uses these timers in all the nodes, while the method in this study depends on the base station to begin the clustering process. While implementing our algorithm on a 10-by-10 grid it does not consider the distance between the monitors as monitors can be far away in FLOCK, which does not fulfil our requirement (distance between CHs;$m >= 2$). Thus, our clustering process ensure clusters that consider a distance of 1 hop between a member to a CH and thus prevent overlapping internally inside the clusters. Moreover, it ensures that each cluster in the network have the same number of nodes in a grid topology.

5.3 Clustering

The most essential energy-saving approach is clustering. The sensor nodes in this technique are arranged into groups known as clusters. Cluster members

are the normal nodes in the cluster, and a Cluster Head (CH) is selected among those nodes[62]. In clustered WSNs, there are two types of traffic: intra-cluster traffic and inter-cluster traffic. The cluster members detect real-world inputs and transmit the detected value to the cluster's CH. The CH receives and aggregates data in order to remove redundant data before transmitting aggregated data to the CH directly or via intermediate CHs.The cluster members are unable to send data directly to BS; instead, they send it to the CH, who then forwards it to BS. Clustering has the following benefits: improved bandwidth usage, reduced overhead, increased connectivity, stabilized network topology, reduced delay, effective load balancing, and a smaller routing table. The CHs that are closer to the BS consume more energy and drain it more quickly than the CHs that are farther away from the BS.Due to intra-cluster traffic from its own cluster members, data aggregation, and inter-cluster traffic from other CHs transmitting data to BS, CHs closer to BS are heavily loaded with traffic. As a result, network connection is disrupted, and coverage limitations in clusters closer to the BS are created. This condition is known as a hot spot or unequal clusters[11].The goals of unequal clustering are the same as those of equal clustering, but with certain extra functions. WSN nodes are grouped with varying aims dependent on application requirements. The most typical goals of unequal clustering are energy saving and the elimination of overlapping. Some of the other goals are listed below.

5.3.1 Scalability

Sensor nodes are deployed in vast numbers in the real world, ranging from hundreds to thousands, depending on the application requirements.The ability to operate with such a large number of sensor nodes should be considered while designing routing strategies. When a node in a cluster wants to send data to a node in another cluster, the nodes should be aware of the receiving cluster head's information (CH). By splitting the sensing field into several levels, each layer is further divided into a number of clusters, hierarchical architecture allows scalability in large scale WSNs[80].This increases scalability while also shrinking the routing table's size.

5.3.2 Fault-Tolerant

Sensors are deployed in hostile environments in a variety of applications (e.g., sensors are dropped from helicopters), and these nodes are at an increased risk of physical damage and node failure. Fault-tolerant nodes are vital in specific applications where the loss of certain sensor data might lead to disaster. Clustering is an excellent method for creating a fault-tolerant and secure WSN[154]. The self-organizing WSN controls the fault by re-clustering the

network. The process of re-clustering not only raises the resource burden but also disturbs current operations. Re-clustering, allocating backup CH, depute CH, or rotating CH produces fault-tolerance with the added benefit of effective load balancing[61].

5.3.3 Data Aggregation/ fusion

Because a high number of sensors observe the same data in the physical world, data redundancy is more frequent. Data aggregation is a good approach to avoid redundant data transmission while simultaneously reducing the number of transmissions. This is a signal processing approach that combines all received packets into a single output packet. This approach boosts common data while suppressing undesired noise[83]. CH aggregates the data received from its cluster members in WSN and sends the aggregated data to (BS) through single hop or multi-hop. As a result, the amount of transmissions and the network's overall load are greatly reduced.

5.3.4 Load balancing

Load balancing is critical for extending the network's lifecycle. Load balancing is a key problem in which CHs are chosen from the network's available nodes[117]. To avoid a hot spot problem, the CHs must have a uniform load distribution. Unequal clustering ensures a uniform load distribution, with each CH consuming roughly the same amount of energy. As a result, a more energy-efficient network is possible.

5.3.5 Increased lifetime

The main goal of clustering is to extend the network lifetime as much as possible. For real-time applications, maximizing the network lifetime is critical because the sensors are energy constrained. Selecting nodes as CHs with more neighbor nodes can reduce intra-cluster communication[148].To maximize lifetime, the clustering and routing processes might be combined[66].Clustering extends the life of a WSN by correctly rotating CHs across cluster members. Sleep modes and cluster maintenance procedures can also be used to extend the network's lifetime.

5.4 Clustering approaches

WSNs are often made up of a large number of sensor nodes, ranging from hundreds to thousands. Clustering is a good approach to organize a large number of nodes in a way that evenly distributes the load and eliminates hot spots. This section discusses a detailed literature review of reported uneven

clustering techniques. These algorithms are divided into three categories: Clustering algorithm that is probabilistic, deterministic, and preset:

5.4.1 Probabilistic clustering algorithm

One of the primary goals of the probabilistic clustering technique is to increase the network lifetime. In this method, the algorithm chooses the CH at random. This clustering algorithm is found to be simple, with near-optimal overhead, rapid convergence, and low energy consumption. Time complexity and message complexity should be low for an energy efficient clustering algorithm.Random techniques and hybrid methods are two types of probabilistic methodologies. Random methods are easy and achieve near-optimal overhead by randomly selecting CHs and forming them. Hybrid approaches combine random methods with some characteristics like residual energy or distance to BS to correctly balance the clusters. Hybrid techniques are iterative or competitive in nature, which adds to the message and time complexity. Random techniques are addressed in the following sections:

Random approaches

Energy Driven Unequal Clustering (EDUC). In heterogeneous WSNs, EDUC is a distributed algorithm that decreases energy usage and avoids hot spot issues[151]. This technique efficiently regulates the energy consumption of nodes within a cluster in order to prevent energy depletion. It uses an unequal clustering technique and a CH rotation mechanism based on energy. Cluster creation and data collection are the two aspects of EDUC. The cluster construction phase consists of two stages: CH competition and cluster creation. During the lifetime of the network, each node only functions as CH once. To avoid intra-traffic collisions, the CHs develops a TDMA schedule for its cluster members. CHs are chosen at random, and the energy level for CH rotation is calculated precisely. The data is sent immediately to the BS by the CHs.In many real-time applications, the assumption of single-hop communication is not feasible. For multi-hop networks, this strategy is ineffective because the energy threshold must be extremely accurate. EDUC, in comparison to LEACH and HEED, can extend the network lifetime.

Hybrid approaches in clustering algorithm

Constructing Optimal Clustering Architecture (COCA).COCA is a distributed, scalable unequal clustering technique that addresses the logical challenges of unequal and overlapping clustering algorithms in homogeneous sensor networks[97]. For approximate equalization of energy usage over the whole network, OCA develops optimal clustering architecture, energy-aware CH rotation, and rout-

ing. COCA employs a technique in which the number of clusters per unit area grows as the distance to the BS decreases. Multi hop routing between clusters saves more energy, resulting in the elimination of the hot spot problem. All nodes transmit residual energy information with their neighbors for CH selection.They are proclaimed CH by the node with the most remaining energy. Each CH selects a neighboring CH as a routing candidate at random, and the CH with the largest residual energy is chosen as the final routing CH. It's a simple and efficient approach that uses less energy and has a maximum lifetime that's twice or three times that of UCR.

5.4.2 Deterministic approaches

In comparison to probabilistic approaches, deterministic procedures select CHs using defined metrics. Traditional metrics such as residual energy, node degree, estimated residual energy, distance to BS, node centrality, and others are commonly used and measured locally.Normally, this information is updated by exchanging messages with its neighbors. Because clusters with elected CHs are more controllable, this strategy is referred to as deterministic approaches. Weight-based, fuzzy-based, heuristic-based, and compound clustering algorithms are the four types of clustering algorithms.A weight is calculated at each node depending on some metrics such as residual energy, node degree, distance to BS, and so on in a weight-based approach. The cluster head is chosen from among the nodes with the least weight.In instances when there are greater uncertainties, fuzzy logic is employed to select CHs. On the basis of fuzzy input parameters, the cluster head is selected. The input fuzzy parameters can be residual energy, node degree, distance to BS, node centrality, and so on, with cluster size and probability of becoming CHs as the output fuzzy parameters.WSN's clustering challenge is an NP-hard problem, and proposed algorithms are well-suited to finding optimal solutions to NP-hard problems. In recent years, heuristic-based clustering algorithms have shown to be the most effective method for determining CHs and cluster size.WSN uses various types of optimization algorithms, including Genetic Algorithm (GA), Ant Colony Optimization (ACO), Artificial Bee Colony Optimization (ABC) Optimization, Particle Swarm Optimization (PSO), Bacterial Foliage Algorithm (BFA), Differential Evolution (DE), Simulated Annealing, and others.To obtain greater performance, each algorithm defines several metrics in the fitness function. Heuristic methods are centralized, with a central authority such as BS overseeing all network operations. Some systems use agent nodes to work in a distributed setting in exceptional instances. In clustering approaches, the compound algorithm employs many metrics such as connected graphs, Sierpinski triangles, and so on.

Weight based clustering algorithms

Improved Energy Aware Distributed Unequal Clustering for heterogeneous WSN (Improved EADUC)[58].The goal of improved EADUC is to extend its lifetime and avoid the hot spot problem in multi-hop heterogeneous WSNs.It's commonly utilized in applications that require continuous data collection. It varies from EADUC[150] in that it considers node degree in addition to residual energy and distance to BS when estimating competition radius.The degree of each node is provided to ensure sufficient energy balance in the network. CHs are chosen in Improved EADUC based on the ratio of average energy of adjacent nodes to residual energy of the node. Three metrics are used to calculate the competition radius: residual energy, distance to BS, and node degree. Improved EADUC selects relay nodes based on energy, whereas EADUC selects relay nodes based on distance to BS. The same cluster arrangement is used for several rounds, which eliminates re-clustering overhead and reduces energy consumption.

Fuzzy based clustering algorithm

Because of the inconsistencies that exist in the WSN environment, a number of protocols rely on fuzzy logic to create clustering more effeicnt. Low computational complexity, better flexibility, lower development costs, less memory, shorter design time, and fault tolerance are only a few of the benefits of fuzzy over traditional approaches. In WSN, fuzzy logic is applied to effectively select CHs. The input parameters for fuzzy logic for CH selection are residual energy, distance to BS, distance from neighbors, node degree, centrality and excepted residual energy , the output parameters are CH selection probability and cluster size. To achieve energy efficiency in the clustering method, various fuzzy-based algorithms are proposed.

Fuzzy logic Based Unequal Clustering (FBUC).FBUC is another distributed clustering algorithm that focuses on how to join cluster members using the CH[100].The enhanced version of EAUCF[14] is FBUC. A probabilistic technique is used to select tentative CHs. The competition radius is determined using fuzzy logic after the tentative CHs have been chosen. Residual energy, distance to BS, and node degree are the fuzzy input factors for determining competition radius. The final CHs are chosen using residual energy and node degree. To effectively utilize the energy and maximize the network lifetime, the nodes join the CH based on the CH degree and distance from the CH. When compared to LEACH,it has the longest lifespan.

Heuristic based clustering algorithm

Genetic Algorithm based Energy-Efficient Adaptive Clustering Hierarchical Protocol (GAEEP)[100]. To reduce energy usage, GAEEP utilizes a Genetic Algorithm (GA) to calculate the number and location of CHs.The entire operation is run through several rounds, each of which has two phases: setup and steady state. BS executes GA during the setup phase to identify the best number of CHs and CH position. In the steady state phase, the inter-cluster routing from CH to BS takes place. When a node is considerably closer to BS than any CH, the node transmits data to BS directly. To avoid collisions caused by intra-cluster communication, each CH utilizes a TDMA schedule and assigns slots to its cluster members. CHs uses CDMA code to reduce energy usage and reduce inter-cluster collision.

Compound clustering algorithm

Energy-Efficient Routing Algorithm Based on Unequal Clustering and Connected Graph in Wireless Sensor Networks (UCCGRA)[146].UCCGRA is a distributed strategy that uses two ways to improve energy efficiency: cluster head election and cluster routing. To reduce intra-cluster traffic and eliminate the hot spot problem, a voting method is applied to generate unequal size clusters and smaller clusters are built near the BS. Topology, residual energy, and transmission power are used to select CHs. Connected graph based routing uses the geographic position of the nodes for inter-cluster multi-hop communication. UCCGRA effectively distributes the load and lowers energy usage.

Preset clustering algorithm

The clusters or CHs, as well as their position, are predetermined in the Preset clustering algorithm before implementation into the real world. The following are some of the major disadvantages of this approach: static and network characteristics are not taken into account. Wireless link/node failures in WSN result in frequent network topology changes. These algorithms are unsuitable for use in real time. The goal of UCS is to maximize network longevity by dissipating the same amount of energy in all CHs[130].

5.5 Model

We designed our network as an ad hoc wireless system where nodes were designed for undirected graph topology. Wireless nodes can transmit and receive messages through the radio model. Each node can transmit messages at a maximum range, and nodes capable of determining which nodes are close

to its range by performing multiple techniques. Nodes that are considered to be the closest and most reliable to transmit and receive messages are communicated through their inner-band *(i-band)*. Nodes that communicate far from their inner-band, where nodes might lose messages, are considered outer-band *(o-band)*. This wireless network scenario has been studied and observed in [145],[152], and [31].

Nodes can determine the distance factor and the place of neighboring nodes by applying different techniques in the wireless sensor network. This helps execute our heuristic algorithm by selecting one or more of these techniques, e.g.:

- Wireless sensor nodes help monitor the signal strength of the received message[60]. This method provides the capacity to measure the distance from the sender. The signal strength method uses $\frac{1}{1+d^2}$, where d represents the distance from the transmitter, i-band nodes have a transmission power range of [0.5, 1], and o-band nodes have a range of [0.2, 0.5]. These mechanisms and ranges provide the capacity to determine whether the algorithm can interact with each node and take further action.This scenario is used in our research to determine who falls into each cluster and to locate other possible CHs.

- Nodes can use other techniques, such as time-of-flight-of-audio, ultrasound signals, or time-to-leave messaging technique[24],[123],[88]. The *ttl* mechanism provides the ability to each node that receives a message to check if it is greater than 1; if it is greater than 1, *ttl* counts reduce and the neighbor receives a message with the current *ttl* state information. Nodes will keep broadcasting messages until the *ttl* message state has a value of 0 where it stops sending messages and takes further action on who receives the message. We use this technique on our algorithm to determine the number of hop counts, where the message travels along a certain path to do some action. In our heuristic clustering algorithm, the message travels from a CH to locate another CH in a specific path.

- In a wireless sensor network, there are other methods, e.g., underlying localization service. This gives the capacity to broadcast distance information for each node in the network [113],[118].

We have implemented the first two techniques mentioned above to develop our heuristic clustering algorithm. The method enables us to define which nodes are in range by measuring the signal strength and to determine which node falls within the i-band and o-band. The second method is used to determine the distance between each CH in the system. This provides us the opportunity

to select CHs based on the aforementioned requirements. For each cluster, both the $(i-band)$ and $(o-band)$ will be defined as members in our clustering, and $(o-band)$ will build a $(border-node)$ to the adjacent node to allocate the next level of CH. To allocate the next level of cluster head, our *ttl* message will travel in a specific route and is valued at three hops. Nodes that tagged $(i-band)$,$(o-band)$ and $(border-node)$ are useful to the ttl message in the first level, where the *ttl* message will consider the route from the first cluster head and consider which nodes would count their *ttl* message from the tagged nodes.Further details on these mechanisms are provided in the sections below.

```
% Sink Set-up Process
init:: start →
    if (j = sink)  then
    bcast⟨SinkMessage⟩;
j.idle ∧recv⟨SinkMessage⟩ →
    if (j ∈ i-band of sink) then
        j.status:=j.i-band;
        bcast⟨IbandMessage⟩;
    else if (j ∈ i-band of sink) then
        j.status:=j.o-band;
    fi;

% Set-up First-Level of cluster by Sink Process
j.idle ∧recv⟨IbandMessage⟩ →
    record ::= start
    unicast.send⟨AssignCh⟩;
j.iband ∧recv⟨AssignCh⟩ →
        j.status := j.CH;
        cluster.level := cluster.level+1;
        bcast⟨ch-msg⟩;
if (j ∈ i-band of ch) ∧ recv⟨ch-msg⟩ then
    j.cluser-id := i;
    j.status := j.i-band;
else if(j ∈ i-band of ch) ∧ recv⟨ch-msg⟩ then
    j.cluser-id := i;
fi;

%BorderNode
j ∈ i-band of ch →
    bcast⟨BorderMesg⟩;
 j ∈ i-band of i-band ∧ recv⟨BorderMesg⟩ →
    j.status := BorderNode;
%Set − upnextLevelofCluster
j.ch→
    bcast⟨j, Msg-id⟩;
    Msg-id:=Msg-id+1;
(j.idle ∨ j.i-band) ∧ recv⟨j, Msg-id⟩ →
    if (Msg-id = Msg-id+3) then
        j.status := j.CH;
        bcast⟨ch-msg⟩;
```

Figure 5.2: Heuristic algorithm to select subset of monitors.

5.6 Clustering Problem Statement and Requirements

In this section, we design and implement a distributed, hierarchical and local heuristic for cluster construction such that:

- The distance between each CH in the network must be ≥ 2 hops. CHs in our study are monitors that evaluate predicates locally. Logically, increasing the number of monitors will increase the network traffic that we are dealing with essentially. Hence our clustering algorithm guarantees that no cluster is allocated next to each other to achieve high rate of correctly evaluated predicates and reduce the loss of messages. Moreover, the radius of influence where predicates are evaluated needs to be measured.

- Each member node must travel no more than one-hop distance to reach its CH. As we are using a TDMA as case study which our predicates checks time slot within 2-hops. Each CH in a grid topology will have a maximum of 8 nodes as members.

- Each cluster in the system has a unique node, known as the CH, where it acts as the cluster leader.

- Every node that falls as *i-band* and *o-band* in a cluster will act as members of that cluster.

- Each node in the network is either a member node or a CH node. This ensures that no nodes will be isolated and will not participate after the clustering process has finished forming [90].

- Each member node in the network must join a single cluster instead of multiple clusters. This prevent each cluster from overlapping with unnecessary messages with other clusters. Overlapping nodes generate more messages, resulting in more power consumption, which is why the clustering network must ensure that this problem is resolved for the effectiveness of the clustering network [149]. The problem of overlapping in WSN is NP-hard; thus, more details is discussed in section 4.2,

5.7 Program

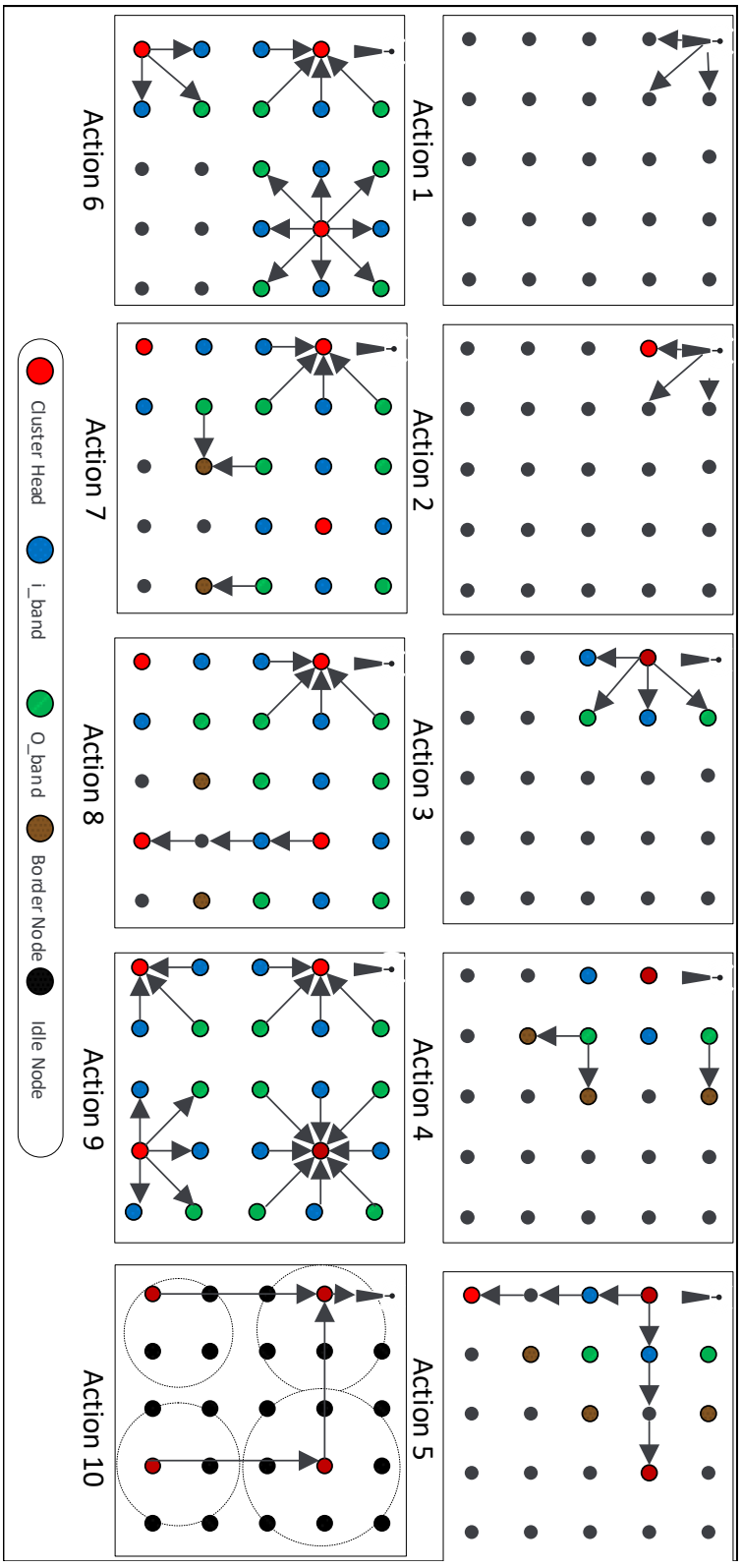


Figure 5.3: Example of a 5x5 network demonstrates the phases of clustering

For each node in the network, there are only two variables that are important to determine the clustering stage: *(status)* and *(cluster-id)*. The *(status)* variable always hold the values of the *(o-band),(i-band),(border-node),(i-band), (idle)* means (inner-band) where nodes are considered to be the nearest nodes to the desired node, where *(o-band)* also means the (outer-band) where it also falls within the range of the desired node but is farthest than *(i-band)* in our experiments *(o-band)* will be the nodes in the corner.*(border-node)* means nodes that border the *(o-band)* nodes where the next cluster level can be constructed and established after the first cluster has been created. *(border-node)* prevents the ttl message count by being counted through it so that message travels along a specific route. *(idle-node)* means node status' variable = $\perp$. The clustering algorithm based on the network hierarchy levels. Levels of hierarchy begin from the sink, where sink considered to be the first cluster level. Our clustering deployment depends on the sink to start and establish a network to build the first level of the cluster. The program consists mostly of five actions, as shown in Figure 5.2, however in order to construct the entire network and larger network size, some of these actions must be repeated, as shown in Figure 5.3, which demonstrates step-by-step actions to form the cluster.Note that this diagram was created for the purpose of illustration, where it will be simple to clarify the algorithm in this manner.

