Towards Design of Lightweight Spatio-Temporal Context Algorithms for Wireless Sensor Networks

by

Anahit Martirosyan

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Ph.D. degree in
Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

To the loving memory of my father

Abstract

Context represents any knowledge obtained from Wireless Sensor Networks (WSNs) about the object being monitored (such as time and location of the sensed events). Time and location are important constituents of context as the information about the events sensed in WSNs is comprehensive when it includes spatio-temporal knowledge.

In this thesis, we first concentrate on the development of a suite of lightweight algorithms on temporal event ordering and time synchronization as well as localization for WSNs. Then, we propose an energy-efficient clustering routing protocol for WSNs that is used for message delivery in the former algorithm.

The two problems - temporal event ordering and synchronization - are dealt with together as both are concerned with preserving temporal relationships of events in WSNs. The messages needed for synchronization are piggybacked onto the messages exchanged in underlying algorithms. The synchronization algorithm is tailored to the clustered topology in order to reduce the overhead of keeping WSNs synchronized.

The proposed localization algorithm has an objective of lowering the overhead of DV-hop based algorithms by reducing the number of floods in the initial position estimation phase. It also randomizes iterative refinement phase to overcome the synchronicity of DV-hop based algorithms. The position estimates with higher confidences are emphasized to reduce the impact of erroneous estimates on the neighbouring nodes.

The proposed clustering routing protocol is used for message delivery in the proposed temporal algorithm. Nearest neighbour nodes are employed for inter-cluster communication. The algorithm provides Quality of Service by forwarding high priority messages via the paths with the least cost. The algorithm is also extended for multiple Sink scenario.

The suite of algorithms proposed in this thesis provides the necessary tool for providing spatio-temporal context for context-aware WSNs. The algorithms are lightweight as they aim at satisfying WSN's requirements primarily in terms of energy-efficiency, low latency and fault tolerance. This makes them suitable for emergency response applications and ubiquitous computing.

List of Publications

- A. Boukerche, A. Martirosyan, R. W. N. Pazzi: An Inter-cluster Communication based Energy Aware and Fault Tolerant Protocol for Wireless Sensor Networks. MONET 13(6): 614-626 (2008).

- A. Martirosyan, A. Boukerche, R. W. N. Pazzi: Energy-aware and quality of service-based routing in wireless sensor networks and vehicular ad hoc networks. Annales des Tlcommunications 63(11-12): 669-681 (2008).

- A. Martirosyan and A. Boukerche, Spatio-Temporal Context in Wireless Sensor Networks, pages 293-318, In S. Nikoletseas, J. D. P. Rolim (Eds.), Theoretical Aspects of Distributed Computing in Sensor Networks, Springer Verlag, DOI 10.1007/978-3-642-14849-1 (2011).

- A. Martirosyan, T. Tran, A. Boukerche, Using Data Mining for Building Context-Aware E-Commerce Systems, In I. Lee (editor), Encyclopedia of E-business Development and Management in the Global Economy, IGI Global, vol. III, pages 1021-1029 (2010).

- A. Martirosyan and A. Boukerche, Spatio-Temporal Algorithms in Wireless Sensor Networks, In Proceedings of 25th Biennial Symposium on Communications, 2010.

- A. Martirosyan and A. Boukerche: Location and Time for Building Context-Aware Wireless Sensor Networks, In Proceedings of 8th International Information and Telecommunication Technologies Symposium I2TS 2009: 216-219.

- A. Martirosyan, A. Boukerche: A Lightweight Iterative Positioning Algorithm for Context-Aware Wireless Sensor Networks: Proof of Correctness. GLOBECOM 2009: 1-6.

- A. Martirosyan, A. Boukerche: Performance Evaluation of an Energy-Aware Clustering Protocol for Wireless Sensor Networks. ICPP Workshops 2008: 67-72.

- A. Martirosyan, A. Boukerche, R. W. N. Pazzi: A Taxonomy of Cluster-Based Routing Protocols for Wireless Sensor Networks. ISPAN 2008: 247-253.

- A. Martirosyan, A. Boukerche and R. W. N. Pazzi: Energy Aware and Cluster-based Routing Protocols for Large-Scale Ambient Sensor Networks, In Proceedings of the 1st international conference on Ambient media and systems, AmbiSys 2008: Article No.: 18, ISBN:978-963-9799-16-5.

- A. Boukerche, A. Martirosyan: An Energy-Aware and Fault Tolerant Inter-Cluster Communication Based Protocol for Wireless Sensor Networks. GLOBECOM 2007: 1164-1168.

- A. Boukerche, A. Martirosyan: An Efficient Algorithm for Preserving Events' Temporal Relationships in Wireless Sensor Actor Networks. LCN 2007: 771-780.

- A. Boukerche, A. Martirosyan: An energy efficient and low latency multiple events' propagation protocol for wireless sensor networks with multiple sinks. PE-WASUN 2007: 82-86.

Acknowledgements

This thesis is dedicated to the memory of my father, my kindred spirit and the greatest source of inspiration while working on my PhD, who unfortunately hasn't lived until this day to see me graduate and get my degree.

I am immensely grateful to my supervisor Dr. Boukerche for his professionalism, guidance, encouragement, moral and financial support throughout the work on my PhD. I couldn't have done it without the motivation that I received from my supervisor.

I would like to thank Dr. Richard W. Pazzi for being there for me whenever I needed help. I would also like to thank Hisham El-Kadiki and Daniel Guidoni for assisting me in exploring ns-2 network simulator.

I would like to thank my friends and colleagues at our PARADISE laboratory, who made every day of work at the lab a fruitful and pleasant experience. My special thank you goes to Haifa Maamar for her continuous support and encouragement.

Last but not least, I am very thankful to my family - my husband Grant and children Sergei and Elena - for filling my life with love and a sense of purpose, for always being patient and understanding, and especially - during these four years of my studies. I would like to thank my mother and sister, who despite being miles away from me, were always with me.

Contents

List of Tables

List of Figures

Glossary of Terms

ACK Acknowledgment

AHLoS Ad-Hoc Localization System

AoA Angle of Arrival

APS Ad hoc Positioning System

APTEEN Adaptive threshold sensitive Energy Efficient sensor Network protocol

CH Cluster head

COBC Clustered Ordering by Confirmation

CPEQ Cluster-based Periodic, Event-driven and Query-based protocol for wireless sensor networks

DDBMS Distributed Database Management System

FIFO First In, First Out

FTSP Flooding Time Synchronization Protocol

GAF Geographic Adaptive Fidelity

GPS Global Positioning System

GTSP Gradient Time Synchronization Protocol

ICE Inter-Cluster communication based Energy aware and fault tolerant protocol for wireless sensor networks

LTS Lightweight Time Synchronization

MDS-MAP Multidimensional Scaling MAP

MRP Multipath Routing Protocol

NN Nearest neighbour

NTP Network Time Protocol

OBC Ordering by Confirmation

PEQ Periodic, Event-driven and Query-based protocol for wireless sensor networks

PDA Personal Digital Assistant

PMI Point-Wise Mutual Information

PPS Pulse-Per-Second

QoS Quality of Service

TOMAC Real-Time Message Ordering in WSNs Using the MAC Layer

RSSI Received Signal Strength Indication

RBS Reference Broadcast Synchronization

RPA Robust Positioning Algorithms for distributed ad-hoc wireless sensor networks

RTS Round-Trip Synchronization

SDP Semidefinite Programming

SCT Synchronization for Clustered Topology

TMOS Temporal Message Ordering Scheme

TEEN Threshold sensitive Energy Efficient sensor Network protocol

TDMA Time Division Multiple Access

ToA Time of Arrival

TDoA Time Difference of Arrival

TDS Time Diffusion Synchronization

TTL Time-To-Live

TPSN Timing Sync Protocol for Sensor Networks

UC Ubiquitous Computing

UHMS Ubiquitous Health Monitoring System

UTC Coordinated Universal Time

WMA Weighted Moving Average

WSAN Wireless Sensor Actor Network

WSN Wireless Sensor Network

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Recent advances in micro-sensor development have resulted in the wide use of Wireless Sensor Networks (WSNs) for remote monitoring tasks [5]. WSNs are used for physical environment monitoring, security surveillance, military applications, emergency response and pervasive health monitoring, among other applications. WSNs are usually composed of a large number of nodes that work together to accomplish a sensing task, as shown in Figure 1.1. WSNs are self-organized and infrastructure-free networks. Sensor nodes may be dropped from an airplane in the area that needs to be monitored for certain phenomena such as fire detection and object tracking, for instance. Sensor nodes may be of a large variety based on their functionality: sensors that monitor temperature, humidity, motion, pressure and vital signs, among others.

A sensor node has limited sensing, computing/storing and transmitting (usually radio) capabilities [56], [33]. UC Berkeley Mica mote sensors are mostly used by the research community. The microcontroller of the motes has 128 Kbytes of flash memory and 4 Kbytes of RAM. The radio module in the sensor has communication rate up to 115 Kbps. The sensor node's transmission radius ranges from inches to tens of meters (20 - 30m). It has limited storing capabilities of 4Mbit external flash to store sensor

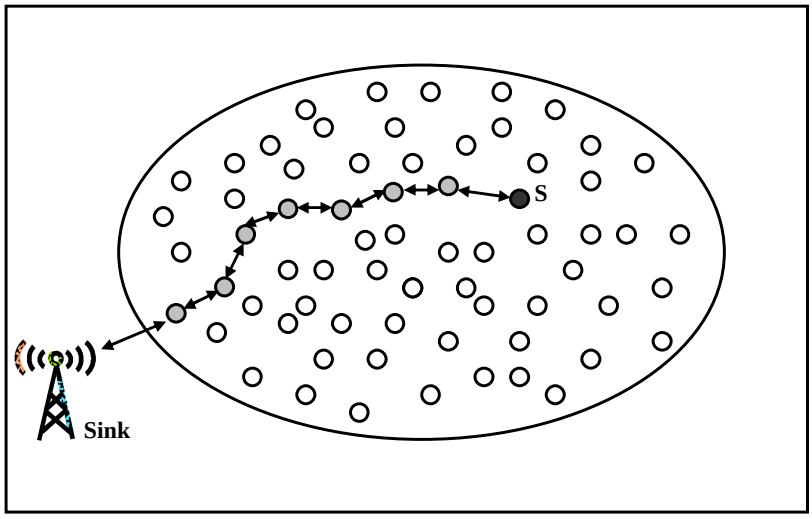


Figure 1.1: Wireless Sensor Network.

data collected. The sensor node runs on a pair of AA batteries, which can last up to two weeks if constantly used. Batteries' lifetime can be extended up to one year if the sensor node uses low duty cycling. The main advantages of sensor nodes are their small size and low cost.