Action 1 As described above, the *(sink)* acts as an initial node at the beginning and is considered to be the first level of the cluster hierarchy. The sink broadcasts and measures the received signal strength indicator (RSSI) of its neighbors. The sink acts as hierarchical clustering tier 0.Initially, the sink will send a message *(SinkMessage)* to the neighbors.The int-variables *(i-band)* and *(o-band)* were set before the network started.There is also a set of time periods during which the node sends a request to collect data, update the neighbours statues list, and measure RSSI from neighbours.

Action 2 After the broadcasting of the sink to measure the RSSI, the sink will allocate the CH based on the measurement of the RSSI. The nodes in the south and east will fall into the *(i-band)*.Both nodes are eligible to be actual first CHs in the network. The sink will send a single message to one of these nodes to avoid collisions and prevent both nodes from being chosen as cluster heads. In our example and experiment, the node in the south is selected as the CH. The new CH acts as the hierarchical clustering level 1.Because this action necessitates receiving *(SinkMessage)* from neighbor nodes,*(idle)* nodes that have not yet set is will alter their status to *(i-band)* or *(o-band)* .

Action 3 Once the node in the south of the sink is chosen as the cluster head, it begins to broadcast the signal strength of its neighbors. As the nodes in

the east and south are the closest to the cluster head, they will fall into the *(i-band)*.To begin the initial message *(AssignCh)*, a list of nodes requested by the sink is recorded *(record::= start)*. These nodes are tagged as *(i-band)* and their *(status)* variable changed if a node is tagged as CH. While the nodes in the northeast and southeast will fall into the *(o-band)*, their variables changed as their signal is detected as *(o-band)* nodes. The *(i-band)* and*(o-band)* nodes will receive a notification *(c-head-msg)* from the CH and join as members to that CH.

Action 4 Only nodes that fall into the range of *(i-band)* and *(o-band)* nodes receive a *(BorderMesg)* message from the *(o-band)* nodes at this phase. These nodes will update their status to the border node. This stage helps us select the next CH that meets our clustering requirements by performing out the following actions.

Action 5 CH broadcasts *ttl* message to its neighbors at this point to allocate the next CH. The *ttl* message moves to count the number of hops in a directed direction. The hop number is set to move 3 hops by default, and if the message cannot count three hops according to our condensation and theory, the message will stop counting at 2 hops. Nodes that are their *statues* are *(o-band)* ,*(i-band)* and *(i-band)* would not receive this message, and *ttl* will change its counter. This prevents the *ttl* message from counting hops that are only accessible in its range as wireless sensor nodes have a limited radio range. This action allows us to make sure member nodes are at a distance of one hop from their CH. As there are three hops in the network, there are two new cluster heads assigned along the south and east directions of the old CH. If there are no hops in the network, *ttl* will allocate the cluster head at a distance of two hops, and the overall clustering will be different from what looks like in the example figure. As our clustering method is hierarchy-based, the two clusters will act as level 3.

Action 6 The new cluster head will repeat Action 3 individually to identify *(i-band)* and *(o-band)* nodes. While the old cluster begins working sensing from its own members. Members transmit their data by default by sending periodic unicast messages to the cluster head; however, the communication model between members and cluster heads can change slightly in our case when we implement the predicates. Few nodes will continue to update their status with new variables, particularly the one with the *(o-band)* status .

Action 7 At this step, action 4 is repeated to identify *(border-nodes)* separately for each cluster recently selected.This explained in Action 4&5.

Action 8 In our case, the CH along the northeast direction managed to select

the new CH before any other CH. This is essential for our clustering with regard to the communication to develop a clustering-based hierarchy. This new cluster will save its history to record hierarchy correspondence to the cluster that selected this new cluster.

Action 9 All clusters that have finished formation will begin to communicate and transmit their information from their members. While the new cluster head will repeat Action 3 individually to identify *(i-band)* and *(o-band)* nodes. The selection of CHs is completed at this stage.

Action 10 As the proposed algorithm is based on hierarchical clustering, each CH must communicate to other CHs. However, each CH transmits its data to the CH from the CH that selected it. This model reduces communication redundancy and power consumption, so that CHs do not receive the same copy of information from different sources. Nevertheless, this feature is a routing scenario where the user requires the ability to detect global predicates.

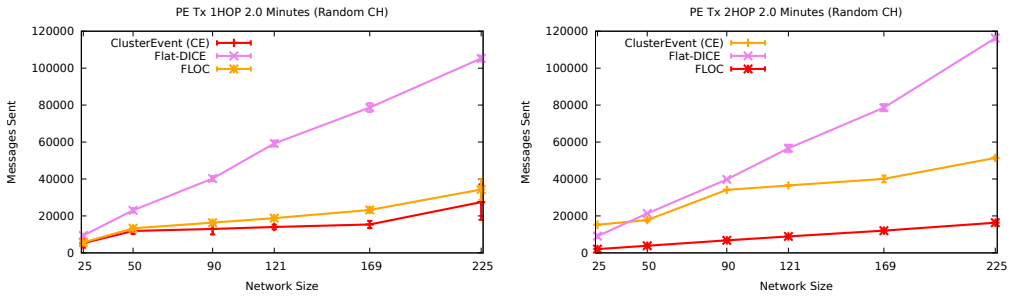
5.8 Network Topology and Clustering Comparison

This section compares the results of our proposed algorithm's implementation against the results of a different clustering approach[31] as well as comparison in different network topology. To have the best results, we first focused on implementing the most common clustering methods and settings in order to run and detect predicates. Unfortunately, most clustering techniques are incompatible with the predicates that the system monitors, by starting to use LEACH algorithm[61], which is one of the most common clustering algorithms. However, when clusters are formed, the cluster heads are unable to maintain stable connections with members for long enough to enable predicate evaluation. This is partly because the algorithm rotates and drops selected cluster heads after a period of time, which is the case when a cluster head is unable to gather sufficient information from the surrounding nodes. Other clustering algorithms cannot ensure that communication between clusters in a network will not overlap[109][10]. A data point can only be assigned to one cluster in traditional techniques. In fact, there are so many different forms of data that a single data point might be assigned to numerous categories, causing ground-truth clusters to overlap. Conventional clustering cannot function properly in this situation. On the other hand, the FLOC[31] clustering algorithm seems the closest clustering algorithm to ours and produces similar results in terms of the members distance that join the CHs. As a result, in subsequent sections, FLOC will be implemented and compared to our contributions.

For the sake of simplicity, as well as to capture the predicates given in sec-

tion4.3.The monitors are configured to collect 1-hop and 2-hop data from the neighboring nodes. We implemented the event-dissemination protocol for the FLOC algorithm in order to deliver more accurate results to our clustering algorithm, and the time to evaluate the predicates was set to 2 minutes.This also applies to the FLOC algorithm, which lacks any form of data dissemination mechanism to transmit the information to the monitors after forming the clusters.

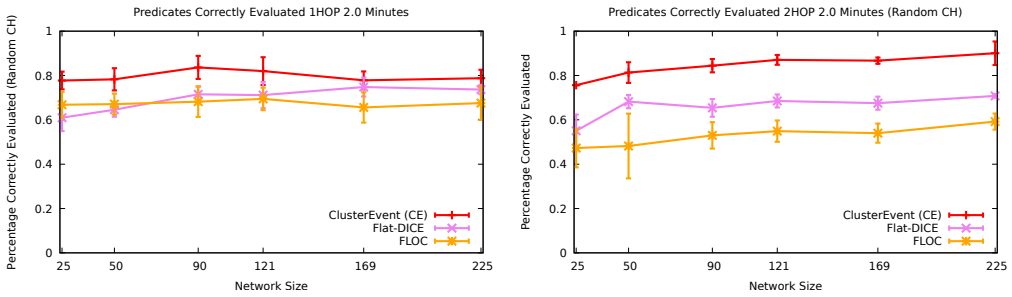
5.8.1 Grid topology



(a) Number of messages in 1-hop.

(b) Number of messages in 2-hop.

Figure 5.4: Predicates for evaluation and comparison between different clustering and flat architectures on the Grid topology with 1 and 2 hops.



(a) Correctly Evaluated in 2 minutes.

(b) Correctly Evaluated in 4 minutes.

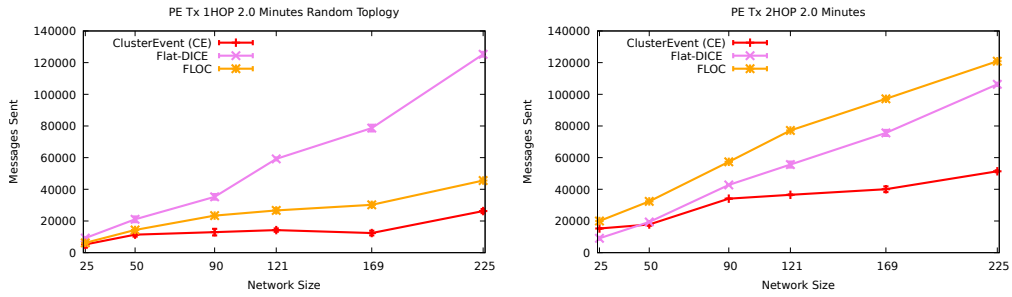
Figure 5.5: percentage of correctly evaluated predicates when monitors evaluate predicates using a 1-hop and comparison of another clustering and FLat architecture (Grid Topology)

.

It's worth noting that the clustering network changes shape each time we conduct a new experiment in FLOC, whereas our clustering produces the same shape in grid topology most of the time.The most important aspect of this finding is that FLOC may choose CHs that are close to each other or far from each other, implying that member nodes may fall in 2-hops depending on the final clustering result.As previously stated, FLOC simply requires one

requirement to establish the clustering: the member nodes must be $m \geq 2$ from the CHs.Our clustering algorithm, on the other hand, considers the distance between CHs in order to eliminate redundancy and cover the network predicates.This may have a different effect than our clustering algorithm, causing messages to increase slightly while overall performance decreases as shown in graph 5.4.FLOC provides the worst results for corrected evaluated predicates, which implies that if the predicates need to be evaluated within 1 or 2 hops, it will most likely miss part of the data that has to be evaluated or gathered.This is also shown when DICE (Flat-Gossiping Propagating) outperforms FLOC in terms of accurately evaluated predicates but consumes a lot of energy Graph 5.4 and 5.5.In FLAT, one or more "sweeps" of the whole network are sufficient to tell all nodes about the new local view if the update rate is low.If there are several concurrent changes, FLAT propagates them in an unstructured manner, with each potentially creating a new update and hence a new dissemination that competes with the others.This results in traffic that climbs linearly with the update rate, due to FLAT's inability to aggregate the changes.In comparison, the structure provided by the other clustering techniques, which is disadvantageous at low update rates, turns advantageous as the update rate increases: local view changes may be successfully aggregated in-network and across local clusters, significantly reducing traffic increase.To compare this, FLOC presents different cluster sizes each time the simulation is executed.On the other hand,our clustering algorithm is capable of balancing the number of clusters and members each time the simulation is run.This stabilizes the overall performance and reduces energy consumption in cases where some clusters have more member nodes, burdening the network and resulting in missed violations.When compared to other examples, our Cluster-Event approach is optimal for this scanrio because the distance between each cluster head (monitors) and its members are fixed, balancing overall performance.

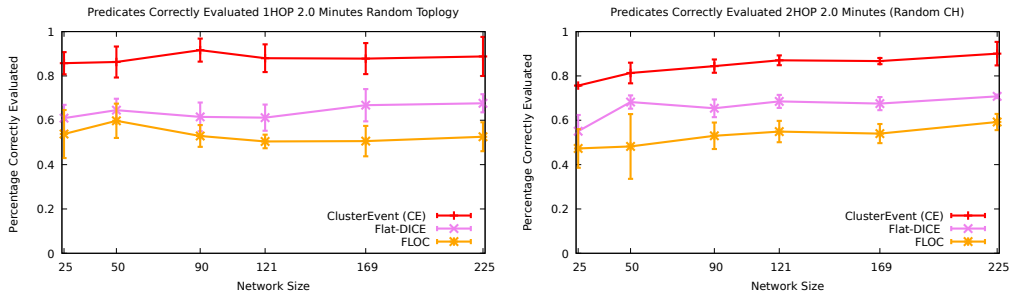
5.8.2 Random topology



(a) Number of messages in 1-hop.

(b) Number of messages in 2-hop.

Figure 5.6: Predicates for evaluation and comparison between different clustering and flat architectures on the Random topology with 1 and 2 hops.



(a) Correctly Evaluated in 2 minutes.

(b) Correctly Evaluated in 4 minutes.

Figure 5.7: percentage of correctly evaluated predicates when monitors evaluate predicates using a 12-hop and comparison of another clustering and FLat architecture(Random Topology).

This section compares the algorithm's implementation results to the results of another clustering approach. To achieve the best results, we focused first on the most common clustering algorithms and settings to apply to the predicates.The above scenario is implemented and compared, however the nodes in the network are placed differently.Instead of being implemented in a grid-like manner, the network is constructed at random, with nodes scattered .The purpose of this implementation is to validate the clustering algorithms when they are implemented in this manner and to see how the overall performance is affected.As previously stated, grid implementations were primarily implemented to provide more insightful analysis for the contributions.This section, in which network nodes are implemented at random, is another technique to examine how the overall performance is affected when most clustering algorithms take the number of hops and distance into account.

Similarly, graph 5.6 and 5.7 produce the same outcome and have the same overall energy consumption result and percentages of correctly evaluated as Grid topology results .Even when the topology is changed to random, the overall results remains the same, especially when using the Cluster-Event approach, which keeps the number of messages below two thousands.While the use of a DICE-like approach has increased slightly, it still lacks a centralization mechanism for gathering/disseminating predicates in the network.The polite-gossiping method, which is implemented in DICE, prevents duplicated communications. The data is propagated via the network to all nodes, but there is no channel or mechanism for sending the data to more nodes rather than a group of nodes.

Since FLOC can select more than 1-hop clustering randomly with each run, the size of the cluster in a random topology may increase in comparison to Cluster-Event, which has a maximum 1-hop clustering size.FLOC also has an

issue selecting CHs in a random topology where these CHs might be either far from each other or very close enough, and when the selecting mechanism does not prevent the network from carefully selecting CHs as it is in the Cluster-Event algorithm.This also supports the previous observation that these monitors will eventually lose part of the messages when there are more member nodes in a cluster.The natural mechanism of all nodes capturing the predicates is observed in the FLat-architecture, which outperforms FLOC in terms of accurately evaluated predicates(graph 5.7), whereas FLOC may miss messages due to the random distance factor and random topology.Our Cluster-Event technique is optimized for this scanrio, where the distance between each cluster head (monitors) and its members are fixed, balancing overall performance compared to other examples.The Cluster-Event also ensures FLat-architecture redundancy by ensuring maximum performance in capturing the most of messages.

Chapter 6

Implementation: Algorithms and Network Dissemination Protocols

The overall implementation of the dissemination protocols is described and explained in this chapter. These protocols are determined by the scenario used. The chapter begins with a list of scenarios in which predicates are gathered in a network. These scenarios primarily outline when and how data must be sent. The second part outlines and explains the details of the algorithms, as well as which algorithms are appropriate for which scenarios.

6.1 Predicates Evaluation Scenarios

With regard to the evaluation of predicates, this section explains how data communication works and which scenario will be implemented. To achieve this, we need to understand where to evaluate the information and when to disseminate the data once predicates propagate. This study focuses on evaluating predicates locally and do not need network knowledge where the sink (1- monitor) evaluates the data. However, this study implemented global predicates (flat protocols) for comparison to our proposed architectures, e.g., Flat-Polite-Gossip and Flat-Event, to analyze and provide more findings to understand how the selection of monitors and data dissemination impact the performance of the predicate evaluation system. On the other hand, predicates are going to be evaluated withing a subset of nodes in a network, and those nodes will act as monitors.

So far, we've discussed what we want to evaluate and how it will be evaluated;

but, we haven't addressed how data will get to its evaluation destination. The method of data evaluation is divided into two parts: where the predicate should be evaluated and when the data should be disseminated. There are two options for evaluating a predicate. Either all of the network state is collected in a single location designated such as the sink, most of the traditional algorithms such as global predicate evaluation would do. Alternatively, the predicate might be evaluated in the network at a specific target node such as monitors. We aim to look into both of these. When it comes to when the data should be disseminated, there are multiple approaches. The first and most straightforward option is to just send out the information on a regular basis. However, because periodic transmission of data may be ineffective, we will also consider event-based dissemination in this project. Finally, nodes such as monitors might make requests for information as needed. We look into all three of them. We look into all three of them. Most scenarios and network transmission libraries that interact together are explained in the following sections:

6.1.1 Cluster-Periodic based

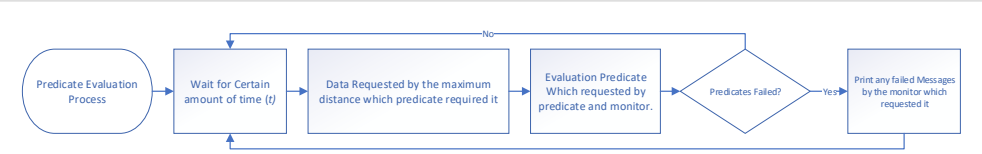


Figure 6.1: Predicate Evaluation of Cluster Periodic

The evaluation based on periodic monitoring is different from checking and determining when the data have changed. This implies that data will only be sent when requested to nodes; in this case, monitors (cluster-heads) will request data from their surrounding members. The concept is that when a node evaluates its predicates, it determines the maximum distance, for which data are required, and floods a message that several hops request for data, and each node replies along the route from which the request message originated. Then, it forgets all that information after a node evaluates all its predicates for that time. Similar to the local event system, members only store the required distance to neighbors and monitors that initiate the data request.

Another important factor for the evaluation of predicates is time limit (t) to collect data and print or send the result to the sink. These values of (t) are described in the simulation and testbed experimental setup section. In this process, a cluster member checks data in the requested hops and periodically sends the data to a monitor (cluster head) if requested by the monitor. As this algorithm considers information such as neighbors for each node, only members can evaluate their 1-hop neighbors if data requested are more than 1-hop and

send these data to the monitor to verify if any predicate has failed. Otherwise, the monitor sends a request to its surrounding members, which, in this case, will be a 1-hop data request. Figure 6.1 illustrates the process of evaluating predicates in the cluster architecture periodically

This scenario is one of the easiest and straightforward routing in a wireless sensor network, where a subset of nodes immediately requests data at a low cost of exchanging messages. Instead of requesting information, each node might broadcast it periodically. That does, however, have the downside of causing nodes to transmit information when they do not need to (an event-based dissemination problem). There is another issue that can cause problems when node clocks are synchronized perfectly. For example, nodes may regularly receive old information from nodes if they continue to send data intended for a previous round to be received only during the current predicate evaluation round (Duplicated messages issue). As this evaluation is based on the period factor, the monitor may receive old information where the old data have changed in a certain node, and the monitor has to wait for the next round to receive it (correctly evaluated predicates); this problem was managed to deal with in another sections and applied to most routing protocols as well.