Despite the limited capabilities of sensor nodes, WSNs are able of performing complex sensing tasks thanks to their synergetic effect: sensor nodes communicate with each other to relay messages from the network to the monitoring centre - Sink. The Sink and sensor nodes can use different modes of communication: periodic, query-based and event-driven. In the periodic mode, the sensors' readings are sent to the Sink in a periodic manner (e.g., every 10 minutes). In the query-based mode, the Sink is able to query the network about a particular event of interest (e.g., a query could look like this: "Select sensors in the areas where temperature is greater than 35 degrees"). And in the event-driven mode, the Sink subscribes to an event of interest, and the sensor node notifies the Sink

when that event occurs. For instance, the Sink will be notified when event "Temperature is greater than 35 degrees" occurs at a sensor node S, which monitors temperature, as shown in Figure 1.1. The messages from source nodes to the Sink usually are propagated in a multi-hop fashion as the sensing areas are large and transmission radii of sensor nodes are limited.

Designing algorithms for WSNs is a challenging task. Sensor networks are substantially different from traditional networks. A major limitation of sensor nodes is their limited energy. As already noted, sensors are battery-operated and remain operational for the lifetime of the batteries. Thus, sensor nodes individually and a WSN in general are concerned with conserving scarce energy resources. This imposes strict constraints on the algorithms that are designed for WSNs. Radio module of a sensor node is the most power-consuming module. Communication consumes a significant amount of nodes' energy: either a sensor node sends, receives or hears a message, a comparable amount of energy is dissipated [40]. To conserve energy, the sensor nodes may use low-duty cycling ("sleeping mode"), i.e., they turn their radios off but are still able to sense the environment. Once the event of interest occurs, the nodes turn their radios on in order to send a notification message toward the Sink. When a node transmits a message, energy consumption is proportional to the square of the transmission radius of the node. Thus, long-range transmissions have a higher energy cost. Multi-hop communication is important in WSNs so that long-range transmissions are avoided. Also, local, in-network data processing (such as data aggregation, fusion, filtering, etc.) contributes to energy conservation.

Another fundamental property of WSNs is their dynamics: at a given time, some nodes in the network may be using low-duty cycling, other nodes may deplete their batteries and "die", while more nodes may be added to the network to increase node density and compensate for the nodes that are no longer available. The dynamics need to be accounted for by the algorithms employed in WSNs so that the changes in the topology of WSNs do not affect the network functionality. The algorithms designed for

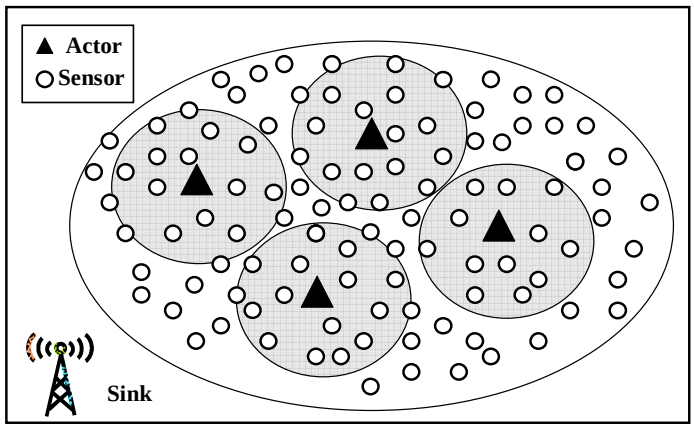


Figure 1.2: Wireless Sensor Actor Network.

WSNs need to be adaptive to the changes in the network topology. Fault-tolerance is thus an essential issue that needs to be considered while designing algorithms for WSNs.

In a Wireless Sensor Actor Network (WSAN), sensing and acting are performed by the sensor nodes and the actors, respectively. Actors are resource-rich nodes that have better processing capabilities, higher transmission range and a longer battery life. These nodes are used to perform certain local control operations on the data collected from the nodes. They may also have the ability to make local decisions or communicate with other Actors and the central monitoring centre, the Sink [6]. In order to provide effective monitoring, the nodes in WSANs - both sensors and Actors - need to be coordinated. Upon receiving information about a sensed event, Actors may need to communicate with each other in order to make local decisions and perform actions.

1.1.1 WSN Applications

Main applications of WSNs can be categorized as military, environmental, home and health. WSNs can be a part of military control, surveillance, reconnaissance systems. Battlefield surveillance can be provided by WSNs, which can provide an invaluable information for military forces' placement and strategic planning. WSNs can be deployed in critical areas of battlefields and can collect detailed intelligence about the opposing forces, before their potential destruction by the enemy forces. WSNs can be used to detect biological, nuclear and chemical attacks among other uses.

Environmental applications of WSNs include movement tracking of birds, animals and insects [24], [13], [67]. Other environmental applications are concerned with monitoring environmental changes. In the forest fire detection, sensor networks may be deployed in a forest and used for timely detecting exact location of a fire and preventing its spreading.

WSNs have great potential for Ubiquitous Computing (UC) applications (such as Smart Homes and Ubiquitous Health Monitoring Systems (UHMSs)) due to the small size and low cost of sensor nodes. WSNs are used to build smart home environment, in which sensors are embedded into appliances and furniture [84]. The resulting sensor networks are used to adapt the environment to the needs of users. Some of the health applications of WSNs include prolonged monitoring of health conditions for disease diagnostic, drug administration and tele-monitoring of human physiological data [78], [74], [79]. Application of WSNs in homes and for health monitoring is to significantly amend the quality of our life.

WSNs are provisioned to revolutionize UC applications by providing a flexible and unobtrusive means for accomplishing monitoring tasks. Even though present wireless sensors do not yet satisfy the requirements of providing invisible technology for UC applications, the future trends in development of wireless sensors are aiming at that. The "smart dust" [111] will comprise of wireless sensor nodes with the size of one cubic millimeter and will have a higher communication capabilities (using optical communication) compared to radio-communication using wireless sensors, thus becoming more suitable

for various applications.

1.2 What is Context?

Even though most people tacitly understand what context is, it is somewhat difficult to explain. The term "context-aware system" was first defined by Schilit *et al.* [97] as software that adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time. We adopted the definition of context given by Dey [34], "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves." With an understanding of what context is, application designers can determine what behaviours or features the applications should support.

Context thus represents any knowledge obtained from WSNs regarding the object being monitored. Context-awareness is an important feature of WSN applications as it provides an ultimate tool for making the applications "smart". The information about a sensed in a WSN phenomenon is comprehensive only when it includes the geographical location and the time of occurrence of the phenomenon. Thus, location and time are essential constituents of WSNs' context though the concept of context is not limited to only space and time.

In this thesis, we consider the spatio-temporal context of WSNs as it serves as a foundation for context-aware systems. Once the context about location and space is considered, then the context can be extended to include other knowledge about the object being monitored. For instance, in the case of a UHMS, the information about a patient that is being monitored (such as heartbeat, blood pressure, temperature etc.) will be collected and analyzed by the Sink(s) (such as a physician, hospital etc.). The knowledge about the patient being monitored will then be included into the context about the patient. By taking into account the context, the health monitoring system

will thus be adapted to the specific patient: observe his/her conditions and predict the evolution of events with a certain probability.

In order to build WSNs that are aware of spatio-temporal context, it is necessary to consider three areas concerning the spatio-temporal correlation of events sensed in WSNs: node localization, temporal event ordering and time synchronization. While localization's task is to provide geographic coordinates of a sensed event, preserving temporal relationships of the events in WSNs is necessary for ensuring their correct interpretation at the monitoring centre and taking proper and prompt actions. The latter can be achieved by guaranteeing time synchronization and temporal event ordering mechanisms.

In the next subsection, we present the main contributions of this thesis, while the proposed algorithms are presented in the following chapters of this thesis.

1.3 Main Contributions of the Thesis

As already stated, we consider the spatio-temporal context of WSNs as it serves as a foundation for context-aware systems. In this thesis, we concentrate on the design and development of a suite of lightweight algorithms on temporal event ordering and time synchronization as well as node localization in WSNs. We then propose an energy-efficient clustering routing protocol for WSNs that is used for message delivery in the temporal algorithm. We also propose an extension to the routing algorithm that considers multiple Sinks.

The algorithms designed in this PhD work are listed below:

- An efficient algorithm for preserving events' temporal relationships in WSANs (presented in Chapter 3) ;
- A lightweight iterative positioning algorithm for context-aware WSNs (discussed in Chapter 5);

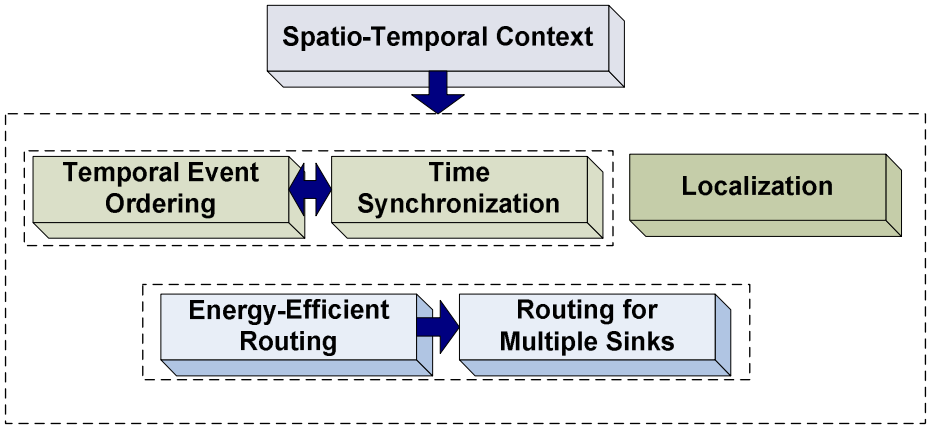


Figure 1.3: Components of Spatio-Temporal context.

- An inter-cluster communication based energy aware and fault tolerant protocol for WSNs (presented in Chapter 4);
- An energy efficient and low latency multiple events' propagation protocol for WSNs with multiple sinks (overviewed in Chapter 4).

The contributions of this thesis can be graphically represented as shown in Figure 1.3. In the diagram, spatio-temporal context consists of two modules - temporal and spatial. The former includes the algorithms on temporal event ordering and time synchronization. In this thesis, we propose an algorithm that deals with both event ordering and time synchronization as a whole. The latter module is concerned with node localization in WSNs. As an underlying routing algorithm for the temporal module, we propose a clustering routing algorithm, which is also extended to multiple Sink scenario. The following subsections briefly discuss the task of each of the modules of the diagram.

1.3.1 Temporal Relationships of Events Sensed in WSNs

Temporal event ordering has been in the focus of research on distributed systems in general. Event ordering for WSNs is a topic of active research as well. The problem of temporal event ordering is concerned with the ordering of events (messages) in the chronological order of their occurrence. The necessary condition for resolving the problem of correctly ordering the events is to ensure that the nodes in the network are synchronized, i.e. every message is timestamped and the timestamps are accurate throughout the network. However, in order to ensure the correct ordering of the events that are received by a monitoring centre (Sink/Actor), temporal event ordering algorithms are used to guarantee that there are no delayed/straggler messages still in transit in the network at the time when the messages are ordered.

1.3.2 Node Localization in WSNs

The goal of localization protocols in WSNs is to provide location information for as many sensor nodes as possible. Due to the ad hoc nature of the WSNs and the limited capabilities of sensor nodes, localization faces challenges of providing accuracy while minimizing computational cost and preserving network's scarce energy resources. A small percentage of sensor nodes may be equipped either with their global or local coordinates (theses nodes are called beacons or anchors), while the rest of the nodes are location un-aware (these nodes are called unknowns).

1.3.3 Routing in WSNs

A large number of routing protocols for WSNs has been developed recently [4]. Due to the limited energy resources of sensor nodes, designing efficient energy-aware routing protocols has been one of the most challenging issues for WSNs. Our primary focus in this thesis is on clustering routing protocols. Clustering protocols aim to reduce the network traffic toward the Sink. Moreover, cluster heads have been used to enhance the

efficiency of the routing protocols. While clustering may introduce overhead due to the cluster configuration and their maintenance, earlier work has demonstrated that cluster-based protocols exhibit better energy consumption and performance when compared to flat network topologies for large-scale WSNs.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 discusses related work for each of the research areas considered in this thesis. We first overview existing work on temporal event ordering for WSNs, followed by an overview of previous work on time synchronization and node localization in WSNs and we conclude the chapter with a discussion of related work on routing in WSNs. Chapter 3 presents our proposed algorithm for preserving temporal relationships of events sensed in WSANs, which consists of two modules: temporal event ordering and time synchronization. In Chapter 4, we discuss our proposed routing algorithm for WSNs, which is used for message delivery in the above mentioned temporal algorithm. We propose a lightweight iterative positioning algorithm for WSNs in Chapter 5. Chapter 6 presents conclusions that we drew from this thesis and discusses some directions for future work.

Chapter 2

Related Work

Before we proceed further, we wish to review related work on each of the research areas considered in this thesis. Thus, in this chapter, we begin by presenting an overview of selected algorithms on temporal event ordering, time synchronization and node localization in WSNs, by first providing the necessary background in each of the areas. We then overview a sample of energy-efficient clustering routing algorithms for WSNs.

2.1 Temporal Event Ordering in WSNs

Temporal event ordering has generally been in focus of research on distributed systems [88], [7], [85], [27]. Event ordering for WSNs is a topic of active research as well [52], [90], [19]. The problem of event ordering is concerned with the ordering of events (messages) in the chronological order of their occurrence. The necessary condition for resolving the problem of correctly ordering the events is to ensure that the nodes in the network are synchronized, i.e., every message is timestamped and the timestamps are accurate throughout the network. However, in order to guarantee the correct ordering of the events that are received by a Sink, it is also necessary to make sure that there are no messages still in transit at the time when the messages are ordered.

The causal ordering in distributed systems is based on employing a "happened before"

relation presented by Lamport [65]. Causal ordering algorithms ([88], [7], [85], [27]) ensure that if $event_1$ happened before $event_2$, then $event_1$ will be processed before $event_2$ despite the possible violation of the order of the events' reception at the receiver's site. Causal ordering has been used in distributed systems [98] and mobile systems [87] among other uses. Multicast communication protocols employed causal ordering as well [45], [59]. Causal ordering employs logical clocks; no physical clocks are used. WSNs often require the exact time at which the events occurred, and causal ordering is thus not sufficient for WSN applications.

An example depicted in Figure 2.1 considers object tracking in an emergency situation such as fire. In the example, a person is trying to find an exit from the fire site, that is being monitored by sensors s_1, s_2 and s_3. A fire commander, which receives the readings of the sensors on his Personal Digital Assistant (PDA), determines the exact location of the person caught in the fire in order to send a rescue team of firefighters. Let us assume that the person's trajectory could be traced by sensors s_1, followed by s_2 and s_3 and the messages from the corresponding sensors be delivered at the commander's site in the correct order. In that case, the rescue team will be sent to the correct location of the person that is covered by sensor s_3 (as shown in Figure 2.1-A) via the correct path. However, if these events are received at the commander out of order, let us say, a notification message from sensor s_1 is received followed by a message from s_3 (assuming that the message from sensor s_2 is still in transit due to a larger delay), then the rescue team will be sent via a wrong path (as can be seen in Figure 2.1-B). To alleviate the problem of delayed messages, rescuers could be sent towards both sensors s_2 and s_3 as shown in Figure 2.1-C. However, if the commander can apply an event ordering mechanism to identify whether there are straggler (delayed) messages still in the network, then he will be able to order the messages received from the sensors correctly and send firefighters along the correct path, thus ensuring a prompt response to the emergency situation.

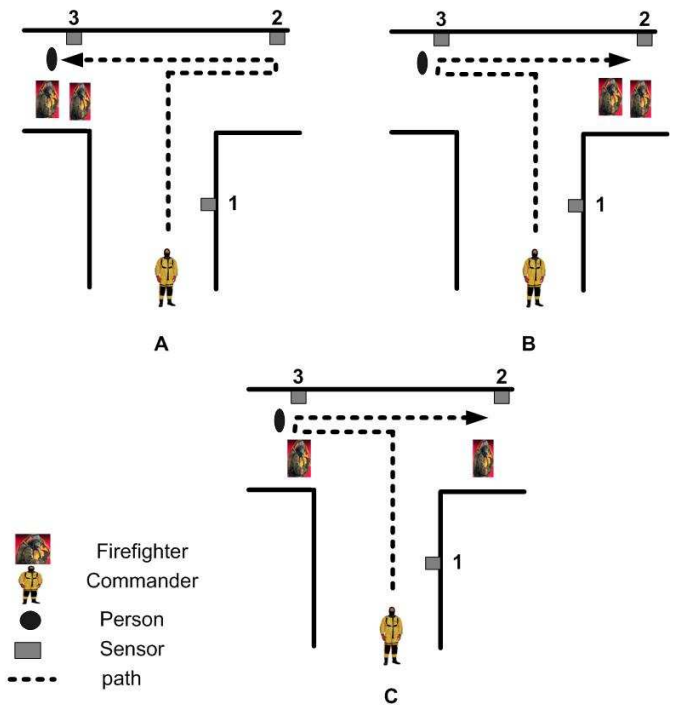


Figure 2.1: An example of possible violation of order of events (adopted from [92]).

2.1.1 Delaying Techniques

Delaying Techniques use a simple idea of postponing event ordering for a certain time, so the Sink node would wait for a selected period before handing out a message to the application to ensure that there are no messages still in transit. In *Delaying Techniques*, Mansouri-Samani and Sloman [70], Shim and Ramamoorthy [100] introduce an assumption about the upper bound of network delay D that messages can suffer. This time is equal to the maximum time it takes for a message to be sent by one of the nodes and received by another. There is a waiting time equal to that delay before the messages are ordered at a receiver's site (such as a Sink). The receiver node keeps a list of the messages received. After a time equal to D, the first message in the list is removed and handed out to the application. These techniques may not suit WSNs because the delay in these networks is highly variable and it is thus difficult to choose an optimal value for D. If a longer time interval is set before the events are ordered, there will be some

unnecessary idle time that will be wasted on waiting at the receiver. A shorter delay, on the other hand, could result in missing some messages that may still be in transit in the network (because it takes a longer time than D to arrive at the destination node).

2.1.2 Heartbeat Protocol

The *Heartbeat* protocol proposed by Hayton [52] follows an approach of sending periodic messages toward the Sink to deal with the problem of straggler messages. The protocol employs the *First In, First Out* (FIFO) property of communication channels to ensure that all previously sent messages have been received before the events are ordered. This is achieved by having every node send a message in time intervals of *delta*. These messages are the control information to ensure that there are no delayed messages in the network. A receiver node (Sink) keeps an ordered list of the received messages. After a message with a timestamp, greater than the timestamp of a received message, is received at the Sink from every node in the network, the first message in the list is removed and handed out to the application. Thus, a message m_1 that was received at the Sink node at time t_1, is removed from the list and handed out to the application, after the Sink has received from every node in the network a message m_i with the timestamp $t_i > t_1$. Due to the FIFO property, all messages prior to the message m_1, have been received by the Sink before the messages m_i.

Like in *Delaying Techniques*, it is difficult to choose an optimal value for the time interval *delta*. If a small interval is chosen then the overhead will be too large, whereas if a large interval is used, then the system will be idle for a long time. Another drawback of the scheme is that whether or not there are messages generated in the network, the control messages from every node will be regularly sent to the Sink.

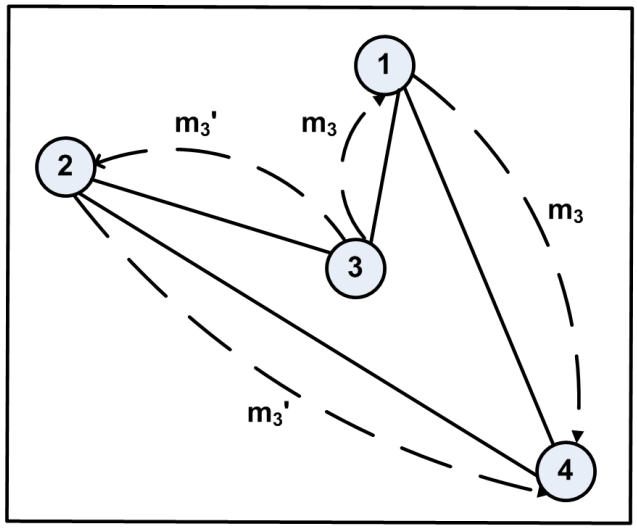


Figure 2.2: TMOS algorithm (redrawn from [90])

2.1.3 Temporal Message Ordering Scheme

Another event ordering algorithm for WSNs that is based on the FIFO property is *Temporal Message Ordering Scheme* (TMOS) [90], proposed by Römer. It constructs logical rings in the network. After the logical ring is constructed, when a node has a message to send, it sends the message in both directions of the logical ring. When both copies of the message are received by a node, it means that all previously sent messages have gone through that node due to the FIFO property of communication channels between the nodes in the network.