Several issues are required to be addressed while using *(request by distance)* library in this predicate evaluation which described in section 6.2.1. After the data have been sent to request a message, how long this message will be expected to return? This cannot be delayed for long as the monitor evaluates the data received from other nodes and eventually has to print the result because (i) no information is available about the waiting time of the node's data and (ii) the data may be lost or never again obtained. The recommendation would be to wait for $2 \times D \times T_{send} + \epsilon$ amount of time units; D is the maximum distance of hops from where data are requested, T_{send} is the time to try and send a message, and ϵ is the buffer time to wait for. Therefore, T_{send} can be challenging owing to a lot of complex factors. For instance, the MAC protocol, which is in use, based on CSMA/CD or CSMA/CA can be accomplished. As such, the message attempting to be sent should be entered into a number of back-off periods before the channel is free and available [136]. There is no upper limit on the number of messages to be sent, which can help the process determine the maximum T_{send}. Thus, this implementation aims to use a fixed large value that is desirable and fair to deliver the return message. To further enhance this aspect, the amount of time should be related to the D function with regard to the number of hops and data requested if detailed information is required by the network.

Overall, the periodic evaluation scenario for cluster predicates contains the

following key steps:

- After the time required by the clustering algorithm to select monitors from cluster heads, the network will form a subset of monitors, and the monitors will evaluate data.
- The rest of the network nodes will function as members, and each member will be associated to their monitors.
- The monitor will periodically request data (1 hop ,depends on the manager of dissemination) from its neighbor (members).
- Each neighbor (1 hops) returns data to the corresponding monitor.
- The monitor will receive these data and wait for t (time period) to collect more data from other nodes.
- The monitor will evaluate predicates and subsequently print success or failed predicates.
- The periodic scenario for cluster predicates contains *(request by distance)* and *(flooding by distance)* algorithms to complete evaluation which described in section 6.2.1 ,6.2.2 .

6.1.2 Cluster-Event based

The most important part in the local evaluation of predicates in the network is to acknowledge that data have changed while implementing event-based techniques in the network. Owing to the absence of a connection to the OS and feedback from a memory section, the local data must be checked regularly for changes. To do this, nodes require to periodically check their data to compare the past history from the last sent message. The process must be aware of the value to be compared because WSN may compare floating point calculations that may produce different representations of the same results.

Moreover, it considers that few violations and data to be checked will be missed as event mechanisms trigger the event when data are checked periodically and found to have changed. In other words, node values can be changed while the evaluation is still being evaluated and compared, and the evaluation cycle has not yet been completed. They required considerable attention during the evaluation period, and with regard to the immediate value changes in the targeted nodes, a flood mechanism is used to transmit data with the required number of hops to be evaluated.

Therefore, when we apply, e.g., two hops of information, nodes will keep a

record of all the predicates within that distance, and they will flood their data when they change. An immediate change occurred in the adjusting node within three hops, and the node will not send its data. If we address this issue, it means that each node in the network should know predicate values of the entire network. Logically, it is expensive, and more messages need to be exchanged within the network; therefore, this problem is mostly solved in cluster monitoring architectures. For each node, the event-based protocol uses an algorithm to compare its values against the last sent message, where any change causes another network library to disseminate the data at a distance required by the evaluation manager.

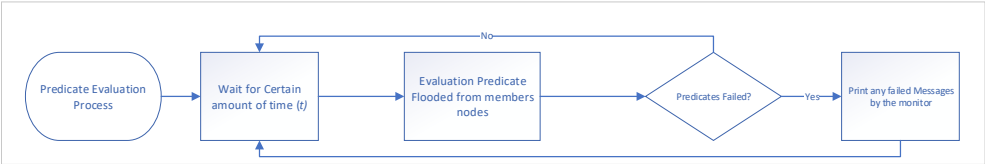


Figure 6.2: Predicate Evaluation of Cluster Event

If the event-based predicates process have started, members of each monitor, who are cluster-heads, check the their history from the last messages sent. Every node checks its data upon assigning a new slot to that node. Moreover, in order to check data in nodes, we implemented an algorithm within the local view manager, so that nodes can keep track of the most recent data sent. While the members check the history of their data, if the past information is the same for each current, the dissemination will not fire the broadcast communication *(flooding by distance)*. More details explain the data checking in section 6.2.3.

Once the predicate value is compared, the dissemination will fire only one member detecting a change. Once a member sends new data and triggers a change in its data, the monitor is included. Once a member detects violation, the member sends data to the monitor (cluster head). The monitor (CH) will analyze changes associated with the previous collected data and print a report of the event. The evaluation manager will consider a time period to gather all the data and start evaluating. In this scenario, Figure 6.2 illustrates the process of evaluating predicates on the monitor.

As the network built with a sink and hierarchical cluster, the monitor (cluster head) will forward this report as a copy to the monitor that appointed it during the clustering process. This ensures that the sink receives a copy of these events from monitors. Monitors that receive the actual evaluation report perform the evaluation process as normal. A copy of the event sent to the

sink is useful if the user is interested in determining which part of the system has the most or least number of violations. As many possibilities can be considered because our clustering technique affects the route of the propagation protocol, we think this segment needs further analysis and discussion for future studies. Therefore, without transmitting unnecessary messages to the sink, the monitors report the evaluation results. During the design of the clustering algorithm, this function was already built in. It's worth noting that the dissemination manager can disable the hierarchical clustering routing for forwarding violation reports.

Overall, the evaluation scenario of the cluster predicate events includes the following key steps:

- After the time required by the clustering algorithm to select monitors from cluster heads, the network will form a subset of monitors, and the monitors will evaluate data.
- The rest of the network nodes will function as members, and each member will be associated to its monitor.
- Every node in the network (member nodes) will periodically check its data depending on the t value entered before the initiation of network evaluation.
- Then, the nodes will flood data (*slot*) once the evaluation process detects a value different to the previous recorded value. The flood distance value depends on the value entered before the initiation of the evaluation process.
- This will be received by the monitor and wait for t to gather data.
- The monitor will evaluate predicates and subsequently print successful or failed predicates.
- The event scenario for cluster predicates contains *(event checking protocol)* and *(flooding by distance)* algorithms to complete evaluation which described in section 6.2.3 ,6.2.2 .

6.1.3 Flat-Event based

This basic algorithm (Flat-Event) is used at the beginning of the project to simulate certain situations where all the nodes are assumed to act as monitors. Before the initiation of monitoring, just as in cluster algorithms, this algorithm does not have selection criteria. However, in the cluster-event scenario, the al-

gorithm exhibit similarities in networking propagation. This algorithm has been implemented for comparative analysis and research purposes. The Flat-event dissemination protocol is similar to the cluster-event dissemination protocol, with the exception that the monitors are selected differently (CHs).

The sink was used to trigger monitoring, in which the sink transmitted the initial packet to every network node randomly. Once this packet is received by the node, it will begin functioning as a monitor and stay idle for 2 or 4 min to capture data from 1 to 2 hops. The remaining nodes will use n-hop flood propagation automatically to propagate their data. They will use *(flooding by distance)* propagation automatically to propagate their results. This network dissemination is identical to the aforementioned cluster-event scenario. This also refers to monitors that need their data to be disseminated in the event of data modification, where most of the network will act as monitors. To ensure that the nodes check their data to determine any changes, the scenario uses the *(flooding by distance)* algorithm.

While the Flat-Polite-Gossiping architecture is discussed in depth in this study [56], there is no monitor selection in the FLAT architectures as all nodes in the network are selected to be monitors. This scenario does not consider to be a part of this project as this has been implemented in previous studies. However, in this study, it has been provided to compare with other existing scenarios such as flat-polite-gossiping with cluster-event-based, cluster-periodic, and cluster-polite-gossiping.

Overall, the evaluation scenario of flat-event predicates includes the following key steps:

- To begin evaluation, the sink will broadcast the initial packet to all the nodes, and whoever receives it will act as a monitor.
- Then, the nodes will flood the data (*slot*) once the checking process detects a value different to the previous recorded value. The flood distance value depends on the value entered before evaluation starts.
- This will be received by the monitor and wait for t to gather data.
- The monitor will evaluate predicates and subsequently print successful or failed predicates.
- The event scenario for cluster predicates contains *(event checking protocol)* and *(flooding by distance)* algorithms to complete evaluation which described in section 6.2.3 ,6.2.2 .

6.1.4 Cluster-polite-Gossiping based and Flat-polite-gossiping based

The observation in terms of overhead communications at the beginning of the project is that the output of such a propagation protocol (Polite Gossiping communication)[94] is an efficient way to implement it to evaluate predicates at low cost. The purpose of polite–gossiping broadcast is to remove duplicate messages exchanged by wireless nodes. To pass messages through the network, the wireless sensor network mainly requires some forms of protocol to handle the broadcast messages. The nodes will not transmit a duplicate message in polite–gossiping broadcast if the data *(time-slot)* of the neighbour nodes node have not changed. In this way, whether the member node or a 2-hop node has not updated its data, these nodes will not send duplicate messages to the monitor. The polite fundamental is a generalization of trickle's polite gossip algorithm [94]. This scenario is nearly identical to the Cluster-Event scenario, but the way the nodes broadcast and transmit their data differs from the one we implemented. The goal of implementing this scenario is to compare it to other possible dissemination protocols and analyze any future improvements that could be proposed in order to implement more efficient propagation protocols. Both of these scenarios use the same Trickle method to propagate data, however the cluster and flat differ in that all nodes (flat) are monitors, whilst the cluster (subset of nodes) are monitors. It is worth noting that the Flat-polite-gossinping scenario is the one used in DICE[56] and is chosen here to compare it to other scenarios for better analysis.

6.1.5 Reply and Notifications

After evaluating predicates in the monitor and reaching the point of obtaining results, the monitor should take useful actions based on this information, so that the user can acknowledge the useful information that can be obtained. The first and most straightforward answer will be to notify the monitor of these results, irrespective of success or failure. Although this is the most detailed solution, it will consume a lot of time and energy as each node will need to deliver a message of response with any predicate it was evaluating.

Alternative options are either to send a message when a predicate fails or when a predicate is successful. That enables to send only a fraction of messages. Thus, there are variations that allow to provide a significant decision with regard to success or failure. First, how the conclusive result is represented? The systematic approach has the drawback of considering the outcome of a result as an *undefined* state. When a response has been received, the outcome may be transferred to a *failed* or *succeeded* state. It would need to assume the

inverse of the messages are waiting on (i.e., assume predicates missed or failed before receiving the successful message and vice versa) on successful or failed messages and change the state until the message is received. When messages are lost, this can lead to false alerts of success while waiting for messages of failure and false failures while waiting for messages of success.

The other aspect to be addressed is what will be reported and notified. Once successful messages are the chosen format of output, then what valuable information might they contain? It would assume the state in which the predicate was evaluated, but this information is not quite as important as the caused state in which a predicate failed. Another consideration is that providing a clear explanation of why an error message failed is complicated and will take more time than just evaluating the message. This suggests that reporting the state that caused the predicate to fail back to the monitor would be obviously better for a failure message and then conduct more analysis on it there.

As it considers a random failure to be a unusual event, the libraries were implemented to monitor only predicate failures. This will save energy in transmitting messages. However, where messages are missed, the network is considered to be vulnerable and does not reveal the full extent of failures (false indication of success). Moreover, the system can print successful predicates, and it is possible for users to hide or extend messages being evaluated.

6.2 Network Protocols and Data Transmission Algorithms

The network libraries that were implanted to evaluate predicates are explained in this section. This section describes the wireless sensor network libraries that are required to disseminate data across the monitoring system, as mentioned earlier. To evaluate these data, the monitor system must investigate when data must be propagated and how data arrived at the monitor. Each networking algorithm is designed to suit the appropriate scenario or infrastructure as described above; however, few networking algorithms can be used in multiple scenarios as long as the final outcome is simple and efficient in terms of data delivery and message loss.

6.2.1 Request Data by Distance

The application and structure of predicates require information from neighboring nodes, and the messages mostly need a way to travel over multiple hops in the network. We reviewed the Contiki libraries at first to see if any of the

existing primitive network could be used for this aim.

First, the mesh protocol is recognized to send data or a packet to a node in the network; therefore, sending these data back to the originator node is preferable and required in this purpose. Nevertheless, few problems occurred once this has been deployed in the network at the beginning, such as sending a single packet to the network at a time. This is how mesh operates, where it just holds a record of the last packet received on the network. This does not work and fits the network requirements as it works fine with one sender like the base station. Therefore, the network has several monitors, in which this case monitors act as senders. Consequently, mesh does not perform this task.

After evaluating and implementing several network libraries and sources, including Contiki libraries, the need to send packets to multiple hops must be in accordance with network objectives. Nevertheless, the implemented network algorithm is an API layer for the exchange and routing of messages. This communication layer can use a flood routing mechanism in the network, holding a distance variable (hop number) from the originator node. This enables the use of the same layer for multiple nodes in the network with different messages. The communication layer was developed and based on Contiki Rime communication libraries.

`stbroadcast` (Stubborn Broadcast) and `runicast` (Reliable UniCast) are communication libraries[1] that are used in this implantation. The main goal of Stubborn Broadcast is to send a request message (with-hop) in the network. If this request message is received by the other node, it will verify the message's time-to-live (ttl); if the ttl value is greater than 1, the message will decrease the value and send the message to the neighbors. It is also important to note that Stubborn Broadcast is used to broadcast a message at a specified rate, which allows the other nodes to receive a few copies if the message is lost or not received. The number of message repeats is being reduced to reduce the energy cost.

[1] runicast.h `http://contiki.sourceforge.net/docs/2.6/a01738.html`

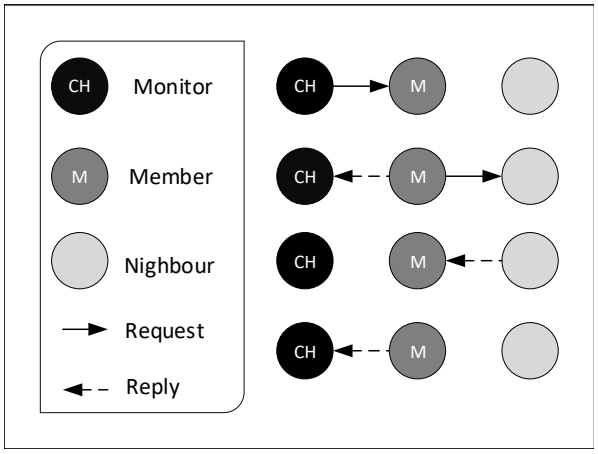


Figure 6.3: Overview of Hop-Request flooding algorithm (Request and Reply path) when N=2

TThe algorithm works as follows: (i) When the node receives the requested message from the source (monitor), the node that sends these data (node-id) will be saved and registered in the memory. Moreover, the node will be able to record the node that was originally requested (2-hop scenario). This information is useful while constructing a route as it is followed by replies rather than broadcasting the replies. (ii) Then, the node waits for a fixed period of time before sending a reply after this information has been stored. The reason for this is to prevent any collision as the messages requested have enough time to propagate. (iii) Then, reliable-unicast is used after the time has passed to reply messages to the recorded nodes, where information is returned using the route registered and requested. The processing of the requested data and the reply data is demonstrated in Figure 6.3. Reliable-unicast is mainly used to provide reliable transmission to reduce energy consumption, which will contain a single ack-message as well.

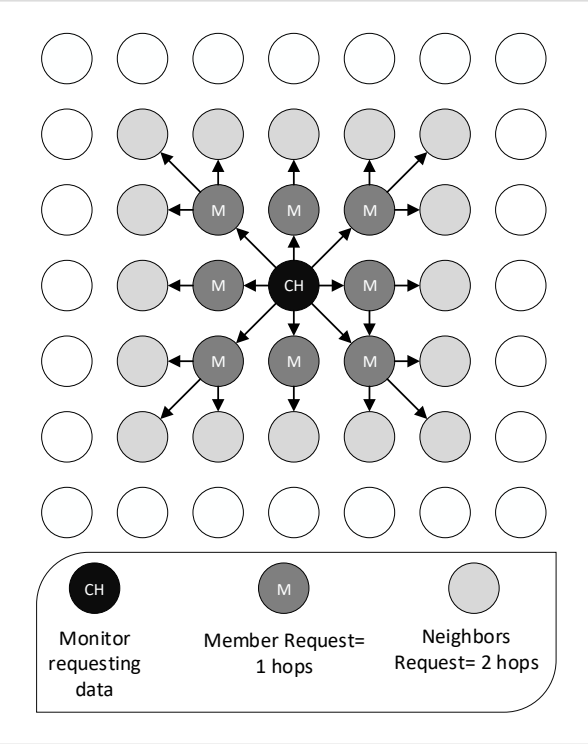


Figure 6.4: Overview of request by distance algorithm when N=2

Once the message is received by the node (with regard to the case of 2 hops) and this message is not the intended target of the requested node (monitor), the node will send the message to the sender node. The message will finally get back to the originator (monitor) who can deliver and evaluate the message. This is an effective way to keep the count of messages low by sending messages to the most effective node to return them back to the requested node (monitor). It is achieved by the number of hops the message requires to reach its destination. A node message with three hops left is a more effective route than a node message with just 2 hops left. When the predefined timer has ended, it will go to the evaluation manager, which performs evaluation, which will contain all the data gathered during the dissemination manager period is finished. Figure 6.4 shows the overall algorithm in a grid topology when a data request is made by the monitor.

Figure 6.5 shows the stages of the algorithm used to evaluate the predicates in this scenario. The algorithm is divided into four steps: first, the monitor will set a timer for the first message requested to be sent, which is the time between receiving a data request and sending it. Before initiating the evaluation system, the dissemination manager will normally set these variables.

Only nodes in the network that have been tagged *(j.status=monitor)* will initiate this process and broadcast the requested message. With the exception

of monitors, this will prevent all network nodes from initiating this procedure and duplicating the unnecessary messages. The monitor will broadcast messages for collecting data, with each message containing the following information;The maximum number of hops the dissemination manager is permitted to travel or request the data form is ($hops$) which is set to 0. We will consider the monitor to be the originator of this process because it will always start this message, and we will tag ($MonitorID$)to capture its address in order to send back the requested data. The ($msgID$) is used to ensure that the requested messages are not duplicated and that each message has its own unique number. When the hop value is greater than 1, the ($SenderID$) is responsible for registering the address values of nodes that are transmitting the desired messages from the $monitor/originator$. After sending these messages over the network, the monitor will keep track of them by incriminating the variable to +1 each time they are sent.

The second step involves other nodes receiving the broadcast taking the following actions;(a) The nodes will keep track of the nodes that register in the monitor records and record them. If the nodes have not yet been registered in the clustering process ($j \neq monitorLIST$), this will record them for monitoring. (b)The nodes will count the number of hops and keep track of the message ($hops < recHops$). (c)In order to track duplicate messages, the nodes will check and record message IDs ($id > recId$). The process will complete these steps by sending a *unicast* message to the neighbors, which will be forwarded to the next hop.

The process that follows deals with nodes that have received the previous message. If the messages are delivered to their destination, the process will generally check the record of the node IDs ($j = target$) and the hop count. If this is not the case, the node will unicast the message to the neighboring nodes while keeping the hop count in check$hops + 1$. In addition, if the hop manager detects that the hop distance is just one hop required for the predicates to travel, the network library will slightly change the way the data is transmitted. If the network uses clustering-monitoring mechanisms, the sending data will use a uni-cast format to transmit the predicates. This would help the network avoid more data (such as member nodes) from being transmitted to unnecessary nodes.

```
Actions
% send request by monitor process
init.request.send:: function(hops) →
    if (j.status = monitor)  then
        % send request
        bcast⟨ttl,hops,msgID,MonitorID,SenderID⟩;
        msgID := msgID + 1;

% Receiving Data Request Message
recRequest:: recv⟨monitor, sender, msgID, ttl, hops⟩ →
    if (j ≠ monitorLIST) then
        stored-msg := False;
        if (monitor ∈ keys(records)) then
            (recHops, recForwardTo, recId) := records[monitor];
            if (hops < recHops) then
                records[monitor] := (hops, sender, recId);
            fi;
            (recHops, recForwardTo, recId) := records[monitor];
            if (id > recId) then
                records[monitor] := (recHops, recForwardTo, id);
                stored-msg := True;
            fi;
        else
            records[monitor] := (hops, sender, id);
            stored-msg := True;
        fi;
        if (stored − msg)
            (recHops, recForwardTo, recId) := records[monitor];
            unicast.send⟨monitor, j, 0, NodeData()⟩ to recForwardTo;
        fi;
    fi;

% Received Data Forward Message
    forwardData:: unicast.recv⟨target, sender, hops, data⟩ →
        (recHops, recForwardTo, recId) := records[target];
        if (j = target) then
            hopreq.deliver(sender, hops, data);
        else
            unicast.send⟨target, sender, hops+1, data⟩ to recForwardTo;
        fi;
```

Figure 6.5: Periodic-request algorithm by the monitor to evaluate the predicates by number of hops .