As shown in Figure 2.2, a logical ring including the nodes {1, 2, 3, 4}, connected with solid lines, is constructed. When node 3 senses an event e_3 at time t_3, it sends two copies of the message: m_3 and $m_3^{'}$ to its neighbouring nodes, each of the messages is sent in one direction. After receiving the message, the receiver node 1 will first send all the events that were sensed locally and then send the message m_3 to its neighbouring node 4. Similarly, node 2 will send message $m_3^{'}$ after first sending any locally sensed events to the neighbouring node 4 in the logical ring. Node 4 will insert the received events in a list ordered on the timestamps. Due to the FIFO property of communication channels,

node 4 will receive at least one copy of all the sensed events, which happened before the timestamp t_3 of the event e_3. After receiving the second copy of the message m_3, node 4 removes the first message of the ordered list and hands it out to the application.

The concept of TMOS method is interesting, but it basically doubles the network traffic: for every message, there are two copies that are sent along the logical ring. For energy constrained WSNs this will imply a lot of energy dissipation. Also, construction of logical rings is not straightforward.

2.1.4 Ordering by Confirmation

Boukerche *et al.* proposed *Ordering by Confirmation* (OBC) [19] event ordering method for Wireless Sensor Actor Networks (WSAN). OBC also employs the FIFO property of channels. The protocol can be categorized as a *flushing* event ordering protocol. In the flushing protocols, only upon receiving a FLUSH REQUEST, do producers send a FLUSH message toward the consumer. This approach does not suffer from non-determinism of delay as *Heartbeat* protocol. However, it requires to send FLUSH REQUEST to every producer and receive FLUSH messages from every producer, which implies a large message overhead. A significant improvement can be achieved if the FLUSH REQUEST is multicast to the producers and the FLUSH messages are sent back to the consumer on the same multicast tree.

OBC takes advantage of the large transmission radius of Actor node in WSAN. So, this method first constructs a hop tree around the Actor, as is required by the routing protocol used for message delivery [14]. When the Actor (depicted as a black triangle in Figure 2.3) needs to order events, it sends a temporal acknowledgment (ACK) request to the leaf nodes (the nodes at the path extremity) in the network. In this algorithm the FLUSH REQUESTs are called temporal ACK requests and the FLUSH messages are referred to as temporal ACKs. The leaf nodes are shaded in Figure 2.3. These nodes then send temporal ACK messages towards the Actor. When these messages are received from all the nodes closest to the Actor, this ensures that all earlier messages have already

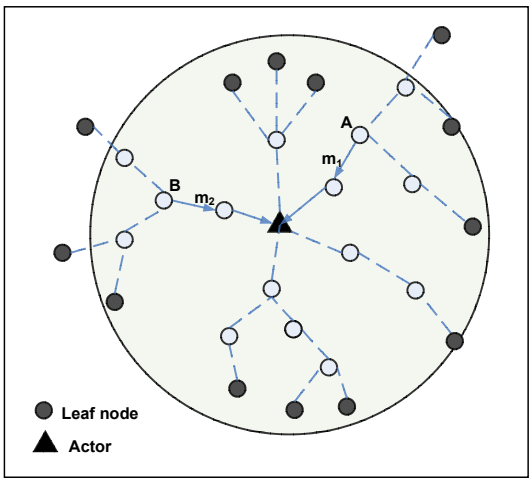


Figure 2.3: OBC algorithm's temporal acknowledgment process (redrawn from [19]).

been received. Next, the first message in the ordered list that is kept at the Actor node is removed and handed out to the application. Thus, this method does not require the nodes in the network to send messages to the Actor periodically. Only after the Actor broadcasts a temporal ACK request do the nodes in the network send temporal ACKs. Thus, the OBC does not suffer from the non-determinism of delay.

In the OBC, a node on a branching path does not forward the temporal ACK reply message before it has received the temporal ACK replies from all its children nodes. This is done in order to reduce the number of messages sent towards the Actor node. In addition, the Actor node maintains buffers for unacknowledged messages. It is possible that during the time when the Actor node waits to receive temporal ACKs regarding message m_1 (as shown in Figure 2.3) that is generated by node A, it receives message m_2 generated by node B. If nodes on the path were not yet traversed by the temporal ACKs, then, both messages m_1 and m_2, ordered according to their corresponding timestamps, will be removed from the list and handed out to the application. This way, more than one message can be temporally acknowledged with a single temporal ACK request. This results in a reduced number of messages and, consequently, conserves energy resources

of WSNs. Otherwise, if the nodes were already traversed by temporal ACKs for the message m_1, the message m_2 is placed into a different buffer. It will be handed out to the application after the Actor sends another temporal ACK request and receives acknowledgments from the nodes in the network.

2.1.5 Real-Time Message Ordering in WSNs Using the MAC Layer

The *Real-Time Message Ordering in WSNs Using the MAC Layer* (TOMAC) approach by Krohn *et al.* [64] proposed to realize temporal message ordering at the MAC layer. It is intended for a network topology where all nodes are in a single-hop distance from each other and they share the same channel. The method uses the measure of elapsed waiting time of a message as an input parameter to the channel access. Based on the waiting time, the messages are assigned a priority level r_i. The priority is initially set to zero and is increased linearly until it reaches a maximum value r_{max}. The messages are initially queued at the MAC layers of the sensors and not transmitted right away. This is done for the two following reasons. First, the access to the channel is slotted with a low-duty cycle and every message has to wait for the next possible access period. At that time, the message will compete with other messages for the channel. Second, this is done in order to avoid bursts of messages so that the correct order of messages is guaranteed. The method is however limited to one-hop distance topologies.

2.1.6 Comparison of Features of Event Ordering Algorithms

Table 2.1 presents a comparison of features of the discussed algorithms on temporal event ordering in WSNs. While *Delaying Techniques* [70], [100] and *Heartbeat* [52] algorithms suffer from the *non-determinism of delay*, the other algorithms discussed in this thesis overcome this problem. The delay in WSNs is varying and thus it is difficult to predict an upper bound of delay D in the former algorithm and the time interval *delta* in the latter

Table 2.1: Comparison of features of algorithms on temporal event ordering.

Algorithm	Non-determinism of delay	FIFO	Periodic	Flushing
Delaying techniques [70], [100]	*			
Heartbeat [52]	*	*	*	
TMOS [90]		*		
OBC [19]		*		*
TOMAC [64]		*		

algorithm. The *FIFO property* of communication channels is employed in numerous algorithms to tackle the problem of temporal event ordering in WSNs (TMOS [90], *Heartbeat* [19], TOMAC [64]). OBC [19] algorithm is a flushing algorithm, in which every node sends a flush reply to the Actor node after a flush request was received.

In the *Heartbeat* algorithm, the control messages are sent periodically to ensure that the straggler messages are delivered to the Sink before the Sink orders the events. This creates an overhead as the control messages may traverse the network even when no events were sensed in the network. In OBC, however, Actor initiates the temporal acknowledgment request only after it has received an event notification from sensor nodes. In TMOS, the process necessary for event ordering is also initiated only when a sensor node has an event notification message to forward towards the Sink: two copies of the message are sent along the logical ring. Thus, the overhead associated with the process of event ordering is reduced.

2.2 Time Synchronization in WSNs

In this section, we first present the necessary background for time synchronization in WSNs, then overview selected synchronization algorithms for WSNs, followed by a com-

parison of their features.

Time synchronization is essential for any distributed system and it is necessary in WSNs for performing data fusion, low-duty cycling, temporal event ordering and other purposes. Figure 2.4 demonstrates the components of packet delay over a wireless link, which were first introduced by Kopetz and Ochsenreiter [62] and Kopetz and Schwabl [61], and then extended by Ganeriwal *et al.* [44]. We briefly describe each of the delay components below:

- *Send Time* - the time that is spent by the sender to construct a message, i.e., the highly variable delays introduced by the operating system caused by the synchronization application (e.g., for context switches). The *Send Time* also includes the time necessary for the message to be transferred from the host to its network interface.

- *Access Time* - the time the packet waits for accessing the channel. The delay is also highly variable as it depends on the MAC protocol that is used (e.g., in TDMA channels, the sender will have to wait for its time slot to transmit).

- *Transmission Time* - the time it takes the packet to be transmitted bit by bit at the physical layer over a wireless link. This delay component is deterministic as it depends on the radio speed.

- *Propagation Time* - the time that it takes the packet to propagate from the sender to the receiver. This delay component is very small and is negligible compared to the other components.

- *Reception Time* - the time spent on receiving the packet bit by bit at the receiver node. This component of the delay is deterministic in nature like the *Transmission Time* component.

- *Receive Time* - the time it takes to construct the packet and pass it to the application layer. This delay component is highly variable like the *Send Time* as it is

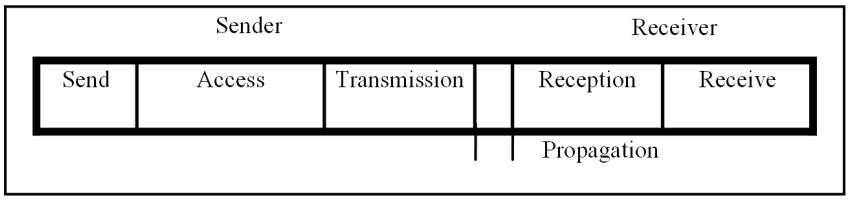


Figure 2.4: Components of packet delay over a wireless link (adopted from [44]).

affected by highly variable delays introduced by the operating system.

The resulting delay of a packet over a wireless link is thus highly affected by the delay associated with the MAC layer on the sender's side as well as the variable delays introduced by the operating system on both the sender's and receiver's sides.

2.2.1 Time Synchronization Techniques

The synchronization techniques that are used in synchronization algorithms for WSNs discussed later in this section can be divided into two broad categories. In the first category falls the traditional sender-receiver based technique, while in the second one falls the receiver-receiver based synchronization technique. Each one of the techniques is described in the following subsections.

2.2.1.1 Round-Trip Synchronization

In *Round-Trip Synchronization* (RTS) [89], the receiver of a sync pulse is synchronized with the sender. Two versions of the RTS technique are presented in [89]. In the first one, the receiver of a sync pulse replies to the sender right away. This way, the sender node can compute the round trip delay and thus synchronize the receiver node's time with its own. In the second version, the receiver of the sync pulse does not need to reply to the sync pulse right away, but only later on. It computes the delay between the reception of the sync pulse and the time when the reply is sent and attaches the delay information to the reply. The receiver node, upon receiving the reply, will subtract the delay time from

the time of the reply's reception. The receiver node will thus be synchronized with the sender.

Figure 2.5 shows the message exchange in RTS. Node N_j sends a sync pulse to node N_i, asking for the timestamp h_b^i. After node N_i sends the sync reply message back to node N_j, the latter calculates the round-trip time D^j and finds out the phase offset between its own clock and the clock of node N_i. A different version of RTS is discussed in Chapter 3.

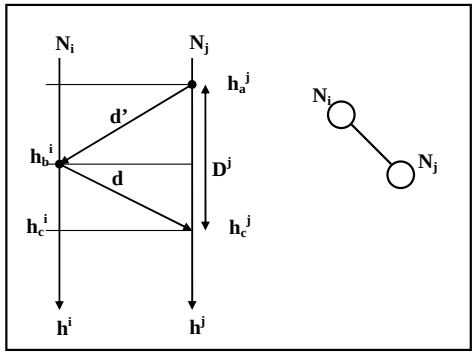


Figure 2.5: Message exchange in Round Trip Synchronization (adopted from [105]).

2.2.1.2 Reference Broadcast Synchronization

Girod and Estrin [47] proposed *Reference Broadcast Synchronization* (RBS), which, unlike traditional synchronization techniques, in which a server synchronizes a set of clients with itself, synchronizes a set of receivers with each other. A beacon node broadcasts a message to the set of receivers using the network's physical layer broadcast. The receivers use the time of the message's arrival for comparing their clocks. Due to the nature of broadcast [50], a message that is sent at the physical layer will be received by the receiver nodes at approximately the same time. Two versions of RBS are presented. In the main version, a beacon node broadcasts a message to nodes in a network. Then the receiver

nodes exchange messages with their local timestamps of the sync pulse's reception. This way, each node will have information about the other node's local time. The two receiving nodes will be synchronized with each other but not with the sender. In another version of RBS, the nodes receiving the reference broadcast send their timestamps of the sync pulse's reception back to the beacon node. This version of RBS is used when two nodes cannot communicate with each other [47], [39].

Figure 2.6 demonstrates the version of RBS in which, after two nodes receive a sync pulse from a beacon node, they send messages back to the beacon node with their timestamps at the moment of the sync pulse's reception. The nodes N_i and N_j reply to the beacon node N_k with messages containing their local times at the moment when the pulse was received (h_a^i for node N_i and $h_{a'}^j$ for node N_j). Thus, the beacon node N_k will store the information about the differences in local times of N_i and N_j.

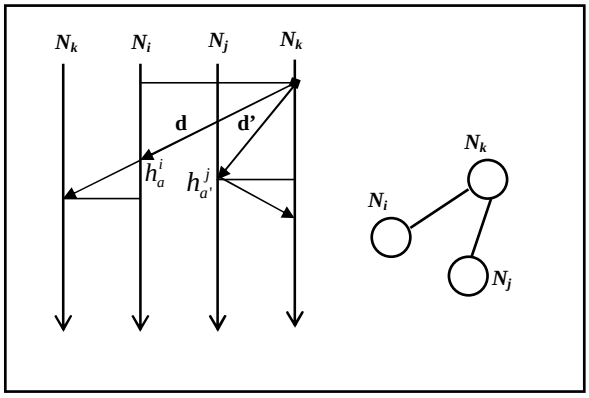


Figure 2.6: Message exchange in Reference Broadcast Synchronization.

2.2.2 Synchronization Algorithms for WSNs

Synchronization algorithms for traditional networks, such as Mills' [73] Network Time Protocol (NTP) that is used for keeping nodes in the Internet synchronized, are not well suited for WSNs. NTP uses a hierarchical scheme for time synchronization, in which a server synchronizes a number of clients. The servers in turn are synchronized by means of external sources (such as GPS). Thus, all the nodes in the Internet are synchronized to a global timescale. Elson and Römer [38] discuss the differences between traditional networks and WSNs. The main concern that makes NTP (and NTP-like synchronization schemes that are based on a global time synchronization) unsuitable for synchronization in WSNs is energy efficiency. GPS, however, is unsuitable for combining with sensor nodes due to its energy consumption and cost. Also, it requires a line of sight to GPS satellites that is not always possible in WSN applications. Among other reasons that suggest refraining from using a global timescale-based synchronization are the absence of infrastructure in WSNs as well as the presence of high dynamics.

In this section, we overview selected algorithms on time synchronization in WSNs and present a comparison of their features.

2.2.2.1 Multi-Hop RBS Time Synchronization

Girod and Estrin [47] proposed a multi-hop version if RBS synchronization technique *Multi-Hop Time Synchronization*, which relies on the use of "gateway" nodes - the nodes that are aware of the local timescales in more than one broadcast domain. In Figure 2.7, each one of the nodes inside a single broadcast domain, such as nodes 1-3, 5, 6, 10, 11, can hear broadcasts with synchronization pulses coming from only one node labeled with a letter (such as A, B, C and D, respectively). On the other hand, nodes 4, 7, 8 and 9 can hear broadcast messages coming from two nodes. For instance, the nodes 8 and 9 can hear synchronization pulses coming from nodes C and D. These nodes are then used in the algorithm to perform multi-hop synchronization. The nodes are aware of the two local time scales and are able to convert the time scales and thus synchronize the other

nodes within the corresponding broadcast domains. When a node senses an event E_i it timestamps it with its local time R_j. The notation $E_i(R_j)$ represents the time of event i according to the receiver j's clock. Thus, to compare the time of event $E_1(R_1)$ with $E_{10}(R_{10})$ the algorithm will go through the following time conversions: $E_1(R_1) \rightarrow E_1(R_4)$ $\rightarrow E_1(R_8) \rightarrow E_1(R_{10})$.

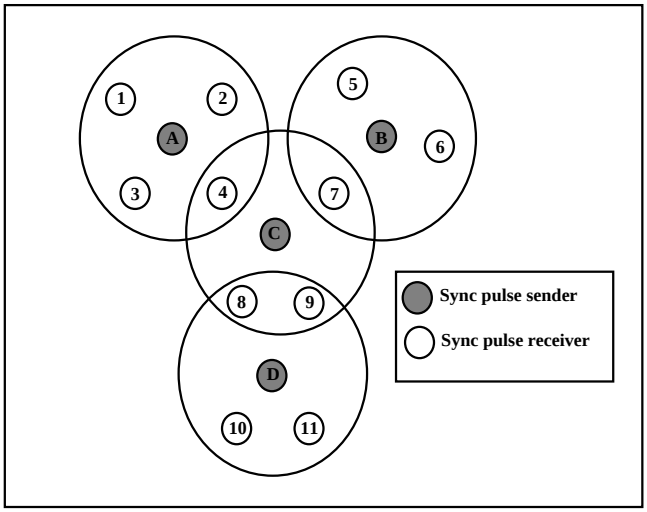


Figure 2.7: The RBS method's multi-hop time synchronization.

2.2.2.2 Timing Sync Protocol for Sensor Networks

The *Timing sync Protocol for Sensor Networks* (TPSN) proposed by Ganeriwal *et al.* [44] is a hierarchical synchronization technique. There are two phases in the protocol: *Level Discovery* and *Time Synchronization*. In the first phase, a spanning tree is built. The root node of the spanning tree then broadcasts a *level_discovery* packet. The nodes are assigned a level in the hierarchy by propagating the broadcast of the root node. After the hop tree is built, the second phase of the protocol starts. Pair-wise synchronization is performed by the nodes along the edges of the hierarchical structure, i.e., every child node is synchronized with its corresponding parent node. The RTS synchronization technique is used by the nodes to achieve synchronization. Initially, the root node broadcasts a *time_sync* packet. After receiving this packet, the nodes in the first level wait for

a random time and then send an acknowledgment to the root node. The procedure continues to the higher levels of the hierarchy of nodes. This way, every node in the network is synchronized to the root node. The important feature of the TPSN is that messages are timestamped at the MAC layer in order to reduce the delay variability by minimizing the *Send Time* component of delay as shown in Figure 2.4. When network topology changes (for instance, due to node failures), the whole procedure of master election and tree construction will be repeated.

2.2.2.3 Flooding Time Synchronization Protocol

In *Flooding Time Synchronization Protocol* (FTSP), Maroti *et al.* [71] proposed to synchronize a network to the root node as well. The node with the lowest ID is selected as a leader to be the reference point for time synchronization. The node periodically floods the network with messages containing its time. All the nodes that have not received the message, record the timestamp contained in the message as well as the message's *time of arrival*. They then broadcast the message to the neighbouring nodes after updating the timestamp. Similar to TPSN [44], FTSP timestamps synchronization messages at the MAC layer as well. After a node has gathered eight pairs of the *timestamp* and *time of arrival*, the node uses linear regression on these data to obtain phase offset and clock skew with regards to the leader node.

2.2.2.4 Gradient Time Synchronization Protocol

Sommer and Wattenhofer [103], in their proposed *Gradient Time Synchronization Protocol* (GTSP) for WSNs, aim at providing accurately synchronized clocks between neighbouring nodes. The motivation of the approach stems from the observation that the existing synchronization algorithms provide a synchronization between arbitrary nodes in the network, whereas synchronization between neighbouring close-by nodes may be poor. In GTSP, every node periodically broadcasts its time. The synchronization messages that are received by one-hop neighbours are used to calibrate (i.e., take into account

the clock drift) the logical clock. A logical clock is computed in GTSP as a function of the current hardware clock, which takes into account the relative logical clock rate and the clock offset between the hardware clock and the logical clock at a certain reference time t_0. The algorithm does not require a tree topology and a reference point. It is thus robust against node and link failures.

The authors of GTSP argue that in a distributed clock synchronization algorithm that is reliable to link and node failures it is not practical to synchronize to the clock of a reference point. Thus, in the proposed algorithm, a node tries to agree with its immediate neighbouring node on the current logical time - on a common logical rate and on the absolute value of the logical clock. According to the experimental results, GTSP is able to provide a better synchronization error than FTSP, which is a tree-based synchronization protocol.