6.2.2 Flooding By Distance

The algorithm described above has been built such that nodes can request information in their request-hop neighborhood once it is needed. This algorithm is used when a monitor requests to evaluate predicates from its neighborhood or any nodes periodically, but if the surrounding nodes have changed their predicates values, the Hop-Request algorithm is not as valid when nodes need to send their data as soon as the values have updated. As such, we have implemented a network library to send data as soon as the data is updated and call it Hope-Flood, where the data is flooded to the appropriate destination.

The Hop-Flood algorithm holds a queue of messages it has to deliver. Each P_{send} cycle broadcasts what is at the top of the queue and raises the amount of times the message has been transmitted. When a message exceeds the full number of retransmits, it is excluded from the list. Fresh messages are assigned to the end of the queue such that they are transmitted after messages have been added to the queue previously.

Once a node receives a flooded packet, it examines to see the time-to-live (TTL) of the packet. If the value of (TTL) is 0, the message will be added to the queue and will continue to be forwarded. Also, the node will examine the ID used in the message. The protocol will deliver the message from a node to the monitor which has not yet registered its ID or received its message from different routes. In the algorithm, though, the member nodes normally start the checking algorithm to check their data and then transmit the information to the desired destination with flooding brodcast. The monitor does not need to flood its data and it is the monitor that saves unnecessary messages where the monitor will check its own data until it has received at least one data from other nodes. Figure 6.7 shows the overall algorithm when a node floods data in a grid topology.

Figure 6.6 shows the stages of the algorithm used to evaluate the predicates in this scenario. In contrast to the previous algorithm, in which the monitor initiates the request ($init.send$ :: **send.flood**(*hop,packetID*)), the members ($j.status = member$) will start disseminating the data. The map representing all messages received from Flood-by-Distance neighbors nodes by a tuple. A tuple of $(packetID, hops)$ is a map of address to $[(int, int)$ init $\emptyset]$ which is stored in the entry. If the hop variable does not reach the maximum distance ($hop \neq 0$), the node will check it and add the new packet to the queue **AddQueue()**. The queue of packets to be broadcast is a tuple containing: $(hops, sender, timeslot.msgId, ttl, resend)$, with the (Queue) being a queue of

$(int, address, int, int, int, struct)$ init $\emptyset$. After adding this to the queue, the *msgId* variable is incremented by one, resulting in a unique ID for each message. The node will then set a *timer* for broadcasting the packet and updating some of the variables. The node will update the queue packet if the top queue is empty ($topQueue \neq \perp$). Then it will check once more before broadcasting the packet. Some queue variables, such as the number of *hops* and the *ttl* value, will be updated for each broadcast. This is for the purpose of keeping track of the distance traveled for each broadcast. With the *resend* variable in this operation, the maximum number of times *maxresend* to re-transmit is updated. The node's final step is to check the queue and update it with the variables that were updated previously, such as *hops*, *ttl*, and *resend*. The packet queue will be received by another node when it receives the above broadcast. The node will check whether the $seenBefore$ variable is true or false. It will initially look in the node ID list to see where the sender ID is listed ($sender \notin$ **keys**(*list-senders*)). If this isn't the case, the $seenBefore$ will be updated. The list of message IDs in the list will then be checked and updated ($seenId < msgId$). The final step is to make sure the $seenBefore$ is up to date by double-checking both ($seenId = msgId \wedge seenHops > hops$). The **flood.deliver**() function will then be called to save the current packet queue.The final stage is to update the packet in the senders list and then check if the *ttl* meets the requirements for updating the queue.

```
% send message with hop
if (j.status = member)    then
init.send:: send.flood(hop,packetID)
    if (hop ≠ 0)then
        AddQueue(hops, sender, timeslot.msgId, ttl, resend)
        msgId := msgId + 1;


check::timeout (timer)→
    topQueue := check(Queue);
    if (topQueue ≠ ⊥) then
        (msgId, sender, ttl, hops, resend, timeslot) := topQueue;
        if (topQueue ≠ ⊥) then
            bcast⟨sender, hops + 1, ttl − 1, msgId, timeslot⟩;
            resend := resend + 1;
        check(Queue);
        if (resend < maxresend ∧ ttl > 0) then
            re-addQueeue(Queue, (msgId, sender, ttl, hops,
                resend, timeslot));
    set(timer);

% Receive Message in node
recvData:: recv⟨sender, hops, ttl, msgId, timeslot⟩ →
    seenBefore := True;
    if (sender ∉ keys(list-senders)) then
        list-senders[sender] := (packetID, hops);
        seenBefore := False;
    else
        (seenMsId, seenHops) := list-senders[sender];
        if (seenId < msgId) then
            list-senders[sender] := (msgID, hops);
            seenBefore := False;
        else if (seenMsId = msgId ∧ seenHops > hops) then
            flood.deliver(sender, hops, seenHops, timeslot);
            list-senders[sender] := (seenMsgId, hops);
    if (¬seenBefore) then
        flood.deliver(sender, hops, 0, timeslot);
            list-senders[sender] := (seenMsgId, hops);
        if (ttl > 0) then
            append(Queue, (msgId, sender, ttl, hops, 0, timeslot));
```

Figure 6.6: Flood by Distance algorithm by the member to evaluate the predicates by the selected monitors .

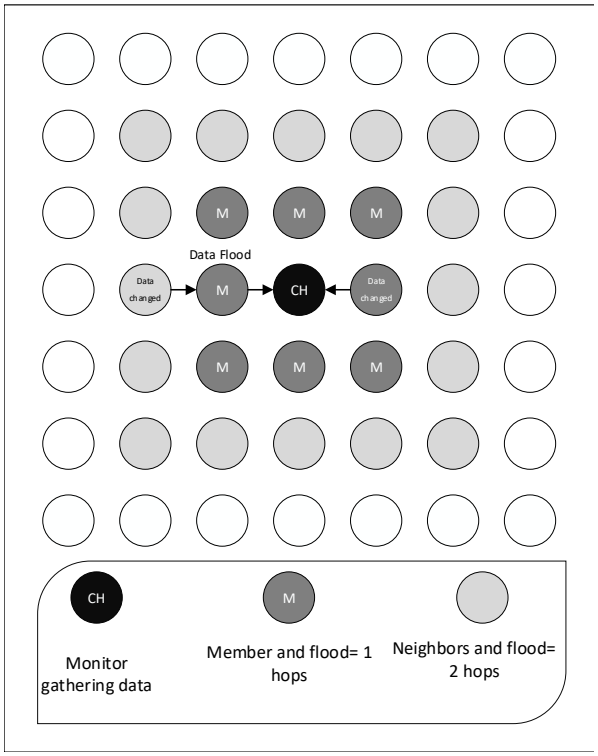


Figure 6.7: Overview of Flood algorithm when N=2

6.2.3 Event Checking Protocol

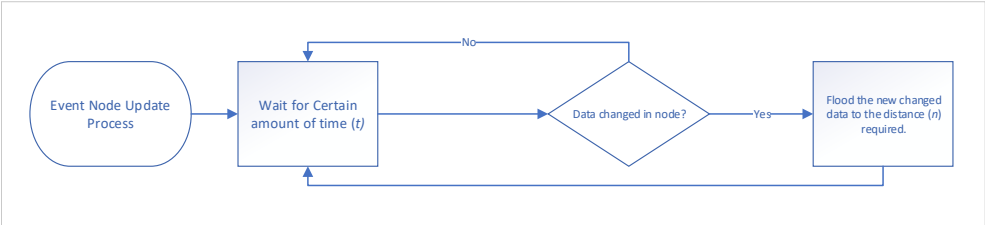


Figure 6.8: Event Node Update Process

The primary purpose for developing this network library is to check if there is a change in the values of the predicates for each node in the network. This process allows the manager of predicates to locally evaluate the predicates and disseminate the data locally once the predicates change. First, the network library used to check the values of predicates for each node has changed and then disseminates the data in the number of hops needed by the Hop-Manager to use other network libraries such as the Hop-Flood algorithm.

First, this network library creates the current state of the mote periodically and then compares it against the previously sent state. If no previously sent state exists or the state has changed as defined by the Differs-function, therefore the specified distance is flooded with the recently updated state. As such,

the maximum distance from which the data is required is disseminated by this network. As mentioned above, the algorithm checks that the data has changed periodically in nodes where this time variable *Period* is an important factor that has a major impact on overall efficiency. In the results and analysis sections, further discussions on this topic are discussed. Figure 6.8 illustrates the processes of the Event Checking algorithm. Figure 6.9 shows how a periodic timer **timeout**(*period*) is used to time the checking process, which is a variable that can be changed before the system is run. The if statement then checks for only members nodes that have broadcast **bcast**$\langle$*SlotSent*$\rangle$ their data messages before storing them in memory (*PreviousSlot* := *SentSlot*). The most important aspect here is to compare what has been stored in memory with the current data (*PreviousSlot* $\neq$ *CurrentSlot*). The algorithm then calls the flood algorithm from the dissemination manager to propagate the data when the values compared are different.

```
% Node checking for a change process
init:: Check timeout(period) →
    if (j.status = member)  then
          % Send the current data once the node go online
        bcast⟨SlotSent⟩;
         % Store the data has sent into the memory
        PreviousSlot := SentSlot;
         % Check the data has changed from previuos broadcast
            if (PreviousSlot ≠ CurrentSlot) then
                % broadcast the current data with hop distance
                FloodFunction.bcast⟨j, CurrentSlot, hop⟩;
            fi;
    fi;
% Set a timer to check how often to check the data
set(period, timer);
```

Figure 6.9: Heuristic algorithm to check data by each member in the network.

6.2.4 Polite Gossiping Dissemination

Trickle is a dissemination scheduling method originally designed to reprogram algorithms to disseminate code updates effectively through a wireless sensor network. It has, however, been found to be a versatile approach that can be used for a wide variety of applications, including service discovery, traffic scheduling control and multicast propagation [35]. To control its transmissions, Trickle utilizes two primitives and basic operations. First, a node that suppresses scheduled transmission in Trickle can hear enough of its neighboring nodes

transmitting the same piece of transmission. Second, if conflicting data is received (e.g. its parent changes its rank), a node can raise the frequency of data transmission. To overcome the resultant inconsistency quickly and minimize the rate of data transmission exponentially each time it hears a consistent data. A node running Trickle schedules a message for each interval to be delivered at randomly chosen times. The scheduled message transmission is governed by the parameters, variables, and steps of Trickle. Trickle uses three maintaining-state variables, three configuration parameters and six steps to function as defined in [95].

6.2.5 Neighbourhood discovery and knowledge

To identify node neighbors in the network, the predicate's local influence should identify which nodes belong to. It is difficult to attempt to evaluate the network state without understanding that nodes are neighbors of other nodes and is perhaps the most expensive energy segment. It means that before evaluating data, there must be a way to determine who is the neighbor for each node so that the data can travel back to the cluster head or parent node. Luckily, part of the implementation has already been completed because Contiki has a library to execute neighborhood detection [2]. Nonetheless, Some work have added some improvements that have already existed from previous work so that the library can modify its round-based data list to restore failures of nodes. However,this method solved when the clustering configuration is complete, the nodes will decide where their function as a member or a monitor in the network. Only in the case of 1-hop propagation would the nodes transmit their data according to their monitor. This would stop the network from overlapping communications between members' nodes to the nodes of another monitor. This also saves time for each node to be discovered in order to evaluate the predicates.

[2] neighbour-discovery.h `http://contiki.sourceforge.net/docs/2.6/a00344.html`

Chapter 7

Evaluations and Results

This chapter first discusses the metrics that will be applied in the evaluation, and then it discusses the results and evaluation.The results and evaluation of the proposed algorithms with various parameters and settings are also presented in this chapter. This chapter includes both simulations and testbed experiments.

7.1 Metrics and Focus of Evaluation

7.1.1 Overhead communication

This study aims to determine which algorithm or method is the lowest cost in terms of the number of messages that changes during the evaluation of the predicates. As wireless sensor networks mainly depend on energy sources, e.g., batteries, it will be practical to determine which routing protocol or debugging system monitoring applications or protocols is more efficient in terms of message quantity. Each time a node transmits or broadcasts a message, it consumes energy. The overhead of the monitoring system is calculated using analysis scripts in the simulation and testbed experiments.

7.1.2 Correctness and missed violations

Correctly evaluated predicate is defined before as predicate P is correctly evaluated at time τ if the results of evaluation that predicate with global knowledge at time τ is the same as the result of predicate P. In other words, when the monitor prints an evaluated predicate stating success status that there is no violation in the current data, this information is collected, which allows the analysis script to go back and check for these nodes at that time and compare whether the data matched the printed data in the predicate result. This is not

a latency metric to determine how fast the monitor system may detect more than how accurately the reports are printed by the monitors. The monitor may be able to print incorrect reports, indicating that there is a violation in a subset of nodes, but these nodes have updated their data and managed to adjust their timeslots, preventing any violation by the application algorithm. This is not the number of reports obtained from the monitoring system, whereas in flat architectures, the number of reports may be duplicated; however, it indicates the number of missed violations reported and the percentage of missed violations based on the records.

7.1.3 Latency

The general purpose of the assessment is to decide how the messages are instantly detected and evaluated. Thus, the principle of global latency is the time difference between the time at which the violation occurred in the node and the time at which the predicate was evaluated in the monitor. When event-based algorithms are implemented, local latency should be considered. This is often done to ensure that the factors of another dissemination algorithm, such as periodic dissemination, are matched. The principle of local latency is the time difference between the time at which the violation occurred in the node and the time at which the violation was detected by the node. This will provide information on the change detected using data and when the node sends its own data to the monitor.

7.2 Simulation Analysis and Evaluation

Communication Overhead

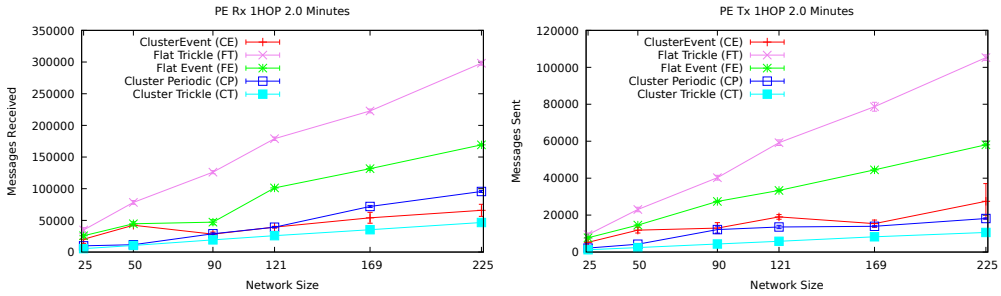


Figure 7.1: The results shown above are the monitors evaluating predicates using a 1-hop distance every 2.0 minutes.

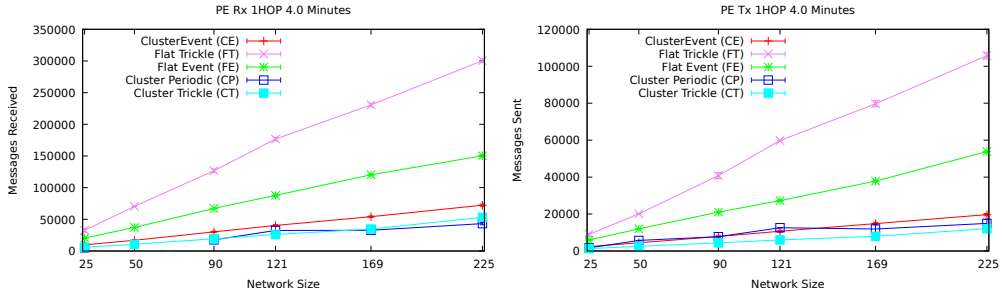


Figure 7.2: The results shown above are the monitors evaluating predicates using a 1-hop distance every 4.0 minutes.

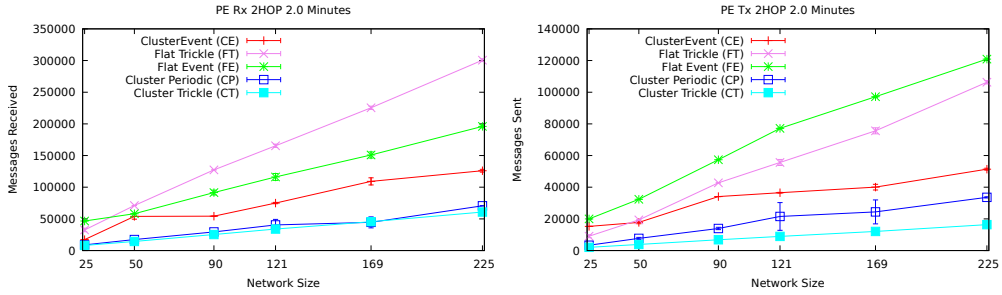


Figure 7.3: The results shown above are the monitors evaluating predicates using a 2-hop distance every 2.0 minutes.

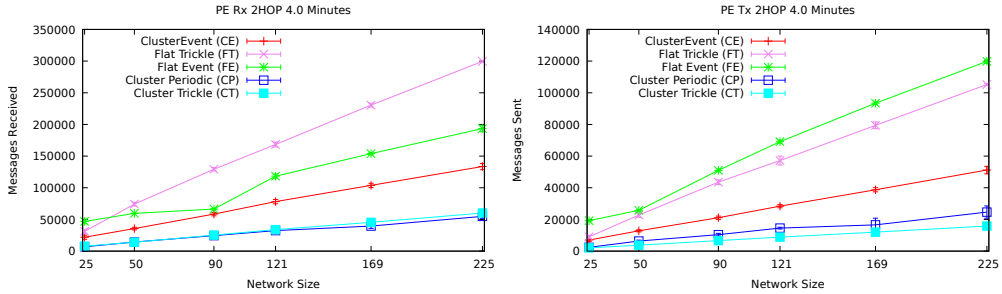


Figure 7.4: The results shown above are the monitors evaluating predicates using a 2-hop distance every 4.0 minutes.

Detection Latency

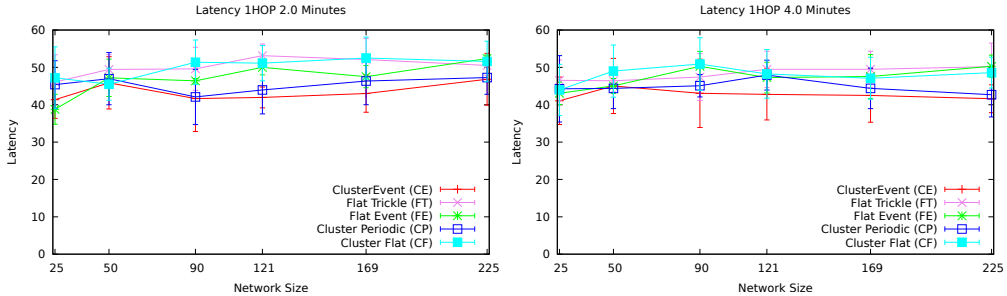


Figure 7.5: The results shown above are the latency in seconds where monitors evaluating predicates using a 1-hop distance every 2.0 minutes and 4.0 minutes.

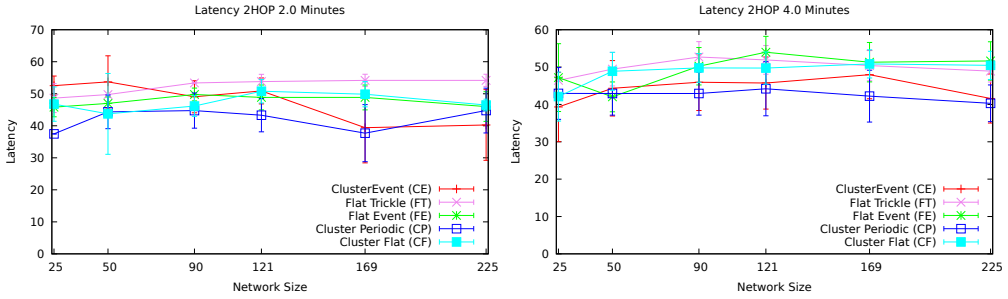
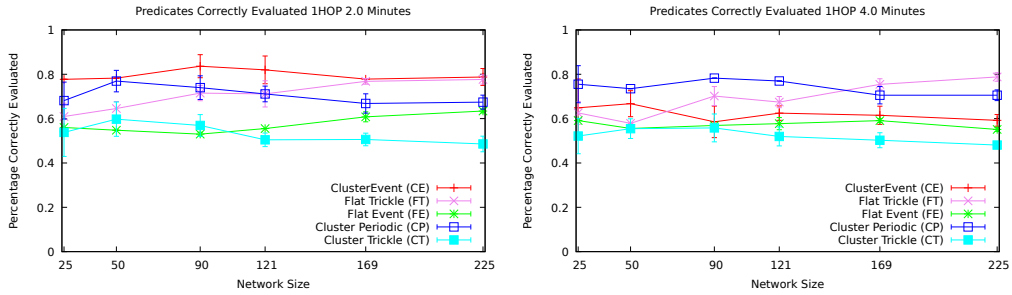


Figure 7.6: The results shown above are the latency in seconds where monitors evaluating predicates using a 2-hop distance every 2.0 minutes and 4.0 minutes.

Correctness



(a) Correctly Evaluated in 2 minutes.