2.2.2.5 Lightweight Time Synchronization

Lightweight Time Synchronization (LTS) is a synchronization approach, proposed by Van Greunen and Rabaey [108], that provides a specific precision. Two versions of the time synchronization algorithm WSNs are proposed. In the first version, an on-demand time synchronization is achieved, while in the second one a proactive approach to node synchronization is proposed, when all the nodes are synchronized proactively. In both versions of the algorithm, there are one or more master nodes that are synchronized to an external reference time. In the proactive version, the algorithm starts with constructing a spanning tree with the master node at the root of the tree. Then, the nodes synchronize to their parent in the tree by means of RTS synchronization technique. The synchronization frequency is based on the requested precision, the depth of the spanning tree and the drift bound p_{max}.

In the reactive version of the algorithm, when a node needs to be synchronized, it sends a request to one of the master nodes. Along the reverse path of the request message, nodes synchronize using RTS measurements. The synchronization frequency

is calculated the same way as in the first version of the algorithm. With the goal of reducing synchronization overhead, each node may ask its neighbouring nodes for pending synchronization requests. If there are such requests, then the node synchronizes with its neighbouring node and not the reference node.

2.2.2.6 Time Diffusion Synchronization

Su and Akyildiz [107] proposed *Time Diffusion Synchronization* (TDS) protocol for WSNs that synchronizes the whole network. The algorithm starts with electing master nodes/leaders. If external synchronization is required, these nodes must have an access to a global time. Otherwise, the masters are not synchronized a priori. A master node then broadcasts a request message that contains its current time, to which all the nodes send a reply message. Using RTS measurements, the master node then calculates and broadcasts the average message delay and standard deviation. The nodes store the data for all master nodes. Then they become so-called "diffused leaders" and repeat the procedure. The nodes on the path from masters sum up the average delays and standard deviations. The diffusion process terminates at a certain number of hops from the master nodes. At this stage of the algorithm, every node has received from one or more master nodes m the time at the initial leader h_m, the accumulated message delay δ_m and the accumulated standard deviation β_m. A clock estimate is then computed as $\sum_m w_m(h_m + \delta_m)$, where the weights w_m are inversely proportional to the standard deviation β_m. At the time when all the nodes update their clocks, new master nodes are elected and the procedure is repeated until all nodes' clocks settle on a common time.

2.2.2.7 Comparison of Features of Synchronization Algorithms

As seen above, the delay of a packet over a wireless link is highly affected by the *Send Time*, *Access Time* and *Receive Time* (shown in Figure 2.4). The traditional synchronization algorithms based on *sender-receiver* mode of time synchronization, such as RTS [89], in which a receiver node is synchronized with a sender node in WSNs, suffer from the

uncertainty of these components of delay. The *receiver-receiver* based synchronization technique such as RBS [47], on the other hand, minimizes the uncertainty of the delay by eliminating the *Send Time* and *Access Time* components from the total packet delay. This happens due to the fact that these components of delay are the same for all the receivers of the packet.

Another approach to the task of reducing the components of delay are followed by the synchronization protocols TPSN [44], [71] that proposed to timestamp messages at the MAC layer at both the sender's and receiver's sides. This way, delay variability is decreased, and consequently, the message delay uncertainty is decreased.

Multi-Hop Time Synchronization [47] is thus able to decrease the synchronization error. [47] also discusses the possibility of further reducing the total delay of a packet, if it is timestamped at the low level in the receiver node host's operating system kernel. Therefore, the *Receive Time* will not include the overhead of system calls, context switches etc.

LTS [108], TPSN [44], GTSP [103] and FTSP [71] follow a hierarchical approach of node synchronization, where nodes are synchronized to a root node of a constructed tree. GTSP [103], on the other hand, does not require a tree topology: neighbouring nodes' clocks are synchronized with each other and not with a reference node's clock. A comparison of features of the algorithms is presented in Table 2.2.

2.3 Node Localization in WSNs

In this section, we discuss the problem of node localization in WSNs, by first reviewing the estimation of distances and angles and then the techniques for obtaining node positions. We then discuss selected algorithms for node localization in WSNs and present a summary of their comparison.

Table 2.2: Comparison of features of algorithms on time synchronization.

Algorithm	Sender-Receiver	Receiver-Receiver	Hierarchical	MAC time-stamping
Multi-Hop RBS [47]		*		
TPSN [44]	*		*	*
FTSP [71]	*		*	*
GTSP [103]	*			*
LTS [108]	*		*	
TDS [107]	*		*	

2.3.1 The Task of Localization Algorithms for WSNs

The goal of localization protocols in WSNs is to provide location information (geographic coordinates) for as many sensor nodes as possible in WSNs. In order for a sensor node to be able to locate itself in a two/three dimensional space it is necessary for the node to know positions of three/four nodes as well as the distances to these nodes (the ranging techniques for obtaining distances are overviewed in subsection 2.3.2). In a WSN, a small percentage of sensor nodes may be equipped with either global or local coordinates (these nodes are called beacons or anchors), while the rest of the nodes are location unaware (these nodes are called unknowns). After a sensor node obtains the required information regarding the beacons, it is able to calculate its own geographic coordinates by performing trilateration. In the case when the angles to the beacons are known instead of the distances, the sensor node will perform triangulation instead in order to obtain its position. It is not always possible for a node to receive the information about the beacon nodes in one hop. In these cases, a multi-hop propagation of the beacon information is used for the sensor nodes to learn of the positions of beacons and the corresponding distances/angles to them.

Due to the ad hoc nature of the WSNs and the limited capabilities of sensor nodes, localization faces the challenges of providing accuracy while minimizing computational

cost and preserving network's scarce energy resources. In the following subsection we first briefly overview the ranging techniques for distance/angle estimation. We then overview trilateration and multilateration techniques for obtaining sensor nodes' geographic coordinates.

2.3.2 Estimation of Distances and Angles

- *Received Signal Strength Indication* (RSSI) [17] is stemming from the fact that radio signal strength is inversely proportional to the distance squared. Thus, if a receiving sensor node measures the received signal strength of a sending node, it should be able to obtain the distance between the sending node and itself. RSSI is practical in the sense of low cost as no additional equipment is required in order to measure the distance between two nodes. Every sensor node is equipped with a radio module and thus is able to calculate the distance. However, RSSI is inaccurate as the measurements are noisy on the order of several meters [10]. The inaccuracy of RSSI can be explained by the fact that radio signal propagation is not universal for different materials.

- *Time of Arrival* (ToA) [17] estimates the distance between the sender and receiver sensor nodes by using the time that it takes for the signal to propagate from one node to another. The radio signal propagates with the speed of light and if the time that it took the signal to propagate from the sender to the receiver is known, then the receiver node can compute the distance separating the two nodes. Obviously, the nodes need to be synchronized in order for the computation to be accurate.

- *Angle of Arrival* (AoA) [17] is obtained by means of directional antennas or using a set of receiver nodes. By using the time of arrival of the signal at the receivers, it is possible to calculate the angle of arrival at the sensor node in question. This technique has an accuracy of a few degrees. However, it requires additional hardware and thus adds to the cost and size of a sensor node.

- *Time Difference of Arrival* (TDoA) [17] uses two different kinds of signals to compute the distance between the sending and receiving nodes. Radio signal can be sent at the same time (or with a delay) with an acoustic signal [112], for instance. The difference in the speed of the two signals allows the receiver node to calculate the distance that the signals have traveled by multiplying the difference in the speeds by the difference in the time of arrival of the two signals. This ranging technique has a very high precision, on the order of centimeters. The main disadvantage of the technique is that it requires extra hardware, which increases the cost and size of a sensor node.

- *Hop Count* cannot be categorized as a ranging technique; however the metric is used by many localization algorithms for WSNs in order to obtain distance measurements to the beacon nodes. The DV-hop based algorithm [77] is the pioneering approach in this category of algorithms. The main idea is based on the fact that if messages sent by two nodes are able to reach each other, then the distance between the nodes is bounded by the transmission range of their radios. The hop count between two nodes is defined as the number of hops on the shortest path connecting the two nodes. The advantage of the hop count-based approaches is their simplicity and low cost as they do not require any additional hardware and simply use the connectivity information to obtain distance estimates between the nodes. However, the inaccuracy of the approach is high due to the accumulated error that results from the averaging of the hop distance between two neighbouring nodes. The inaccuracies may be high in the cases when two nodes are separated by an obstacle.

2.3.3 Trilateration

In trilateration, an unknown node requires the information about three beacon nodes as well as the distances to these beacon nodes in order to obtain its position estimate. Once

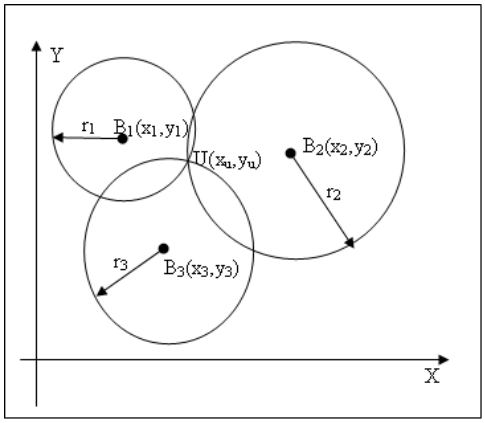


Figure 2.8: Trilateration to beacons B_1, B_2 and B_3 performed by node U.

a node has received the information about three beacons and the average hop size, it performs trilateration and obtains its position estimate. In order to be able to perform trilateration, the unknown node U depicted in Figure 2.8 needs to know the coordinates of the beacon nodes B_1, B_2 and B_3 as well as the distances to these beacon nodes (r_1, r_2, r_3). Using the theorem of Pythagoras, the information about the beacons' coordinates and the distances to the beacon nodes from the unknown node U can be expressed in the equations (2.1 - 2.3) below.

$$(x_1 - x_u)^2 + (y_1 - y_u)^2 = r_1^2 \tag{2.1}$$

$$(x_2 - x_u)^2 + (y_2 - y_u)^2 = r_2^2 \tag{2.2}$$

$$(x_3 - x_u)^2 + (y_3 - y_u)^2 = r_3^2 \tag{2.3}$$

Subtracting the equation 2.3 from equations 2.1 and 2.2 will give:

$$(x_1 - x_u)^2 - (x_3 - x_u)^2 + (y_1 - y_u)^2 - (y_3 - y_u)^2 = r_1^2 - r_3^2$$

$$(x_2 - x_u)^2 - (x_3 - x_u)^2 + (y_2 - y_u)^2 - (y_3 - y_u)^2 = r_2^2 - r_3^2$$

After rearranging the equations, we get a system of two linear equations with two

unknowns as shown below. The coordinates of the unknown node U(x_u, y_u) can be obtained by solving the following system of equations.

$$2(x_3 - x_1)x_u + 2(y_3 - y_1)y_u = (r_1^2 - r_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2)$$
$$2(x_3 - x_2)x_u + 2(y_3 - y_2)y_u = (r_2^2 - r_3^2) - (x_2^2 - x_3^2) - (y_2^2 - y_3^2)$$

When the information about angles to the beacon nodes is used instead of the distances, *triangulation* is performed by an unknown node by means of using the laws of sines and cosines to compute its coordinates.