(b) Correctly Evaluated in 4 minutes.

Figure 7.7: percentage of predicates correctly evaluated where monitors evaluating predicates using a 1-hop distance every 2.0 minutes and 4.0 minutes.

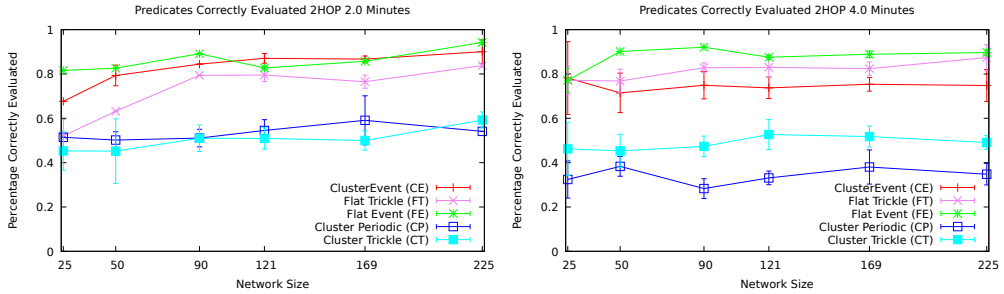


Figure 7.8: The results shown above are percentage of predicates correctly evaluated where monitors evaluating predicates using a 2-hop distance every 2.0 minutes and 4.0 minutes.

Based on the data shown in Graphs 7.1 ,7.2,7.3 and 7.4, all flat algorithms(flat-event and flat-polite-gossiping) exhibit worst performance in terms of overhead rates when the network size increases. The important point is that flat-architectures will transmit data through the network where all nodes operate as monitors; therefore, monitors (all nodes) will receive/send messages to obtain data required to evaluate predicates. Therefore, there is no routing protocol

or networking structure to reduce the total cost of communications. This also supports the fact that when the number of messages received is reflected in the correctly evaluated predicates where flat-architectures exhibit excellent performance when the data that need to be traveled increase, as shown in Graph 7.8. Moreover, this is recognized as the number of messages received and reflected in the correctly evaluated predicates, where flat-architectures exhibit the best performance when data that need to be traveled increase, thereby increasing the network coverage (Graph 7.8).

Another point is that the percentage of correctly evaluated predicates for 2-hop evaluation is lower than when evaluating 1-hop predicates in cluster-polite-gossip-based and cluster-periodic-based algorithms (Graphs 7.8and 7.7). This is undoubtedly made worse by the fact that more data are generated because by the time it arrives to the evaluating monitor, there is a greater chance that more of it will be out-of-date. There is also a higher probability that not all data will reach the monitor. In this implementation, where there is an increase in the distance of the required data, the cluster-periodic-based algorithm is unlikely to be improved. On the other hand, cluster-polite-gossip is likely to be enhanced by the parameters that affect the overall performance, such as timers that affect the broadcast and the time to suppress the broadcast in each node when duplicate messages are heard. The following sections or future studies will explore different parameters and consider an in-depth analysis focused on the dissemination protocol of polite-gossiping.

It is assumed that if the number of hop parameters (in flat-architectures) is increased to cover the overall network, it would have more overhead rates in evaluated predicates and performance rates (Graphs 7.7 and 7.8). However, this is not the case in latency in Graphs 7.5 and 7.6, where flat-architectures are slightly slower than the clustering algorithms, and this is the case for the clustering structure and conflicts with messages that may occur owing to the increase in the number of messages generated by flat-architectures. In other words, the clustering structure avoids the delay that can slow the data from traveling to the targeted monitor with fewer messages, which can overlap in flat-architectures or messages lost. As mentioned, latency measurement is calculated based on the average of the overall latency from correctly evaluated predicates that are detected. This is the main reason why a small number of messages are guaranteed to reach the monitor in cluster-periodic, and therefore, cluster-periodic has less latency than other algorithms, particularly the algorithms of flat-architectures.

The flat-event-based algorithm varies in terms of performance (graphs 7.7 and 7.8), where the accuracy percentage is more in 2-hop than 1-hop results. As

described before, flat-architectures can detect violations and gather information from all nodes instead of selected monitors, e.g., cluster algorithms. In addition, to evaluate predicates, the algorithm comprises redundant data of various nodes. The key distinction in these graphs is that the radius of influence of the predicates or, in other words, the number of nodes that need to be evaluated in each monitor is more within the distance of 2 hops than that within 1 hop, providing the flat-event potential to capture these changes correctly. If the flat-event-based algorithm needs to be improved, the dissemination manager would have to increase the number of hops to propagate data, thereby increasing the energy cost of the system where more data will be generated for transmission across the network.

It may be argued that expanding the predicate time would improve the rate of success of evaluation, as shown in Graph 7.7, as a small number of messages would conflict in the network. Increasing the time will, of course, based on the latter logic minimizes the number of messages sent owing to periodic implementations (cluster-periodic), thereby extending the battery life. This also applies to Graphs 7.1 and 7.2 where cluster-periodic has slightly increased the use of messages when the evaluation duration decreased. There were equal output ranges relative to the two sets of data, which can be noticed, regardless of the duration of the predicate period except the above notice. As shown in Graph 7.7, (flat-event), (flat-polite-gossiping), and (Cluster-Polite-Gossiping) remain in the same levels when the time period increased. With regard to the correctness of predicate evaluation, there is an important distinction. The local cluster-event-based evaluator exhibited excellent performance while evaluating the 1-hop predicate. Nevertheless, for the 2-hop performed best by the local periodic evaluator, this would expect that this would usually have been caused by the rise in message transmissions leading to a busier network of more collisions and lost messages. The number of messages sent increases as the network size increases, but the ratio of correctly evaluated predicates remains the same; thus, it will initially appear that collisions do not occur. On the other hand, as these predicates are evaluated throughout the network, they are local by nature; therefore, an increase in messages, sent across the whole network, does not paint an accurate representation. What is assumed to be the case is that the 2-hop predicate has more local traffic, and therefore, more local collisions are compared with the 1-hop predicate, which leads to cluster-event-based in-network evaluators performing poorly when a large amount of redundant data are transmitted. With cluster-periodic algorithm, data are transmitted as requested. Thus, although the sending of cluster event data may have improved 1-hop predicates with regard to the higher number of transmissions, due to the additional number of hops data requires to travel to

evaluate 2-hop predicate sending data as changes eventually lead to more local collisions result in fewer current data reaching nodes, allowing event-based evaluators to do worse for 2-hop predicates than 1-hop predicates. Another observation is that the number of predicates correctly evaluated for 2 hops is less than that evaluated for 1 hop. This is possibly caused as more data are required and there is a greater risk for more of it to be out-of-date by the time it arrives the monitor node. Thus, the cluster-event-based algorithm preforms better with less distance to cover and less time to evaluate predicates.

This is observed in Graphs 7.5 and 7.6 where latencies of cluster-event and cluster-periodic are performing better than other evaluating algorithms mentioned above. The cluster-periodic algorithm remains between 40 and 50 seconds in most scenarios. This demonstrate that low messages, which considerably reduce latency, and the speed of requesting data well impact the algorithm in terms of delivering data faster; however, this is not the case in the correctness in Graphs 7.7 and 7.8 where the cluster-periodic as mentioned remain one of the lowest, which is influenced by the time when the monitor requests data and gather information. If this problem is to be improved, the time for the monitor to request data and receive new updates and changes from the member nodes will be reduced. However, this would increase the number of messages and latency, so that the messages may overlap and may not reach the monitor because they get lost while transmission. In general, the cluster-periodic algorithm might be appropriate for applications that need to check their predicates for a longer time and less changes in the targeted predicates. In this case, in terms of latency and overhead cost, the cluster-periodic algorithm would be the best option. Another factor that can impact this algorithm is that where the targeting data need to be evaluated is far from the monitor, results will be less accurate to evaluate these predicates where the data will be old, and the time request for the monitor needs to be decreased more to solve the problem. This shows that in graphs 7.7 and 7.8 where the hop is increased, less accurate results are obtained to evaluate predicates.

This study was intended to investigate various algorithms and different dissemination protocols with the highest performance to match the infrastructure developed (selecting monitors). With each algorithm and various parameters, the overall performance that can balance cost and performance is always affected by the implementation of more algorithms. Moreover, the goal at the beginning was to explore an infrastructure such as cluster-event with regard to 1-hop parameter for high performance at low cost (Graphs 7.7 and 7.1. The 1-hop and 2-min cluster-event algorithm is one of the best algorithms that provide a minimum number of messages, greater accuracy, and latency

efficiency while evaluating TDMA time-slot predicates shown in Graph 7.5. This is because there will be less member nodes in a grid topology where the monitor will be a 1-hop away from the monitor. To check their time-slot values if they have changed, these member nodes will use the checking-node algorithm and then fire the flooding-broadcast if the data have changed immediately. This is supported as the evaluation period decreases, allowing the monitor to obtain the latest data from member nodes, and as the changes occur, the member nodes will keep flooding the data. While the TDMA protocol changes compare relatively highly with other protocols, the cluster event algorithm can indeed detect violations faster with low cost.

One of the main remarkable observation when comparing the overhead rates between 1-hop and 2-hop in Graph 7.1 and Graph 7.3 . It noticed that the only Flat-Event have increased massively and become the highest in terms of messages sent. This is expressed in the nature of WSN where the messages to be sent on the network are often increased by common protocols including standard Flooding-broadcast when the distance to be flooded is increased. This also supports the purpose of this project, in which Flat-Architectures often expected to be the highest in terms of energy consumption, where monitors are evaluated by all nodes.Since the Flat-Polite-Gossiping algorithm is designed in such a way to minimize the total messages, both algorithms remain one of the highest predicate evaluation algorithms. However, Flat-Event evaluation increased in 2-hop predicates because the fact that the neighborhood of data to be disseminated has increased from 1-hop to 2-hops mainly. And the opposite to the techniques of Polite-gossiping where nodes do not send data until they have received any change from neighbor nodes or suppressed by a timer. The Flat-Event uses the simple event mechanism to flood immediate change after checking its own data for each node. As a concept, this mechanism is often useful to implement to have more performance rates of evaluating predicates in increased distance such as in Graph 7.8.

7.3 Transients: Sudden Time-slot Changes for TDMA Protocol

This section examines the system's behavior using synthetic attribute changes on the protocol, evaluating the accuracy of the update dissemination, latency efficiency, and the probability of missing violations separately. The actual data or predicates where the monitor system is to capture and then evaluate them are referred to as "changes" in this section. The application algorithm or a hardware event, such as temperature sensors, determines how often these values or data change in the nodes. On the TDMA protocol, a series of

time-slot modifications were fabricated. This allows the system to measure its performance in the worst-case scenario, where the number of changes in the debugging system is extremely high compared to default TDMA protocol and the ventilation system. To put it another way, this thesis investigates the value of low-changing applications and high-changing protocols, with provided algorithms evaluating the impact when the predicates (changes) shift from low to high. In the previous sections, two applications were evaluated, the first one was the Ventilation System, which has low changes, and the second application would be the default TDMA protocol, which has higher changes. This section will evaluate the predicates in which the predicates (changes) will change at a rapid rate. As a result, when compared to previous tow applications, this will be the one with the highest changes. As a result, and before introducing such a scenario, the results by logic would be worse than the previous results presented by the default TDMA protocol. The aim of this implementation is that some algorithms and dissemination protocols do not perform well with very high changes, whereas others may perform better than detecting and tracking smaller changes. Furthermore, the debugging system might be able to monitor other applications or network protocols for extreme changes and violations. It's worth noting that the original TDMA protocol is still one of the most common protocols and applications for changing data distribution.

```
starup:: init →
∀n ∈ Nodes
    generate(rndvar);
    set⟨Node[rndvar]⟩ := probability(0 to 1);
    if⟨Node[rndvar] <= 0.5 ∧ Node[rndvar] >= 1⟩then
        TDMA-Distribution-Function(solt);
    else⟨Node[rndvar] <= 0 ∧ Node[rndvar] >= 0.5⟩then
        Node(solt) := current[solt];
    fi;
```

Figure 7.9: Pseudocode of the synthetic TDMA Time-Slot distribution across the network

It was difficult at first to introduce a similar algorithm to TDMA distribution with the aim of increasing the similarity of nodes with the same slot value. For example, manually or randomly assigning slot values across the network could be simple task. However, the system will generate duplicate reports, and when the results are gathered, the correctness percentage will be inaccurate. To put it another way, the system would continue to allocate and assign slot values in order to determine when the change occurred and when it was detected. As a result, latency estimates would not be able to be measured using simple

methods where changes must be implemented properly. The synthetic TDMA allocation algorithm was implemented to prevent these drawbacks and maintain the analysis focused on monitoring the predicates.

Generally, the algorithm does not differ from the original algorithm used in the previous results, with the goal of increasing the changes by at least 50%. The Pseudocode of the synthetic TDMA Time-Slot distribution across the network is shown in Figure 7.9. First, the algorithm uses the original TDMA protocol to distribute and allocate slot values across the network. The difference here is that the slot values are allocated with fixed and predefined numbers $generate(rndvar)$ to decide which nodes have the slot value within a range. The variable value range is set between 0 and 1 by a probability function $probability(0\ to\ 1)$. However, when the time-slots are distributed, the TDMA function (**TDMA-Distribution-Function**($solt$);) will be called to distribute and assign the slot values from the original algorithm with the following conditions. When the node value is in the range of 0.5 to 1, this function will be called. That is, the algorithm will distribute and assign time slots in the same way that the original TDMA protocol does. Otherwise, when the node's time-slot value is between 0 and 0.5 ($Node[rndvar] <= 0.5 \wedge Node[rndvar] >= 1$), the algorithm will not readjust it, as all TDMA protocols do. In order to increase the number of violations in the network, the time-slot value in these nodes will not change ($Node(solt) := current[solt]$). This means that at least half of the nodes will hold the same time slot, which is significantly increasing the number of violations. Every TDMA protocol will always attempt to eliminate the values-like (time-Slot) within two hops.

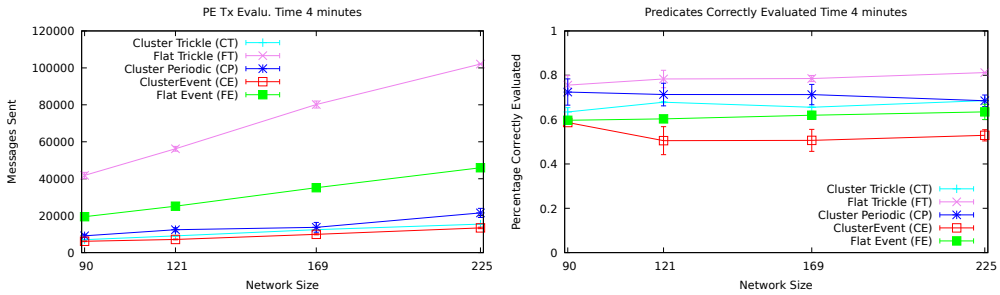


Figure 7.10: The results shown above are the monitors evaluating predicates using a 1-hop distance every 4.0 minutes and applying the new change in the TDMA protocol

When these changes are simulated and introduced in 7.10, it's worth noting that all Flat algorithms remain extremely costly in terms of communication overhead. When the differences are compared, all clustering algorithms are extremely low, and there is a large gap. Flat algorithms have the advantage of

broadcasting messages to all nodes, and all nodes in the network can monitor these messages, as previously mentioned. This will come at a high cost and with a high level of efficiency, especially if the changes in these experiments are extremely high. The correctness percentage in the other graph, where the percentages are reflected from the communication cost, supports this statement. It is worth noting that Cluster-periodic has maintained one of the highest percentages of correctly evaluated messages due to its guarantee of message delivery on request within a 1-hop distance. However, in other parameters and scenarios, such as a 2-hop distance parameter, the periodic algorithm will maintain lower rates.

The most important observation from this graph is that Cluster-Event appears to have the lowest percentage of correct predicates. Changes would be extremely high, as mentioned in this implementation, where messages will flood constantly and monitors will not be able to evaluate the predicates in a timely manner. More messages would result in more messages being lost due to the nodes' frequent changes. Since the algorithm for checking changes in nodes floods the data, the networking library's similarity is reflected in the percenters of correctly evaluated predicates in both Flat-Event and Cluster-Event. The next section gives more information about the problem and how to solve it.

7.4 Correctness limitations and solutions

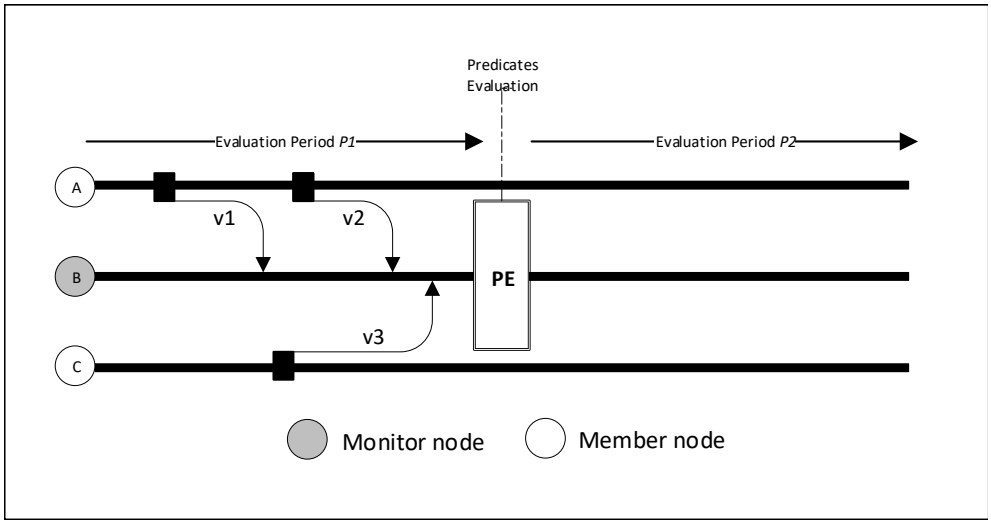


Figure 7.11: shows how missing violations can occur when monitors are introduced to extreme changes.

The problem that the monitoring system encounters as more data is sent from the surrounding nodes is shown in Graph 7.11. For instance, in the current implementation, a monitor would collect the required data for a defined

period of time in order to evaluate enough surrounding nodes. If the time parameter is increased, the monitor can only evaluate the most recent data obtained from the nodes, resulting in missed violations. If the time parameter is decreased, the monitors will be overloaded, and there will be enough nodes to evaluate, since each monitor must evaluate at least two nodes. In Graph 7.11 node A broadcasts its data as v1 to the monitor, then sends another update after a period of time, signaling that its data has changed to v2. This occurred during the first round of predicate evaluation, and the monitor did not start evaluating the data. The monitor will evaluate v1 and v3 from node C in this situation.

When the number of changes is extremely high, and monitors are having difficulty detecting these changes in a timely manner, improvements must be implemented to address this problem. Therefore, detecting changes quickly and gathering the information needed to evaluate the correct predicates is a difficult task. When this scenario is used, some solutions can help to improve the accuracy of the detecting predicates. Implementing a timer in each node, for example, will allow the monitor to collect the most recent data obtained. However, this approach is similar to the one already in existence for monitors, where monitors evaluate their own data after a period of time. As a result, some changes may go undetected, and old predicates will not be evaluated, as monitors will only evaluate at the most recent data collected. Another solution is to use a history buffer on the monitors, where the monitors can store their most recent received data in the buffer and evaluate it over time based on the received messages. However, this approach will consume up a lot of memory on top of the debugging system, and some motes specification won't be able to handle it.

Another option for resolving this problem is to implement a FiFo mechanism on the monitors. The FIFO (first-in, first-out) method of handling program work requests from queues or stacks is to handle the oldest request first. This allows monitors to evaluate received data as soon as it is received and queue new data until the previous set of data is evaluated. However, in order for this solution to work properly, it may cause some heavy workload processing on the monitors. Multiple tasks cannot be performed by the monitors at the same time, especially if an application or network protocol is already running on top of the monitoring system. To address this issue, we added history buffers to the monitors we selected for our strategy in order to improve detection of violations. In section 7.7, the results and implementations of this improvement are demonstrated.

7.5 Parameters and Settings

This section contains a more in-depth analysis and research. The method of obtaining results and changing parameters during simulations, for example, has changed since the previous graphs. Furthermore, when some of the experiments have changed, this would provide more conclusion and overall of the findings. The principles of local and global latency, for example, are introduced. In addition, the number of violations that occur in the system is calculated and measured. This would also provide more findings and insights from the experiments, such as the number of reported violations, which can be extracted from the data. In order to achieve more accurate latency results, the checking time where the predicate can be flooded or requested is also changed. In terms of the changing analysis scripts, the correctly evaluated percentage in previous graphs was only taking success predicate reports, while this section is taking both success and failure predicate reports (success reports mean there is no violation in the report when the monitor evaluates the predicates). More definitions and explanations for each evaluation metric are listed and explained in detail in Chapter 4.

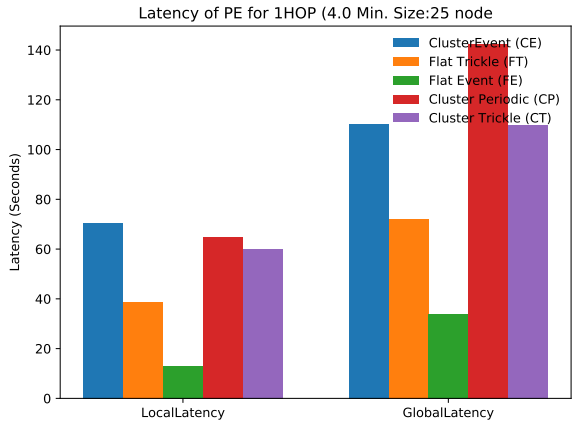


Figure 7.12: shows the latency for 1-hop and 4-minutes evaluating in 25 nodes

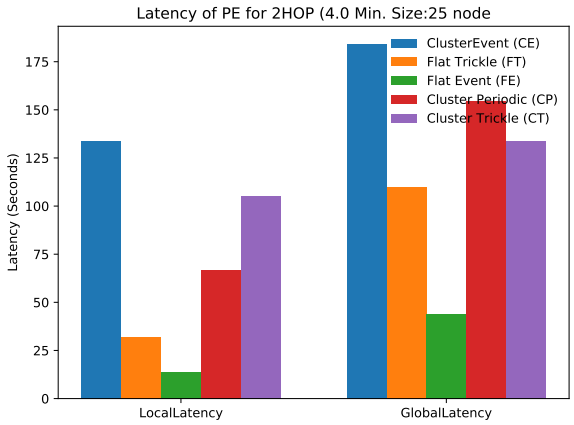


Figure 7.13: shows the latency for 2-hop and 4-minutes evaluating in 25 nodes

To begin, it's worth noting that these graphs measure local and global latency differently than previous graphs in order to provide more accuracy in the actual details of the traveling messages. The local latency is used here to provide further insight into the analysis and knowledge of the individual actions. It is applied by a single node before reach to the monitor which evaluates the predicates. In Cluster-Event and Flat-Event, for example, the time parameter for checking changes in each node has been changed. This parameter was changed in the Event Checking algorithm so that after 2 minutes, each node would check its own data. This parameter was changed in the Event Checking algorithm so that after 2 minutes, each node checks its own data and, if there is a change it will then floods the change to the monitor.

The above changes have a significant negative impact on the Cluster-Event in both metrics. Since the parameter time has changed and increased, it will have an effect on the overall latency. To solve this, it would return the parameter time as before, and the node would be forced to check its own data with a shorter time period. This has also been noted in CLuster-Event, where latency increases when data must be transmitted over multiple hops in Graph 7.12 and 7.13.i)In nodes, the checking protocol's parameter period is set to zero,ii)the hop parameter is set to a distance of one hop,iii)in a maximum of two minutes the monitor evaluates the predicates.

As shown in graphs 7.12 and 7.13, the Cluster-Periodic algorithm continues to be one of the slowest in terms of latency, as the algorithm must check the predicates from the monitors for longer periods of time. Both Flat algorithms have increased their latency by approximately 20% as compared to previous graphs. As previously stated, the checking time has increased, so the time it takes for data to reach the monitor has increased. It's worth noting that when

data needs to be suppressed or propagated in a timely manner, Cluster-Trickle needs to be improved by its own parameters.

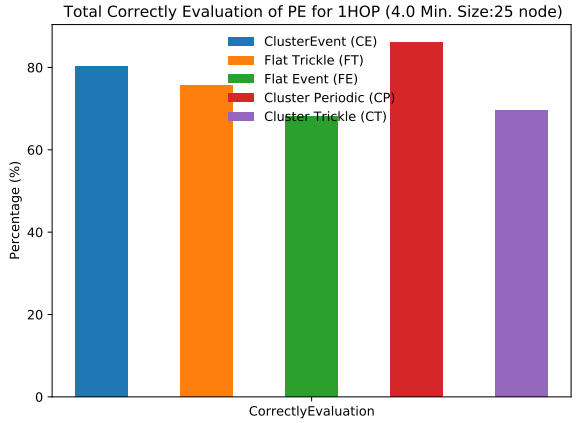


Figure 7.14: shows the correctness reports for 2-hop and 4-minutes evaluating in 25 nodes

When a new parameter is added to a scripting analysis files, the percentage of correct evaluating predicates is calculated differently than before. Graph 7.14 illustrates the percentage of correctly evaluated predicates based on both the report's success and failure states after the reports are completed. As previously noted, the previous assessments were taking reports that only had success states when the monitor finished evaluating predicates. The percentage of correctness on both clustering algorithms remained similar in this graph to the previous graphs presented in the previous chapter. This graph illustrates that even though new parameters or analysis methods were used, the results remained identical to the previous graphs.

7.6 Testbeds Results

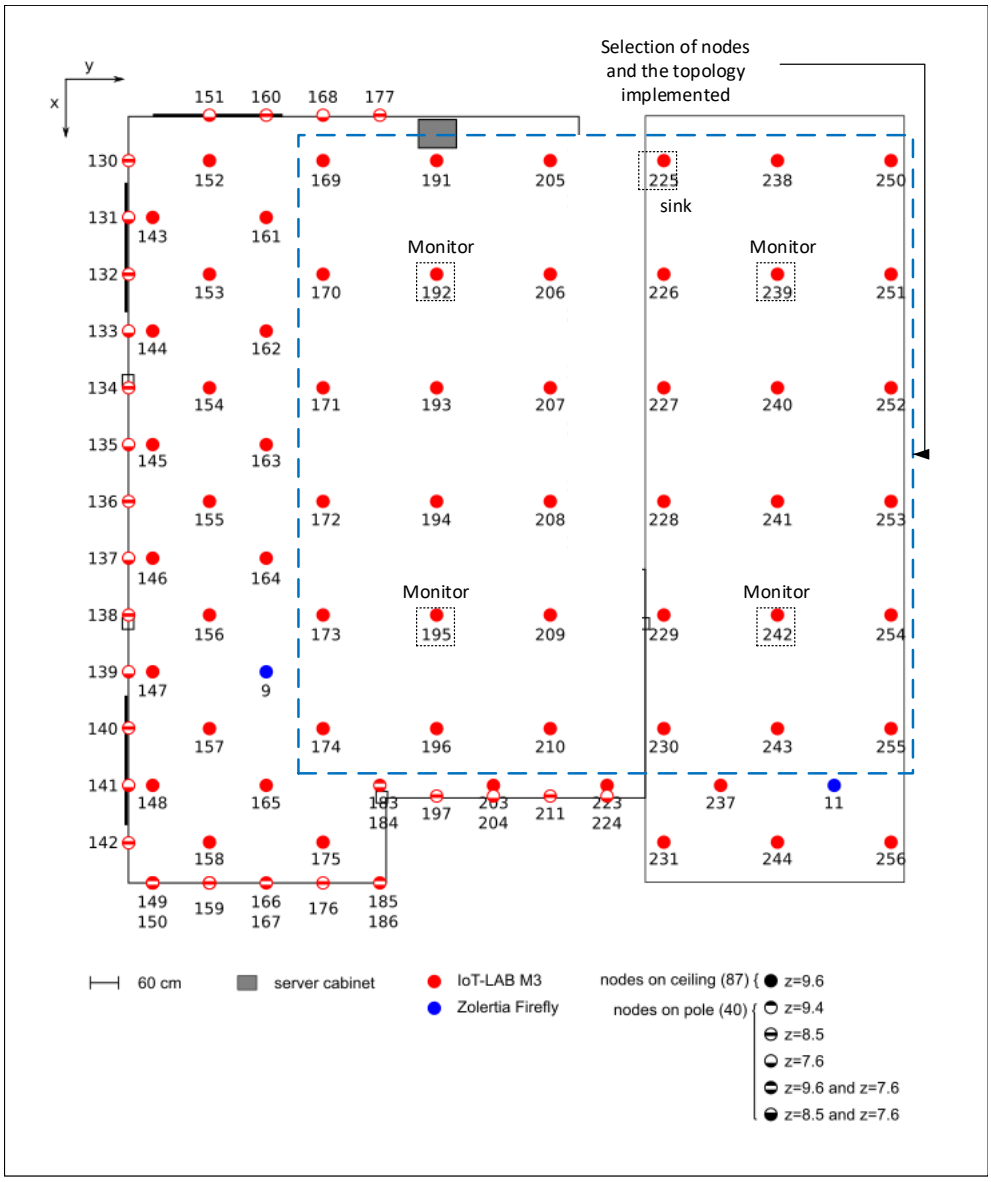


Figure 7.15: Testbed topology and selected nodes

At first, the experiments were conducted with a 6x6 grid topology similar to the approaches of the simulation experiments. As mentioned in section 3.4, the testbed and mote type selection is explained. The final topology and node selection are shown in Graph 7.15. The sink is implemented in the middle as node225, where the first monitor is expected to be selected is node239, as seen in graph7.15. The final output of the monitor selection by the clustering algorithm is shown in graph 7.15 when the cluster algorithm is first run into the hardware network. The nodes in the north, east, west, and south are separated by about 10 meters, which correspond to an RSSI value of -49 for the i-band range. While nodes in the northeast,northwest,southeast, and southwest are

12 meters away from the sink, monitor, or node that broadcasts its message in the network. The RSSI value for these nodes is set to -52 as o-band. The implementation will begin with a smaller node size of 3x3 since each node size will have an additional monitor, bringing the total number of monitors expected in 6x6 to four. Every run takes 4 hours to determine the average rates of all nodes transmitted and received messages. Cluster-Event (CE) algorithm and Flat-Event (FE) algorithm configuration; 2-minutes set evaluate the predicates on the monitors and 0 to the parameter check the changes on nodes.

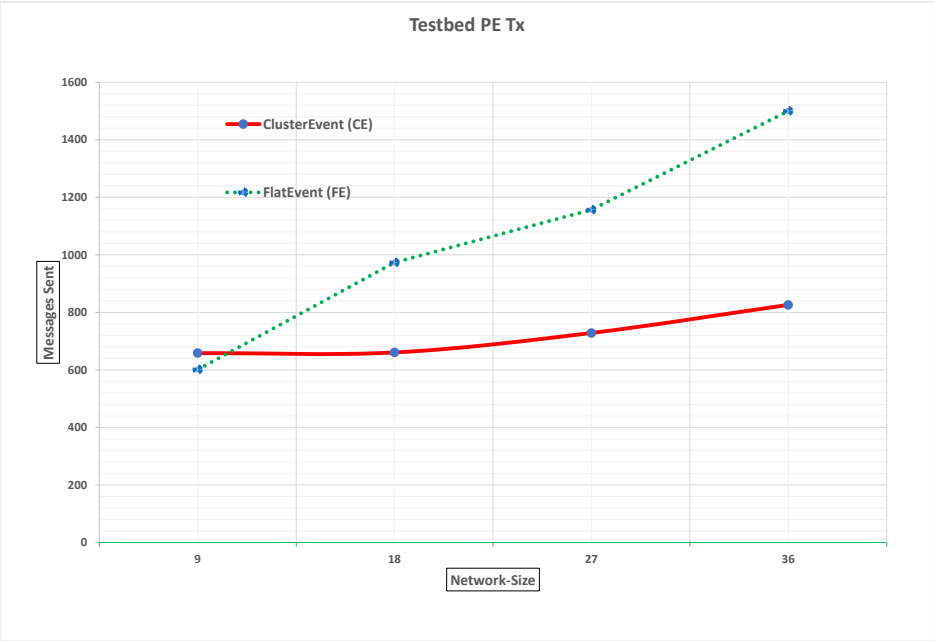


Figure 7.16: shows the overhead communication results in tesbed experiments

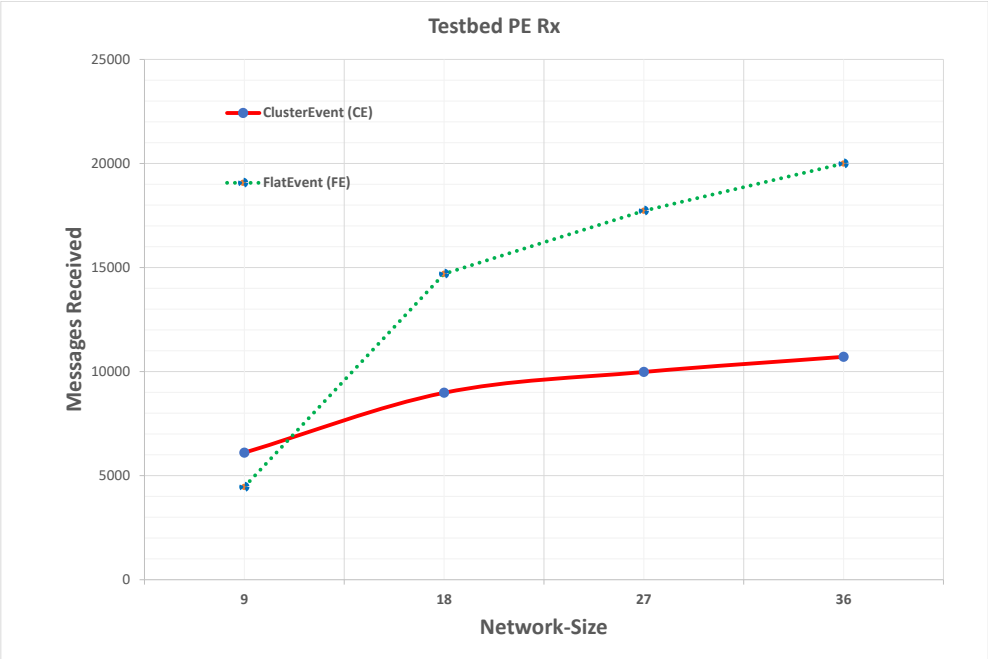


Figure 7.17: shows the rates of number of received messages results in tesbed experiments

It is worth noting in graphs 7.16 and 7.17 that both algorithms have an increase in the number of packets sent and received when the number of nodes increases. The important segment for this is that Flat-Architectures will always transmit data through the network where all nodes operate as monitors, so monitors (all nodes) will always receive/send messages in order to obtain the data needed for predicates evaluation. Flat-Architectures will always transmit data across the network where all nodes act as monitors, so monitors (all nodes) will always receive/send messages more in order to obtain the data required for the evaluation. As a result, there is no routing protocol or networking mechanism in place to lower the overall communication expense. This is also supported when the number of messages received is correctly expressed in the predicates evaluated, where both Flat-Architectures perform the best when the amount of data to be transported increases. As a result of this note, it is predicted that the number of received messages would follow a similar pattern. These graphs and outcomes, which have been implemented on testbeds, illustrate the similarity of the simulation experiments presented in the previous chapter.

7.7 Experiments Results (Ventilation System)

7.7.1 Update Dissemination and Simulation Experiments

Three complimentary methodologies are used to evaluate the performance and correctness of the monitoring system prototype. By focusing on global knowledge of the network state, we will examine and compare Cluster-Event, FLAT, and TREE. Furthermore, implementing a simulator makes assigning value distributions to network nodes easier and allows us to test alternative scenarios efficiently. Finally, it enables us to examine reasonably large networks (up to 225 nodes) at the cost of a less precise wireless transmission representation. To perform simulation experiments, we use our Contiki implementation of our monitoring system. The user can choose which distribution protocol to utilize based on system and application needs because our methods are configurable.

We use artificial distributions of attributes to investigate the behavior of the monitoring models, and we evaluate the performance of the update dissemination and the chance of correctly evaluated predicates separately. The nodes are organized in a square grid with a 10 m spacing between them. We analyze network configurations ranging from 25 nodes to 225 nodes. The details of the simulation settings are mostly applied similarly in section 3.3 . The default CTP parameters of the Contiki distribution are used in TREE[1].The TREE protocol is outlined in section 2 . This section will explain and examine why this comparison is important in order to determine which structure propagation algorithm offers the best performance results. This section compares Custer-Event, our best algorithm, to the state-of-the-art and current DICE [56] propagation protocols FLAT and TREE. We are primarily concerned with detection latency, correctness, and communication overhead. Section 3.3 explains these measurements and metrics in detail.

The complexity of the invariant determines the number of attribute values that must be contained in the nodes attribute values. We employ five invariants to capture this element, which are collectively defined as:

$\forall n1, ...n_k \sum_{i=1}^{k} x - n_i < T$ where $k \in$ [2,3,4,5]

In other words, some nodes in the network will randomly change their temperature values. The number of nodes that the monitor can evaluate is represented by k. The monitor is set to capture and check the values of the surrounding nodes for this reason. To avoid clogging up our graphs, we only present the data

[1]collect.h `https://anrg.usc.edu/contiki/index.php/Collect-view_Code_Details#Source_Codes`

for the extremes, i.e. for $k \in [2,6]$. The numbers sensed in the range [1; 100] are used to emulate a physical phenomenon (e.g., a heating source). This distribution is important in that violations do not follow a pattern and can occur anywhere in the network, making these random changes more artificial.

7.7.2 Overhead Results

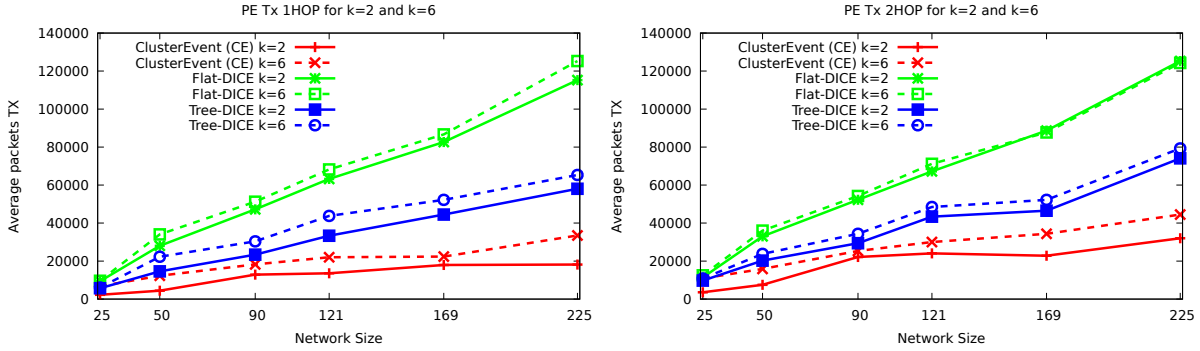


Figure 7.18: The results shown above are the monitors evaluating predicates using a 1-hop and 2hop distance and k=2,6.

The number of messages required to disseminate the predicates over the network is shown in graph 7.18 . As stated previously, these numbers of messages are required to evaluate the predicates, which must be minimal in order to conserve network energy. When the number of nodes in the network increases, and the number of nodes required by the monitor increases, it's obvious that the number of messages will increase in the graphs. The important thing to note from graph 7.18 is that Cluster-Event has always been the lowest of the three architectures as the number of hops has increased. The data clearly shows that overhead rates, or the amount of messages transmitted, are significantly high in all FLAT approaches. TREE remains in the middle of the two methods, with modest overhead rates. Even though the dissemination manager's setting is set to collect/flood data in 2-hops, the overall performance is similar to 1-hop. This is due to the natural architecture of the structure, which has fewer monitors than the other two, and nodes in the network will forward the necessary messages to these monitors. Messages propagate in an unstructured form in FLAT: one message may "quench" another at a specific node, but because messages propagate along arbitrary paths, the same message may reach other nodes via a different path and cause additional update changes. TREE's greater value is defined by the messages required to keep network caches and consistent, and so, in essence, by the structure generated by TREE. Cluster-Event structure update propagation, on the other hand, does not have this overhead. However, the number of messages in FLAT is greater than TREE because a message change from one child (representing the state of a complete

sub-tree rooted at it) may be superseded by a message from another child at a parent node, resulting in a single message change propagated upstream.

7.7.3 Latency Results

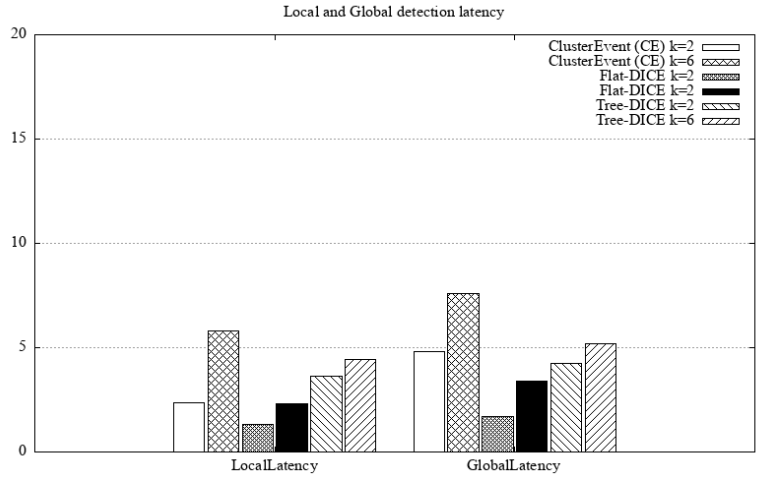


Figure 7.19: shows the latency in secounds for 1-hop in random violations and evaluating over 25 nodes.