2.3.4 Multilateration

In the case of multilateration, the information about more than three beacon nodes as well the distances to the beacon nodes is used by an unknown node in order to obtain its position estimate. Multilateration may yield a more accurate position then trilateration as the information to more beacon nodes is taken into account by an unknown node to calculate its coordinates. The multilateration equations are shown in the equations 2.4 - 2.6, when there are n beacon nodes and the distance errors are considered. Similar to the case of trilateration, the system of linear equations can be rewritten in a matrix form Ax $\approx$ b by subtracting the last equation as shown in the equation 2.7. The system of equations is over-determined as there are more equations than unknowns. It can be solved by using the least squares method [94], [48], [58]. Then, x = $(A^T A)^{-1}(A^T b)$, where A^T is a transpose of the matrix A. The least squares method minimizes the sum of the squares of the differences between estimated and computed distances.

$$(x_1 - x_u)^2 + (y_1 - y_u)^2 = r_1^2 - e \tag{2.4}$$

$$(x_2 - x_u)^2 + (y_2 - y_u)^2 = r_2^2 - e \tag{2.5}$$

$$\vdots$$

$$(x_3 - x_u)^2 + (y_3 - y_u)^2 = r_3^2 - e \tag{2.6}$$

$$2 \begin{bmatrix} x_n - x_1 & y_n - y_1 \\ ... & ... \\ x_n - x_{n-1} & y_n - y_{n-1} \end{bmatrix} \begin{bmatrix} x_u \\ y_u \end{bmatrix} \approx \begin{bmatrix} (r_1^2 - r_n^2) - (x_1^2 - x_n^2) - (y_1^2 - y_n^2) \\ ... \\ (r_{n-1}^2 - r_n^2) - (x_{n-1}^2 - x_n^2) - (y_{n-1}^2 - y_n^2) \end{bmatrix} \tag{2.7}$$

Another method for calculating unknown node's coordinates is *bounding box* [95], [17], in which squares around the beacon nodes are used (instead of circles as in lateration) to represent the boundaries of each beacon node. The unknown node is then located at the intersection of the squares.

2.3.5 Localization Algorithms for WSNs

The problem of node localization in WSNs has received an increased attention in the research community. There have been proposed a number of approaches to node localization in static WSNs [8], [94], [99], [106], [25], to name just a few. Other approaches such as [101], [113], [86], [104], [63], [18] considered mobile node-assisted localization. While some approaches such as [36], [99] consider a centralized solution to node localization problem, others follow a distributed approach [77], [93], [96], [95].

The *Global Positioning System* (GPS) [82] is the most known and used location system. In GPS, the ranges to at least four from the existing twenty four satellites are used for trilateration in order to find the coordinates of the receiver that needs to be localized. However, as already noted above, GPS is not suitable to be used in WSNs for

a number of reasons. From a point of view of the low price and small size of a sensor node, it is not reasonable to equip every sensor node with a fairly expensive and large in size GPS unit. Next, GPS's energy consumption is high and thus its use is prohibitive in energy-constrained WSNs. Also, GPS cannot be employed indoors as it requires a line-of-sight.

2.3.5.1 Semidefinite Programming Approach

One of the centralized localization algorithms is *Semidefinite Programming* (SDP) presented by Doherty *et al.* [36]. In this approach, geometric constraints of the node's topology are presented as linear matrix inequalities. After all the constraints present in the network are expressed as linear matrix inequalities, the inequalities can be combined into a single semidefinite program. Solving the semidefinite program yields a bounding region around each node in the network.

The limitation of this approach stems from the fact that not all geometric constraints can be expressed as linear matrix inequalities but only constraints that form convex regions can be represented in that way. To solve the linear semidefinite program all the linear matrix inequalities must be combined at the central node and it is a very time consuming task. The advantage of SDP is its elegance: from a set of convex constraints, SDP simply finds the intersection of the constraints. However, the algorithm faces challenges in terms of scalability.

2.3.5.2 Multidimensional Scaling MAP

Another centralized algorithm for node localization in WSNs is *Multidimensional Scaling MAP* (MDS-MAP) proposed by Shang *et al.* [99]. Instead of using semidefinite programming, MDS-MAP uses multidimensional scaling (MDS), a technique from mathematical psychology. MDS is based on the idea that if there are n nodes, whose positions are not known but the distances between the nodes are known, it is possible by using the Law of cosines and linear algebra to reconstruct the relative locations of the nodes, only using

the pair-wise distances.

MDS-MAP algorithm goes through four stages. In the first stage, the ranging data is collected from the network and a matrix of distances between the nodes is built. An element of the matrix d_{ij} corresponds to the distance between the nodes i and j. The matrix can also contain zeros for the cases when ranging information was not collected from the network. In the second stage, the algorithm runs a standard all pairs shortest path algorithm on the matrix to find inter-node distances. In the third stage, MDS is applied on the obtained distances to find position estimates of the nodes. In the final stage of the algorithm, the estimated node positions are transformed into global coordinates system using some number of fixed beacon nodes. The performance of MDS-MAP improves as the ranging improves.

2.3.5.3 Ad Hoc Positioning System

One of the pioneering approaches to node localization for WSNs, proposed by Niculescu and Nath [77], is the *Ad Hoc Positioning System* (APS). It consists of three different versions as described below. The APS is similar to distance vector routing, in which every node communicates only with its immediate neighbours. Each message exchanged in the APS contains the node's available estimates to landmarks (beacons). The method uses multi-hop propagation from landmarks to sensor nodes. Once a node receives three range estimates to three or more landmarks, it is able to compute its own position by using trilateration.

In the *DV-hop* propagation method, there are three non-overlapping stages. In the first stage, each node propagates distances in hops to each landmark in the WSN. In the second phase, a landmark estimates an average size of a hop after it receives a message from another landmark. It then floods the network with the average size of a hop in meters as a correction to distance measurements. In the third phase, an arbitrary node, after receiving the correction, estimates distances to a landmark in meters and uses this information to perform trilateration. Another version of the algorithm is *DV-distance*, in

which, as opposed to *DV-hop*, the distances between neighbouring nodes are measured using radio signal strength RSSI and are propagated in meters rather than hops. *Eucledian* propagation method propagates true *Eucledian* distances to landmarks. The advantage of *DV-hop* method is its simplicity. The fact that no range measurements are used by an arbitrary node to perform trilateration frees the approach from the inaccuracies inherent to range measurement techniques (such as RSSI, AoA, ToA and TDoA discussed above). However, the approach is not scalable mainly due to the high communication cost.

2.3.5.4 Robust Positioning Algorithms for WSNs

Savarese *et al.* [93] proposed another DV-hop based algorithm *Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks* (RPA). The algorithm goes through two phases, namely Hop-TERRAIN and Refinement. In the Hop-TERRAIN phase, the algorithm finds the number of hops from a node to each beacon node in the network. The nodes then multiply the hop count by a shared metric of an average hop distance to estimate the distance to each of the anchor (beacon) nodes. Triangulation is performed by using the least squares algorithm. In the Refinement phase, given the position estimates of the Hop-TERRAIN, the goal is to obtain more accurate positions by means of using ranges between neighbouring nodes. Refinement goes through iterations, in which each node broadcasts its position estimate and waits for replies with position estimates from its one-hop neighbours.

By using one of the above-mentioned ranging techniques, the node also obtains distance measurements to its neighbouring nodes. It then computes a least squares lateration to obtain its new position. As a result, in the majority of cases, after a number of iterations, the node's position will change towards its true position. To mitigate error propagation, the Refinement algorithm assigns a confidence level to each node's position. The confidence levels are used to weigh the linear equations when solving the system of linear equations by means of the least squares trilateration. The anchors have high confidence levels, whereas a node that observes poor conditions (e.g., few neighbours, poor

constellation) associates a low confidence level with its position estimate. Consequently, the nodes with poor confidence levels about their position estimates have less impact on the outcome of the triangulations performed by the neighbouring nodes.

2.3.5.5 Ad-Hoc Localization System

The *Ad-Hoc Localization System* (AHLoS) proposed by Savvides *et al.* [96] goes through a two-phase process - ranging and estimation - to dynamically discover nodes' locations. During the ranging phase, each node estimates its distances from neighbouring nodes. In the estimation phase, nodes use the ranging information and the known beacon locations to estimate their positions. Once a node has a position estimate, it becomes a beacon and assists other nodes in estimating their positions in an iterative manner. RSSI and ToA are employed to obtain ranging information in AHLoS.

Two versions of the algorithm are considered, such as centralized and distributed. In the centralized version of the algorithm, location estimation is performed at a central, more powerful node. After that, the results are forwarded to the nodes. In the algorithm's distributed version, the location estimation is performed at the nodes, without involving a central node. The algorithm uses multilateration when a node has ranging information to more than three beacons. Atomic multilateration, iterative multilateration and collaborative multilateration algorithms are used in the AHLoS. In the atomic multilateration, if a node has the ranging information to more than three beacon nodes, it performs multilateration. In the case of iterative multilateration, such a node then becomes a beacon after it estimates its position. It then assists the neighbouring nodes in obtaining position estimates. The collaborative multilateration is considered when two neighbouring nodes do not have ranging information to the required number of beacons, but may assist each other (collaborate) in obtaining the needed information. The iterative multilateration can be applied in small-scale networks.

2.3.5.6 The n-Hop Multilateration Primitive

Savvides *et al.* [95] proposed *n-Hop Multilateration* approach. As its title suggests, the algorithm considers a multilateration that spans over multiple hops. The algorithm goes through three main phases and a post-processing phase. In the first phase, the nodes self-organize into groups, collaborative subtrees, so that the nodes that do not have position estimates are over-constrained and can have only one possible solution.