The average local detection latency is shown in Figure 7.19 . The random of violation changes has a major impact on the relative performance of FLAT, CLUSTER, and TREE: the behavior of TREE and CLUSTER with this setting is considerably worse than FLAT—an order of magnitude in the case of k = 6.Section 3.3 describes the differences and explanations of the evaluation measures for local and global latency. The k value, in general, increases overall latency. The local latency for the CLUSTER with k=2 is small and comparable to the FLAT. When k is raised, however, this becomes worse. Indeed, maxima might appear everywhere in such a distribution, and a violation is difficult to detect by neighbors in such circumstances. TREE and CLUSTER have much greater latency because: i) the possibility of detection is higher in nodes closer to the root, and ii) the root can be far away from the nodes participating to the detection in these approaches. Instead, the propagation of maxima in FLAT can be considered of as a "bubble," whose expansion (generated by polite gossiping propagation) is not limited to a structure, as it is in TREE and CLUSTER. Therefore, when the "bubbles" corresponding to the attribute maxima overlap at some node, the violation is detected. Aside from the attribute value distribution, which is typically unknown in practice, the complexity of the monitored invariant, which in our studies is represented by the value of k, is another factor influencing the relative performance of the three methods. To accomplish detection, a node requires more data from more

sources as k rises. Thus, as seen in Graph 3.3, the latency increases as k grows for FLAT, CLUSTER, and TREE. In CLUSTER and TREE, however, this variable has a far greater impact.

7.7.4 Correctly evaluated reports

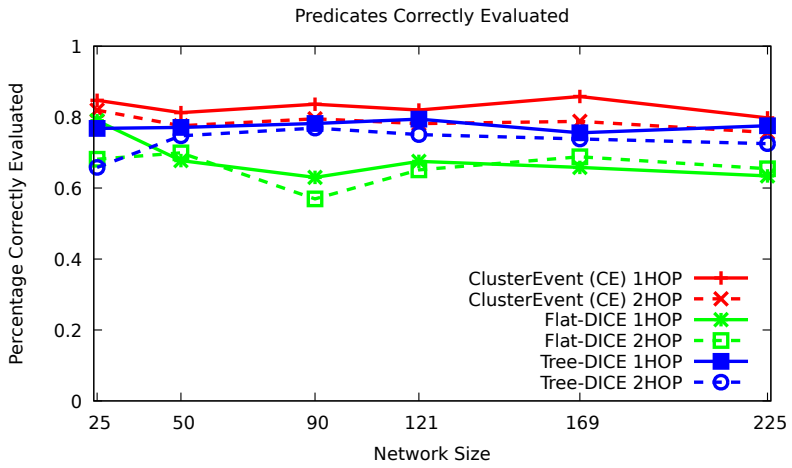


Figure 7.20: displays the percentage of 1-hop and 2-hop settings in random violations for different network sizes.

Similar reasoning can be used to explain patterns in detection count, or the percentage of nodes that detect a violation. As previously stated in sections 7.2 and 7.4. Because FLAT and TREE ensure eventual detection at all nodes, this statistic is only relevant for CLUSTER. As seen in Figure 7.20, the lower the detection count, the more complex an invariant(random distribution) is. As number of hops grows, the likelihood of a node performing detection decreases. The attribute value distribution follows a similar logic: the more dispersed the values triggering a violation, the fewer the nodes with all the essential data to enable detection, which explains the lower values for the implemented random distribution in this case. In general, the FLAT approach appears to be the worst of the three. The graph illustrates that the CLUSTER performs best under 1-hop parameter settings. In terms of the number of correct reports, the TREE approach appears to rank second after the CLUSTER approach. In Section7.4 we discussed the compromises we make w.r.t. the potential of missing a violation; here, we evaluate their usefulness. Therefore, we examine the history buffer's effectiveness in improving violation detection. We assess the extent to which history buffers help avoiding missed violations by monitoring the above temperatures predicates with different network sizes. Without history buffers, the percentages of capturing violations were below 70% when we first implemented our selection monitors protocol. Thus, the percentages of capturing violations after using this strategy are shown in

graph7.20. What's essential is that the strategy is only used in monitors to reduce code size and avoid adding more history buffers to all nodes. In our experiments, the maximum history size was 8. We report the percentage of violations detected by all monitors over 60 runs in order to obtain conclusions. CLUSTER manages to capture all the violations triggered by all updates, even having a limited history size of only 8 elements. The rationale comes in the shape of CLUSTER's topology for our selection monitors: the average number of members is 1, while the maximum is 5. Therefore, each monitor receives data from a small number of members, reducing the chance of the history buffer being filled. This isn't true in FLAT, because a node can receive packets from several neighbours. Overall, history buffers significantly improved the overall performance of detecting violations in such an approach with small of history size.

Chapter 8

Discussion

Several consistent characteristics have been noticed throughout the research, even across very different techniques. These similarities express specific techniques, challenges, and concepts that may be applied to monitoring and detection predicates in general. This chapter provides an insight at these common patterns and explain why they are significant.

Q1. In this work, we have assumed links to be symmetric, i.e., bidirectional. Will the algorithms still work if the links are asymmetric?

A1. If the network has bidirectional links, then it means that messages can only travel in one direction only. This will have impact on several aspects of the work. For example, the cluster heads will be different (or there may not be a cluster head). The data collection heuristics proposed may not work if the data does not reach the cluster heads.

Q2. One of the major drawbacks of WSN is that one or more nodes in the network may crash or fail to respond. Is it possible that these algorithms will be effective in this situation?

A2. It will not work if clusterheads fail because monitoring data will be lost. Despite the fact that some research works have contributed to solving this problem in general, most clustering algorithms are unable to reselect cluster heads due to the energy required to restart the algorithm across the entire network and the difficulty in determining which part of the network is not functioning. Therefore, node failure in a WSN is a significant issue that needs more resources and effort, and it is outside the scope of this thesis.

Q3. Is this work capable of detecting violations and working in a protocol that is similar to or equivalent to TDMA/TSCH (MAC layer)?

A3. Yes it will still work as long as cluster heads receive data from nodes and links are bidirectional.

Q4. What if an application captures predicates with a large radius of influence? How does our work deal with this issue?

A4. It will work by selecting the correct number of cluster heads. If the radius of influence if equal to the network diameter, then the sink can be the cluster head and all monitoring data are sent to the sink, which will then evaluate the predicate. The radius of influence may be increased by adjusting the parameters in the dissemination protocols and changing the parameters in the monitors selection phase at the beginning.

Chapter 9

Conclusion

The objective of this thesis was to develop an approach that enables accurate predicate detection in wireless sensor networks. Predicate detection is important in such problems as debugging, mechanism trigger. Based on the explained challenges, we split the predicate detection problem along two dimensions, namely (i) monitor selection problem and (ii) data collection problem. We started by presenting a novel formalization of the monitor selection model. We showed that the MSP problem is intractable and thus a heuristic can be used. We showed that our proposed heuristic performs at least as well as another well-known protocol called FLOC. For the second problem, that of data collection, we develop multiple dissemination protocols to more efficiently propagate state information throughout the network. We evaluate our techniques using two different applications that are very different in properties. The first application is a TDMA algorithm, for which correctness predicates are well-known. The other application is a temperature application to monitor a temperature predicate in a ventilation application. Finally, in order to examine the overall performance and correctness of each implemented protocol and algorithm, we perform extensive simulation and physical hardware (i.e., testbed) experiments.

Our results show that our approach outperform state of the art detection techniques. Specifically, these detection protocols provide high performance and can capture more than 80% of violations in different nodes size. The aim is also to reduce overall overhead when implementing new detection protocols. This means that the techniques must address redundancy messages as well as messages lost during detection. The monitor's ability to receive the response and necessary data in less time is another factor in detecting violations in WSN. Specific conditions (predicates) are implemented that must be met when the detecting protocols are executing in a real-time setting.

9.1 Future Work

Expanding on the ideas offered in this thesis could lead to a number of new directions of research.This section will describe a number of immediate areas in which the work presented could be explored further.However, this strategy must be handled with caution, as predicate requirements and modifying dissemination protocol codes may have an impact on overall performance.

9.1.1 New Configurations

Since parameters and settings have an impact on overall performance, changing configurations may generate new ideas and encourage the investigation of new methodologies. Expanding the distance parameters and the time required to flood and request by a monitor, for example. Even the time it takes for a monitor to collect and evaluate data is a major factor in wireless sensor network detection protocols.

9.1.2 File Size

Since the debugging system should be light weight when deployed in hardware motes, further work is expected to be done on this problem. This problem occurs in any recently implemented debugging system, where the routing protocol must always be applied on top of the application or protocol. However, another approach in this project may be to further optimize the code in order to reduce the memory size.

9.1.3 Expanding Dissemination Protocols

Several techniques and protocols for collecting and transmitting data in WSN have not yet been addressed or implemented in this thesis for dissemination and network library, particularly for detection predicates in WSN. Even if a strategy for selecting specific monitors exists in the WSN, these methods are still required to implement dissemination protocols in order to propagate the predicates. We have already discussed and compared some of the existing protocols for propagating predicates in WSN, such as CTP and Trickle.However, future research opportunities exist, such as artificial intelligence algorithms and others.

Bibliography

[1] Zolertia mote specifications. URL `http://wiki.zolertia.com/wiki/index.php/Main_Page`.

[2] J. C. Laprie A. Avizienis, V. Magnus U and B. Randell. Fundamental concepts of dependability. In *Research Report No 1145*, April 2001.

[3] Thiemo Voigt Muneeb Ali Adam Dunkels, Oliver Schmidt. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. *ACM SenSys*, 2006.

[4] K. Adere and G.R. Murthy. Solving the hidden and exposed terminal problems using directional-antenna based mac protocol for wireless sensor networks. In *Wireless And Optical Communications Networks (WOCN), 2010 Seventh International Conference On*, pages 1–5, 2010. doi: 10.1109/WOCN.2010.5587352.

[5] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. Fit iot-lab: A large scale open experimental iot testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464, 2015. doi: 10.1109/WF-IoT.2015.7389098.

[6] advanticsys. CM5000. Online, 2011. URL `http://www.advanticsys.com/shop/mtmcm5000msp-p-14.html`.

[7] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325 – 349, 2005. ISSN 1570-8705. doi: 10.1016/j.adhoc.2003.09.010. URL `http://www.sciencedirect.com/science/article/pii/S1570870503000738`.

[8] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002. ISSN 1389-1286. doi: https://doi.org/10.1016/S1389-1286(01)00302-4. URL `https://www.sciencedirect.com/science/article/pii/S1389128601003024`.

[9] J.N. Al-Karaki and A.E. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, 11(6):6–28, dec. 2004. ISSN 1536–1284. doi: 10.1109/MWC.2004.1368893.

[10] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 1, pages 32–41 vol.1, March 2000. doi: 10.1109/INFCOM.2000.832171.

[11] Sariga Arjunan and Sujatha Pothula. A survey on unequal clustering protocols in wireless sensor networks. *Journal of King Saud University - Computer and Information Sciences*, 31(3):304–317, 2019. ISSN 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2017.03.006. URL `https://www.sciencedirect.com/science/article/pii/S1319157817300927`.

[12] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010. ISSN 1389-1286. doi: 10.1016/j.comnet.2010.05.010. URL `http://www.sciencedirect.com/science/article/pii/S1389128610001568`.

[13] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004. ISSN 1545-5971. doi: 10.1109/TDSC.2004.2.

[14] Hakan Bagci and Adnan Yazici. An energy aware fuzzy unequal clustering algorithm for wireless sensor networks. In *International Conference on Fuzzy Systems*, pages 1–8, 2010. doi: 10.1109/FUZZY.2010.5584580.

[15] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, and Mario Gerla. Glomosim: A scalable network simulation environment. Technical report, 1999.

[16] Gyorgy Balogh, Gabor Pap, and Miklos Maroti. JProwler, September 2004. URL `http://w3.isis.vanderbilt.edu/projects/nest/jprowler`.

[17] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 1028–1037 vol.2, April 2001. doi: 10.1109/INFCOM.2001.916296.

[18] Shilpa Bansod and Jean Mayo. A distributed algorithm for unstable global predicate evaluation with approximately synchronized clocks. *Studia Informatica Universalis*, 3(2):151–168, 2004.

[19] R. Barry et al. Freertos. volume 2, 2009.

[20] B. Bates, A. Keating, and R. Kinicki. Energy analysis of four wireless sensor network mac protocols. In *Wireless and Pervasive Computing (ISWPC), 2011 6th International Symposium on*, pages 1–6, feb. 2011. doi: 10.1109/ISWPC.2011.5751321.

[21] Jacob Beal. A robust amorphous hierarchy from persistent nodes, 2003.

[22] J. Beutel, M. Dyer, L. Meier, and L. Thiele. Scalable topology control for deployment-support networks. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 359–363, 2005.

[23] Michael D. Bond, Nicholas Nethercote, Stephen W. Kent, Samuel Z. Guyer, and Kathryn S. McKinley. Tracking bad apples: reporting the origin of null and undefined value errors. *SIGPLAN Not.*, 42(10):405–422, October 2007. ISSN 0362-1340. doi: 10.1145/1297105.1297057. URL `http://doi.acm.org/10.1145/1297105.1297057`.

[24] V. Michael Bove Jr. and Ben Dalton. Audio-based self-localization for ubiquitous sensor networks. In *Audio Engineering Society Convention 118*, May 2005. URL `http://www.aes.org/e-lib/browse.cfm?elib=13061`.

[25] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 307–320, New York, NY, USA, 2006. ACM. ISBN 1-59593-343-3. doi: 10.1145/1182807.1182838. URL `http://doi.acm.org/10.1145/1182807.1182838`.

[26] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3 (1):63–75, February 1985. ISSN 0734-2071. doi: 10.1145/214451.214456. URL `http://doi.acm.org/10.1145/214451.214456`.

[27] Pilsoon Choi, Hyung Chul Park, Sohyeong Kim, Sungchung Park, Ilku Nam, Tae Wook Kim, Seokjong Park, Sangho Shin, Myeung Su Kim, Kyucheol Kang, Yeonwoo Ku, Hyokjae Choi, Sook Min Park, and Kwyro Lee. An experimental coin-sized radio for extremely low-power wpan

(ieee 802.15.4) application at 2.4 ghz. *Solid-State Circuits, IEEE Journal of*, 38(12):2258 – 2268, dec. 2003. ISSN 0018-9200. doi: 10.1109/JSSC.2003.819083.

[28] Chee-Yee Chong and S.P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003. ISSN 0018-9219. doi: 10.1109/JPROC.2003.814918.

[29] Robert Cooper and Keith Marzullo. Consistent detection of global predicates. *SIGPLAN Not.*, 26(12):167–174, December 1991. ISSN 0362-1340. doi: 10.1145/127695.122774. URL `http://doi.acm.org/10.1145/127695.122774`.

[30] John H. Davies. *MSP430 Microcontroller Basics*. Newnes, USA, 2nd edition, 2015. ISBN 008098326X.

[31] M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani. Design and analysis of a fast local clustering service for wireless sensor networks. In *First International Conference on Broadband Networks*, pages 700–709, Oct 2004. doi: 10.1109/BROADNETS.2004.30.

[32] M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani. A fault-local self-stabilizing clustering service for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):912–922, Sep. 2006. doi: 10.1109/TPDS.2006.113.

[33] M. Dhanaraj and C. Siva Ram Murthy. On achieving maximum network lifetime through optimal placement of cluster-heads in wireless sensor networks. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 3142–3147, 2007. doi: 10.1109/ICC.2007.521.

[34] M. Ding, X. Cheng, and G. Xue. Aggregation tree construction in sensor networks. In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 4, pages 2168 – 2172 Vol.4, oct. 2003. doi: 10.1109/VETECF.2003.1285913.

[35] B. Djamaa and M. Richardson. Optimizing the trickle algorithm. *IEEE Communications Letters*, 19(5):819–822, 2015. doi: 10.1109/LCOMM.2015.2408339.

[36] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, 2004. doi: 10.1109/LCN.2004.38.

[37] Adam Dunkels. Full TCP/IP for 8-bit Architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 85–98, New York, NY, USA, 2003. ACM. doi: 10.1145/1066116.1066118. URL http://doi.acm.org/10.1145/1066116.1066118.

[38] Adam Dunkels, Juan Alonso, and Thiemo Voigt. Making TCP/IP Viable for Wireless Sensor Networks. *Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN2004)*, 2004.

[39] Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 15–28, New York, NY, USA, 2006. ACM. ISBN 1-59593-343-3. doi: 10.1145/1182807.1182810. URL http://doi.acm.org/10.1145/1182807.1182810.

[40] Adam Dunkels, Fredrik Österlind, and Zhitao He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 335–349, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-763-6. doi: 10.1145/1322263.1322295. URL http://doi.acm.org/10.1145/1322263.1322295.

[41] Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th workshop on Embedded networked sensors*, EmNets '07, pages 28–32, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-694-3. doi: 10.1145/1278972.1278979. URL http://doi.acm.org/10.1145/1278972.1278979.

[42] Mathilde Durvy, Julien Abeillé, Patrick Wetterwald, Colin O'Flynn, Blake Leverett, Eric Gnoske, Michael Vidales, Geoff Mulligan, Nicolas Tsiftes, Niclas Finne, and Adam Dunkels. Making sensor networks ipv6 ready. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 421–422, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-990-6. doi: 10.1145/1460412.1460483. URL http://doi.acm.org/10.1145/1460412.1460483.

[43] Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin, and Philipp Blum. Deployment support network. In Koen Langendoen and Thiemo Voigt, editors, *Wireless Sensor Networks*, pages 195–211, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-69830-2.

[44] A. S. Martinez-Sala P. Pavon-Marino E. Egea-Lopez, J. Vales-Alonso and J. Garc'-Haro. Simulation tools for wireless sensor networks. *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS05)*, 2005.

[45] Cheng Tien Ee, Rodrigo Fonseca, Sukun Kim, Daekyeong Moon, Arsalan Tavakoli, David Culler, Scott Shenker, and Ion Stoica. A modular network layer for sensorsets. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 249–262, Berkeley, CA, USA, 2006. USENIX Association. ISBN 1-931971-47-1. URL `http://dl.acm.org/citation.cfm?id=1298455.1298479`.

[46] Emre Ertin, Anish Arora, Rajiv Ramnath, Vinayak Naik, Sandip Bapat, Vinod Kulathumani, Mukundan Sridharan, Hongwei Zhang, Hui Cao, and Mikhail Nesterenko. Kansei: a testbed for sensing at scale. pages 399–406, 01 2006. doi: 10.1109/IPSN.2006.243879.

[47] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 263–270, New York, NY, USA, 1999. ACM. ISBN 1-58113-142-9. doi: 10.1145/313451.313556. URL `http://doi.acm.org/10.1145/313451.313556`.

[48] D. Evans and D. Larochelle. Improving security using extensible lightweight static analysis. *Software, IEEE*, 19(1):42–51, 2002. ISSN 0740-7459. doi: 10.1109/52.976940.

[49] J. Fagerström. Design and test of distributed applications. In *Proceedings of the 10th international conference on Software engineering*, ICSE '88, pages 88–92, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press. ISBN 0-89791-258-6. URL `http://dl.acm.org/citation.cfm?id=55823.55833`.

[50] V. K. Garg and B. Waldecker. Detection of weak unstable predicates in distributed programs. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):299–307, March 1994. doi: 10.1109/71.277788.

[51] V.K. Garg and B. Waldecker. Detection of strong unstable predicates in distributed programs. *Parallel and Distributed Systems, IEEE Transactions on*, 7(12):1323–1333, 1996. ISSN 1045-9219. doi: 10.1109/71.553309.

[52] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. Emstar: A software environment for developing and deploying wireless sensor networks. In *Proceedings of the*

Annual Conference on USENIX Annual Technical Conference, ATEC '04, page 24, USA, 2004. USENIX Association.

[53] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 1–14, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-519-2. doi: 10.1145/1644038.1644040. URL http://doi.acm.org/10.1145/1644038.1644040.

[54] A.J. Goldsmith and S.B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *Wireless Communications, IEEE*, 9(4):8–27, 2002. ISSN 1536-1284. doi: 10.1109/MWC.2002.1028874.

[55] V.C. Gungor and G.P. Hancke. Industrial wireless sensor networks: Challenges, design principles, and technical approaches. *Industrial Electronics, IEEE Transactions on*, 56(10):4258–4265, 2009. ISSN 0278-0046. doi: 10.1109/TIE.2009.2015754.

[56] Ştefan Gună, Luca Mottola, and Gian Pietro Picco. Dice: Monitoring global invariants with wireless sensor networks. *ACM Trans. Sen. Netw.*, 10(4):54:1–54:34, June 2014. ISSN 1550-4859. doi: 10.1145/2509434. URL http://doi.acm.org/10.1145/2509434.

[57] Shuo Guo, Ziguo Zhong, and Tian He. Find: Faulty node detection for wireless sensor networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 253–266, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-519-2. doi: 10.1145/1644038.1644064. URL http://doi.acm.org/10.1145/1644038.1644064.

[58] Vrinda Gupta and Rajoo Pandey. An improved energy aware distributed unequal clustering protocol for heterogeneous wireless sensor networks. *Engineering Science and Technology, an International Journal*, 19(2):1050–1058, 2016. ISSN 2215-0986. doi: https://doi.org/10.1016/j.jestch.2015.12.015. URL https://www.sciencedirect.com/science/article/pii/S2215098616000045.

[59] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, April 1950. URL http://www3.alcatel-lucent.com/bstj/vol29-1950/articles/bstj29-2-147.pdf.

[60] M. J. Handy, M. Haase, and D. Timmermann. Low energy adaptive clustering hierarchy with deterministic cluster-head selection. In *4th In-*

ternational Workshop on Mobile and Wireless Communications Network, pages 368–372, Sep. 2002. doi: 10.1109/MWCN.2002.1045790.

[61] W.B. Heinzelman, A.P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002. doi: 10.1109/TWC.2002.804190.

[62] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10 pp. vol.2–, 2000. doi: 10.1109/HICSS.2000.926982.

[63] Douglas Herbert, Vinaitheerthan Sundaram, Yung-Hsiang Lu, Saurabh Bagchi, and Zhiyuan Li. Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2(3): 8, sep 2007. ISSN 1556-4665. doi: 10.1145/1278460.1278462.

[64] J.L. Hill and D.E. Culler. Mica: a wireless platform for deeply embedded networks. *Micro, IEEE*, 22(6):12 – 24, nov/dec 2002. ISSN 0272-1732. doi: 10.1109/MM.2002.1134340.

[65] Suzanne Hoppough. Shelf life. *Forbes Mag*, 2006.

[66] Y.T. Hou, Yi Shi, H.D. Sherali, and S.F. Midkiff. On energy provisioning and relay node placement for wireless sensor networks. *IEEE Transactions on Wireless Communications*, 4(5):2579–2590, 2005. doi: 10.1109/TWC.2005.853969.

[67] J. Hui and R. Kelsey. Multicast protocol for low-power and lossy networks (mpl). document RFC 7731, 2016.

[68] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, page 81–94, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138792. doi: 10.1145/1031495.1031506. URL `https://doi.org/10.1145/1031495.1031506`.

[69] Jonathan W. Hui and David E. Culler. Ip is dead, long live ip for wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 15–28, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-990-6. doi: 10.1145/1460412.1460415. URL `http://doi.acm.org/10.1145/1460412.1460415`.

[70] Konrad Iwanicki and Maarten van Steen. Towards a versatile problem diagnosis infrastructure for largewireless sensor networks. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, pages 845–855, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-76890-6.

[71] N. Finne F. Osterlind J. Eriksson, A. Dunkels and T. Voigt. Mspsim–an extensible simulator for msp430-equipped sensor boards. volume in Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, page 27, 2007.

[72] Gunmo Jeong, Mingyu Park, and Jeongyeup Paek. A²-trickle: Adaptive amp; aligned trickle for rapid and reliable dissemination in low-power wireless networks. *IEEE Access*, 8:214374–214382, 2020. doi: 10.1109/ACCESS.2020.3040160.

[73] Yan Jin, Ling Wang, Yoohwan Kim, and Xiaozong Yang. Eemc: An energy-efficient multi-level clustering algorithm for large-scale wireless sensor networks. *Computer Networks*, 52(3):542 – 562, 2008. ISSN 1389-1286. doi: 10.1016/j.comnet.2007.10.005. URL `http://www.sciencedirect.com/science/article/pii/S1389128607002927`.

[74] John Jubin and J.D. Tornow. The darpa packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, 1987. ISSN 0018-9219. doi: 10.1109/PROC.1987.13702.

[75] Raja Jurdak, Antonio G. Ruzzelli, Alessio Barbirato, and Samuel Boivineau. Octopus: monitoring, visualization, and control of sensor networks. *Wireless Communications and Mobile Computing*, 11(8):1073–1091, 2011. doi: 10.1002/wcm.826. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/wcm.826`.

[76] Mohammad Khan, Hieu Le, Hossein Ahmadi, Tarek Abdelzaher, and Jiawei Han. Dustminer: Troubleshooting interactive complexity bugs in sensor networks. pages 99–112, 01 2008. doi: 10.1145/1460412.1460423.

[77] Mohammad Maifi Hasan Khan, Hieu K. Le, Michael LeMay, Parya Moinzadeh, Lili Wang, Yong Yang, Dong K. Noh, Tarek Abdelzaher, Carl A. Gunter, Jiawei Han, and Xin Jin. Diagnostic powertracing for sensor node failure analysis. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, page 117–128, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589886. doi: 10.1145/1791212.1791227. URL `https://doi.org/10.1145/1791212.1791227`.

[78] G. Khanna, P. Varadharajan, and S. Bagchi. Self checking network protocols: a monitor based approach. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, pages 18–30, Oct 2004. doi: 10.1109/RELDIS.2004.1353000.

[79] Hyung-Sin Kim, Jeonggil Ko, David E. Culler, and Jeongyeup Paek. Challenging the ipv6 routing protocol for low-power and lossy networks (rpl): A survey. *IEEE Communications Surveys Tutorials*, 19(4):2502–2525, 2017. doi: 10.1109/COMST.2017.2751617.

[80] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks performance evaluation and optimization. *Computer Networks (1976)*, 1(3):155–174, 1977. ISSN 0376-5075. doi: https://doi.org/10.1016/0376-5075(77)90002-2. URL `https://www.sciencedirect.com/science/article/pii/0376507577900022`.

[81] W. Kluge, F. Poegel, H. Roller, M. Lange, T. Ferchland, L. Dathe, and D. Eggert. A fully integrated 2.4-ghz ieee 802.15.4-compliant transceiver for zigbee trade; applications. *Solid-State Circuits, IEEE Journal of*, 41 (12):2767 –2775, dec. 2006. ISSN 0018-9200. doi: 10.1109/JSSC.2006.884802.

[82] M. Bala Krishna and M. N. Doja. Self-organized energy conscious clustering protocol for wireless sensor networks. In *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, pages 521–526, 2012.

[83] L. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, pages 575–578, 2002. doi: 10.1109/ICDCSW.2002.1030829.

[84] Veljko Krunic, Eric Trumpler, and Richard Han. Nodemd: diagnosing node-level faults in remote wireless sensor systems. In *MobiSys '07*, 2007.

[85] S.S. Kulkarni and Limin Wang. Mnp: Multihop network reprogramming service for sensor networks. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 7 –16, june 2005. doi: 10.1109/ICDCS.2005.50.

[86] Eugene L. Lawler, Karl N. Levitt, and James Turner. Module clustering to minimize delay in digital networks. *Computers, IEEE Transactions on*, C-18(1):47–57, 1969. ISSN 0018-9340. doi: 10.1109/T-C.1969.222524.

[87] T.J. LeBlanc and J.M. Mellor-Crummey. Debugging parallel programs with instant replay. *Computers, IEEE Transactions on*, C-36(4):471–482, 1987. ISSN 0018-9340. doi: 10.1109/TC.1987.1676929.

[88] B. Lee and K. Lim. An adaptive data reporting scheme considering node contexts in wireless sensor networks. In *2011 First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering*, pages 382–386, May 2011. doi: 10.1109/CNSI.2011.23.

[89] Timothy C. Lethbridge and Robert Laganière. *Object-oriented software engineering*. McGraw-Hill Higher Education, 2nd edition, 2005. ISBN 0-07-70109082.

[90] J. Leu, T. Chiang, M. Yu, and K. Su. Energy efficient clustering scheme for prolonging the lifetime of wireless sensor network with isolated nodes. *IEEE Communications Letters*, 19(2):259–262, Feb 2015. doi: 10.1109/LCOMM.2014.2379715.

[91] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In Werner Weber, JanM. Rabaey, and Emile Aarts, editors, *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-23867-6. doi: 10.1007/3-540-27139-2_7. URL `http://dx.doi.org/10.1007/3-540-27139-2_7`.

[92] Philip Levis and David Culler. Maté: A tiny virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(5):85–95, oct 2002. ISSN 0163-5980. doi: 10.1145/635508.605407. URL `https://doi.org/10.1145/635508.605407`.

[93] Philip Levis and Nelson Lee. Tossim: a simulator for tinyos networks. 12 2003.

[94] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI*, pages 15–28, 2004.

[95] Phil Lewis, Thomas Heide Clausen, Jonathan Hui, Omprakash Gnawali, and Jeonggil Ko. Rfc6206: The trickle algorithm. 03 2011.

[96] Hong Li, HongYi Yu, and ANa Liu. A tree based data collection scheme for wireless sensor network. In *Networking, International Conference on Systems and International Conference on Mobile Communications*

and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on, pages 119–119, 2006. doi: 10.1109/ICNICONSMCL.2006.36.

[97] Huan Li, Yanlei Liu, Weifeng Chen, Weijia Jia, Bing Li, and Junwu Xiong. Coca: Constructing optimal clustering architecture to maximize sensor network lifetime. *Computer Communications*, 36(3):256–268, 2013. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2012.10.006. URL `https://www.sciencedirect.com/science/article/pii/S0140366412003751`.

[98] Chung-Yu Liu. A study of flight-critical computer system recovery from space radiation-induced error. *Aerospace and Electronic Systems Magazine, IEEE*, 17(7):19–25, 2002. ISSN 0885-8985. doi: 10.1109/MAES.2002.1017791.

[99] H. Liu, L. Selavo, and J. Stankovic. Seedtv: Deployment-time validation for wireless sensor networks. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 23–27, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-694-3. doi: 10.1145/1278972.1278978. URL `http://doi.acm.org/10.1145/1278972.1278978`.

[100] Logambigai and A. R Kannan. Fuzzy logic based unequal clustering for wireless sensor networks. *Wireless Networks*, 22, 2016.

[101] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, page 88–97, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581135890. doi: 10.1145/570738.570751. URL `https://doi.org/10.1145/570738.570751`.

[102] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM. ISBN 1-58113-589-0. doi: 10.1145/570738.570751.

[103] S. Mccanne, S. Floyd, and K. Fall. ns-2 (network simulator 2). Online, 2011. URL `http://nsnam.isi.edu/nsnam/index.php/Main_Page`.

[104] Aleksandar Milenković, Chris Otto, and Emil Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Comput. Commun.*, 29(13-14):2521–2533, August 2006. ISSN 0140-3664.

[105] Rodolfo Miranda Pereira, Linnyer Ruiz, and Maria Ghizoni. Mannasim: A ns-2 extension to simulate wireless sensor network. 04 2015.

[106] Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43(3):19:1–19:51, April 2011. ISSN 0360-0300. doi: 10.1145/1922649.1922656. URL `http://doi.acm.org/10.1145/1922649.1922656`.

[107] M. Muhlhauser. Software engineering for distributed applications: the design project. In *Software Engineering, 1988., Proceedings of the 10th International Conference on*, pages 93–101, 1988. doi: 10.1109/ICSE.1988.93692.

[108] S.A. Munir, Biao Ren, Weiwei Jiao, Bin Wang, Dongliang Xie, and Man Ma. Mobile wireless sensor network: Architecture and enabling technologies for ubiquitous computing. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 2, pages 113 –120, may 2007. doi: 10.1109/AINAW.2007.257.

[109] Radhika Nagpal and Daniel Coore. An algorithm for group formation in an amorphous computer. In *Proceedings of the Tenth International Conference on Parallel and Distributed Systems (PDCS*, 1998.

[110] H. Nama, Mung Chiang, and N. Mandayam. Utility-lifetime trade-off in self-regulating wireless sensor networks: A cross-layer design approach. In *Communications, 2006. ICC '06. IEEE International Conference on*, volume 8, pages 3511–3516, 2006. doi: 10.1109/ICC.2006.255616.

[111] Nicholas Nethercote and Julian Seward. How to shadow every byte of memory used by a program. In *Proceedings of the 3rd international conference on Virtual execution environments*, VEE '07, pages 65–74, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-630-1. doi: 10.1145/1254810.1254820. URL `http://doi.acm.org/10.1145/1254810.1254820`.

[112] Kevin Ni, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, Mark Hansen, and Mani Srivastava. Sensor network data fault types. *ACM Trans. Sen. Netw.*, 5(3), June 2009. ISSN 1550-4859. doi: 10.1145/1525856.1525863. URL `https://doi.org/10.1145/1525856.1525863`.

[113] Dragoş Niculescu and Badri Nath. Dv based positioning in ad hoc networks. *Telecommunication Systems*, 22(1):267–280, Jan 2003. ISSN

1572-9451. doi: 10.1023/A:1023403323460. URL https://doi.org/10.1023/A:1023403323460.

[114] L. Oliveira and J. Rodrigues. Wireless sensor networks: A survey on environmental monitoring. In *JCM, vol. 6, pp. 143–151, 04 2011*, pages 800–804, Dec 2007.

[115] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648, Nov 2006. doi: 10.1109/LCN.2006.322172.

[116] Jeongyeup Paek, Ben Greenstein, Omprakash Gnawali, Ki-Young Jang, August Joki, Marcos Vieira, John Hicks, Deborah Estrin, Ramesh Govindan, and Eddie Kohler. The tenet architecture for tiered sensor networks. *ACM Trans. Sen. Netw.*, 6(4), jul 2010. ISSN 1550-4859. doi: 10.1145/1777406.1777413. URL https://doi.org/10.1145/1777406.1777413.

[117] Nikolaos A. Pantazis and Dimitrios D. Vergados. A survey on power control issues in wireless sensor networks. *IEEE Communications Surveys Tutorials*, 9(4):86–107, 2007. doi: 10.1109/COMST.2007.4444752.

[118] B. Parkinson and J. Spilker. Global positioning system: Theory and application. In *Am. Inst. of Aeronautics and Astronautics*, 1996.

[119] Kunjan Patel, Lim Jong Chern, C.J. Bleakley, and Wim Vanderbauwhede. Maw: A reliable lightweight multi-hop wireless sensor network routing protocol. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 2, pages 487–493, 2009. doi: 10.1109/CSE.2009.104.

[120] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 95–107, New York, NY, USA, 2004. ACM. ISBN 1-58113-879-2. doi: 10.1145/1031495.1031508. URL http://doi.acm.org/10.1145/1031495.1031508.

[121] Joseph Polastre, Jonathan Hui, Philip Levis, Jerry Zhao, David Culler, Scott Shenker, and Ion Stoica. A unifying link abstraction for wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 76–89, New York, NY, USA, 2005. ACM. ISBN 1-59593-054-X. doi: 10.1145/1098918.1098928. URL http://doi.acm.org/10.1145/1098918.1098928.

[122] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras. Atemu: a fine-grained sensor network simulator. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 145–152, 2004.

[123] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 32–43, New York, NY, USA, 2000. ACM. ISBN 1-58113-197-6. doi: 10.1145/345910.345917. URL http://doi.acm.org/10.1145/345910.345917.

[124] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, page 255–267, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 159593054X. doi: 10.1145/1098918.1098946. URL https://doi.org/10.1145/1098918.1098946.

[125] Matthias Ringwald and Kay Römer. Snif: Sensor network inspection framework. Technical report, 2006.

[126] K. Romer and F. Mattern. The design space of wireless sensor networks. *Wireless Communications, IEEE*, 11(6):54–61, 2004. ISSN 1536-1284. doi: 10.1109/MWC.2004.1368897.

[127] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. Distributed Channel Allocation Algorithms for Wireless Sensor Networks. Technical report, Washington University in St. Louis, 2011. URL http://cse.wustl.edu/Research/Lists/Technical%20Reports/Attachments/958/Channel_WSN.pdf.

[128] Julian Seward, Nick Nethercote, and J Fitzhardinge. Valgrind, an open-source memory debugger for x86-gnu/linux. Online, 2004. URL http://www.ukuug.org/events/linux2002/papers/html/valgrind.

[129] Aggeliki Sgora, Dimitrios J. Vergados, and Dimitrios D. Vergados. A survey of tdma scheduling schemes in wireless multihop networks. *ACM Comput. Surv.*, 47(3), April 2015. ISSN 0360-0300. doi: 10.1145/2677955. URL https://doi.org/10.1145/2677955.

[130] S. Soro and W.B. Heinzelman. Prolonging the lifetime of wireless sensor networks via unequal clustering. In *19th IEEE International Parallel*

and Distributed Processing Symposium, pages 8 pp.–, 2005. doi: 10.1109/IPDPS.2005.365.

[131] Jessica Staddon, Dirk Balfanz, and Glenn Durfee. Efficient tracing of failed nodes in sensor networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, page 122–130, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581135890. doi: 10.1145/570738.570756. URL `https://doi.org/10.1145/570738.570756`.

[132] Richard Stallman and Roland H Pesch. *The GDB Manual: The GNU Source-level Debugger*. Free Software Foundation, 4.03 edition, January 1992. ISBN 9781882114115.

[133] J.A. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and obligations for physical computing systems. *Computer*, 38(11):23–31, 2005. doi: 10.1109/MC.2005.386.

[134] Robert Szewczyk, Joseph Polastre, Alan M. Mainwaring, and David E. Culler. Lessons from a sensor network expedition. In *EWSN*, pages 307–322, 2004.

[135] Matthew Tancreti, Vinaitheerthan Sundaram, Saurabh Bagchi, and Patrick Eugster. Software-only system-level record and replay in wireless sensor networks. pages 400–401, 04 2015. doi: 10.1145/2737095.2741839.

[136] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2003. ISBN 0130661023.

[137] Gilman Tolle and David Culler. Design of an application-cooperative management system for wireless sensor networks. volume 2005, pages 121 – 132, 01 2005. ISBN 0-7803-8801-1. doi: 10.1109/EWSN.2005.1462004.

[138] Wei K. Tsai, G. Huang, John K. Antonio, and Wei-T Tsai. Distributed aggregation/disaggregation algorithms for optimal routing in data networks. In *American Control Conference, 1988*, pages 1799–1804, 1988.

[139] John Tsitsiklis. Problems in decentralized decision making and computation. *Ph.D. dissertation, MIT*, 1984. doi: http://web.mit.edu/jnt/www/Papers/PhD-84-jnt.pdf.

[140] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and*

Systems Workshops, Simutools '08, Brussels, BEL, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 9789639799202.

[141] S. Venkatesan and B. Dathan. Testing and debugging distributed programs using global predicates. *Software Engineering, IEEE Transactions on*, 21(2):163–177, 1995. ISSN 0098-5589. doi: 10.1109/32.345831.

[142] J. Viega, J. T. Bloch, Y. Kohno, and Gary McGraw. Its4: a static vulnerability scanner for c and c++ code. In *Computer Security Applications, 2000. ACSAC '00. 16th Annual Conference*, pages 257–267, 2000. doi: 10.1109/ACSAC.2000.898880.

[143] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: a wireless sensor network testbed. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, pages 483–488, 2005.

[144] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 381–396, Berkeley, CA, USA, 2006. USENIX Association. ISBN 1-931971-47-1. URL `http://dl.acm.org/citation.cfm?id=1298455.1298491`.

[145] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 14–27, New York, NY, USA, 2003. ACM. ISBN 1-58113-707-9. doi: 10.1145/958491.958494. URL `http://doi.acm.org/10.1145/958491.958494`.

[146] Hui Xia, Ruihua Zhang, Jia Yu, and Zhen-Kuan Pan. Energy-efficient routing algorithm based on unequal clustering and connected graph in wireless sensor networks. *International Journal of Wireless Information Networks*, 23:141–150, 2016.

[147] Jing Yang, Mary Soffa, Leo Selavo, and Kamin Whitehouse. Clairvoyant: A comprehensive source-level debugger for wireless sensor networks. pages 189–203, 01 2007. doi: 10.1145/1322263.1322282.

[148] Mohamed Younis, Moustafa Youssef, and Khaled Arisha. Energy-aware management for cluster-based sensor networks. *Computer Networks*, 43 (5):649–668, 2003. ISSN 1389-1286. doi: https://doi.org/10.1016/S1389-

1286(03)00305-0. URL https://www.sciencedirect.com/science/article/pii/S1389128603003050.

[149] M. Youssef, A. Youssef, and M. Younis. Overlapping multihop clustering for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1844–1856, Dec 2009. doi: 10.1109/TPDS.2009.32.

[150] Jiguo Yu, Yingying Qi, Gang Wang, Qiang Guo, and Xin Gu. An energy-aware distributed unequal clustering protocol for wireless sensor networks. *International Journal of Distributed Sensor Networks*, 7(1): 202145, 2011. doi: 10.1155/2011/202145. URL https://doi.org/10.1155/2011/202145.

[151] Qi Y. Wang G Yu, J. An energy-driven unequal clustering protocol for heterogeneous wireless sensor networks. *Control Theory Appl*, 9, 2011.

[152] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 1–13, New York, NY, USA, 2003. ACM. ISBN 1-58113-707-9. doi: 10.1145/958491.958493. URL http://doi.acm.org/10.1145/958491.958493.

[153] Li Zheng. Zigbee wireless sensor network in industrial applications. In *SICE-ICASE, 2006. International Joint Conference*, pages 1067–1070, 2006. doi: 10.1109/SICE.2006.315751.

[154] Yun Zhou, Yuguang Fang, and Yanchao Zhang. Securing wireless sensor networks: a survey. *IEEE Communications Surveys Tutorials*, 10(3): 6–28, 2008. doi: 10.1109/COMST.2008.4625802.