If a node does not satisfy the required constraints, it does not become part of a collaborative subtree. During the second phase, the nodes use geometric relationships (a method of constructing a bounding box [95], [17]) between measured distances and beacon locations to obtain initial position estimates. In the third phase - refinement - iterative least squares approach is used to obtain final position estimates. The position estimates are further refined in the post-processing phase of the algorithm similarly to the process used in the second phase. The n-hop collaborative multilateration approach to node localization is scalable to larger networks as opposed to multilateration approach described above [96].

2.3.5.7 Comparison of Features of the Localization Algorithms

The localization algorithms presented in this chapter were of two categories: centralized and distributed. In the centralized category fall SDP [36] and MDS-MAP [99] algorithms, while the remaining algorithms discussed in this chapter belong to the distributed category.

There exists a trade-off between the overhead of localization algorithms and precision in node localization that they provide. The algorithms using only the connectivity information (hop-based algorithms) have a lower overhead but the precision is compromised. On the other hand, the algorithms using ranging techniques, may require additional hardware, which increases the overhead (the cost and size). However, this also increases the precision of node localization.

The *DV-hop* algorithm [77] and SDP [36] use only the connectivity information to

Table 2.3: Comparison of features of the discussed localization algorithms.

Algorithm	Connectivity information	Ranging techniques	Iterative	Distributed
SDP [36]	*			
MDS-MAP [99]	*	*		
DV-hop [77]	*			*
RPA [93]	*	*	*	*
AHLoS [96]		*		*
n-Hop Multilateration [95]	*	*	*	*

localize nodes. DV-distance [77] and AHLoS [96], on the other hand, use ranging techniques such as RSSI and ToA to obtain distances between the nodes. RPA [93], n-Hop Multilateration [95], on the other hand, use both: the connectivity information for the initial position estimation and the ranging techniques for refining the position estimates iteratively.

Multilateration is used in the approaches RPA and AHLoS to improve upon position estimates, while the method *n-Hop Multilateration* [95] considers multilateration that spans over multiple hops. The discussed features are summarized in Table 2.3 below.

2.4 Energy-Aware Clustering Routing Protocols for WSNs

In this section, we overview a sample of clustering routing algorithms for WSNs and present a comparison of their features.

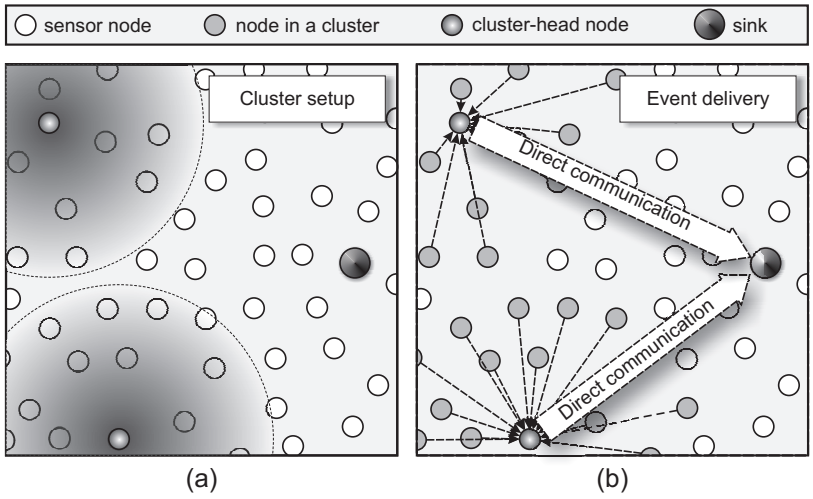


Figure 2.9: The LEACH protocol (redrawn from [83]).

2.4.1 Low-Energy Adaptive Clustering Hierarchy Protocol

The *Low-Energy Adaptive Clustering Hierarchy protocol* (LEACH) by Heinzelman *et al.* [53] is a well-known hierarchical routing protocol and it was one of the pioneering clustering approaches in the literature on WSNs. LEACH is a cluster-based protocol that utilizes the randomized rotation of cluster heads to evenly distribute the energy load among the sensors in the network. In LEACH protocol, the sensor nodes organize themselves into local clusters, with one node acting as the cluster head (CH). The LEACH mechanism includes the randomized rotation of the CH function among the sensor nodes so as not to drain the battery of a single node. It also performs local data aggregation in the CHs to reduce the amount of data being sent from the clusters to the Sink, further reducing energy dissipation and enhancing the network lifetime.

The sensor nodes elect themselves to be CHs at any given time, with a certain probability. These CH nodes broadcast their status to the other sensors in the network. Each sensor node determines which cluster it wants to belong to by choosing the cluster head that requires the minimum communication energy as shown in Figure 2.9 (a). Once all the nodes are organized into clusters, each CH maintains a schedule for the nodes in its cluster. This allows the radio components of each non-cluster-head node to be turned off

at all times except during its transmission time, thus minimizing the energy dissipated in the individual sensors.

According to the simulation results presented by Heinzelman *et al.* [53], the sensor nodes reach energy depletion randomly and the dynamic clustering increases the network lifetime. The LEACH protocol is completely distributed and requires no global knowledge of a network. However, it uses one-hop communication from sensor nodes to and from a CH to the Sink, as exemplified in Figure 2.9 (b). The LEACH mechanism therefore faces scalability problems if applied to large networks, due to the long distance communication necessary to relay the sensed data.

2.4.2 Geographic Adaptive Fidelity Protocol

Xu *et al.* [114] proposed *Geographic Adaptive Fidelity* (GAF) location-based protocol, which was designed primarily for mobile ad hoc networks and could be used in WSNs as well. GAF does not depend on the underlying routing algorithm used for message delivery. GAF is an example of adaptive fidelity, a technique proposed for extending the lifetime of self-configuring systems due to using redundancy. The purpose of exploiting redundancy is to conserve energy while application fidelity is maintained. The idea of the protocol is to conserve energy by using the sleeping mode for some of the nodes without affecting the connectivity of the network. The method forms a virtual grid for the nodes in the network and decides which nodes in the same grid (geographical area) are equivalent in terms of their ability to communicate with the nodes in the neighboring grid. Each node uses its location information to associate itself with a virtual grid, while all nodes in a particular grid square are equivalent with respect to forwarding packets. GAF can achieve significant energy savings by using the sleeping mode for those nodes without affecting routing fidelity - especially when the network size grows. Although GAF is a location-based protocol, it resembles hierarchical protocols in which the clusters are associated with the geographical location. In GAF, each node in the network uses GPS to indicate their position. However, GPS is an energy-consuming and costly tool to be

combined with an energy-scarce sensor node, as already discussed above.

2.4.3 Threshold Sensitive Energy Efficient Sensor Network Protocol

Threshold sensitive *Energy Efficient sensor Network protocol* (TEEN), proposed by Manjeshwar and Agrawal [68], is a hybrid of hierarchical clustering and data-centric protocols designed for time-critical applications. This protocol is responsive to sudden changes in some of the attributes observed in WSNs (e.g., temperature). The algorithm first goes through cluster formation. The CHs then broadcast two thresholds to the nodes in their clusters. Those are hard and soft thresholds for the sensed attributes. Hard threshold is the minimum possible value of the attribute, which triggers the node to turn the radio on and transmit to the CH. Using the threshold results in reducing the number of transmissions and thus saving energy. After the attribute's value reaches the hard threshold, the node will transmit again only when the attribute's value changes by the soft threshold. By adjusting hard and soft thresholds, it is possible to save energy by decreasing the number of transmissions. However, TEEN cannot be applied for sensor networks where periodic sensor readings should be delivered to the Sink, since the values of the attributes may not reach the threshold at all. Another limitation of the protocol is that the message propagation is accomplished by CHs only. If CHs are not in each other's transmission radius, the messages will be lost.

Adaptive threshold sensitive Energy Efficient sensor Network protocol (APTEEN) [69] is an extension of the TEEN protocol. It can be used for both periodic and responsive data collection. The disadvantage of the two approaches is overhead and complexity of cluster formation. The Sink is engaged in forming the clusters, which obviously creates a lot of network traffic. Another drawback of the protocol is difficulty maintaining the threshold-based functions.

2.4.4 Multipath Routing Protocol for WSNs

Yang *et al.* [115] proposed *Multipath Routing Protocol* (MRP) for WSNs, which is a clustering multipath routing protocol for WSNs that employs ant colony optimization techniques [23], [49] for selecting multiple paths leading from the source nodes towards the Sink. The algorithm goes through three phases: cluster formation, multipath construction and data transmission. The initial phase of the algorithm - cluster formation - happens when an event is sensed in the network. The algorithm follows an approach of dynamic clustering: the nodes nearby the sensed phenomenon join a cluster. In the second phase of the algorithm, ant colony optimization is used to construct multiple paths with optimal and suboptimal energy consumption. The multiple paths are used for balancing the load in the network. A cluster head dynamically chooses a route to forward data through, based on energy consumption. In the final phase of the algorithm, the data is propagated toward the Sink via the multiple paths. At this stage, maintenance of the routes is performed. Depending on the residual energy of cluster head nodes and availability of the multiple routes, a new route discovery process involving ant colony optimization is initiated by the nodes. Performance evaluation of MRP reported by Yang *et al.* [115] shows that the algorithm is able to achieve a fair load balancing and to extend network lifetime.

2.4.5 Cluster-based Periodic, Event- and Query-based Protocol for WSNs

Boukerche *et al.* [14] presented *Cluster-based Periodic, Event-driven and Query-based Protocol* (CPEQ) for WSNs. CPEQ is a cluster-based routing protocol in which nodes with more residual energy are selected as cluster heads (CH). It was designed to provide a uniform energy consumption among the sensor nodes and to reduce network traffic. In CPEQ, a cluster head (CH) node forms a cluster and the nodes within the cluster forward data to the CH. Thereafter, the CH performs data aggregation on the gathered

data, e.g., average or filtering, and relays the results to a base station (Sink). Every node in the network can become a CH for a specific period of time. When the time expires, other nodes are selected as CHs based on the remaining energy of the nodes. The network is initially configured using the PEQ [14] protocol, which floods the network and assigns hop levels to the nodes based on their hop distance to the Sink. Thereafter, every node will have the chance to become a selector node with a certain probability. A selector node requests the energy level of its neighbor nodes and selects the one with more energy remaining as a CH, as depicted in Figure 2.10. After forming its cluster, a CH delivers data to a Sink through a multi-hop path, as shown in Figure 2.11. CPEQ shows better performance than the PEQ, and reduces the network traffic.

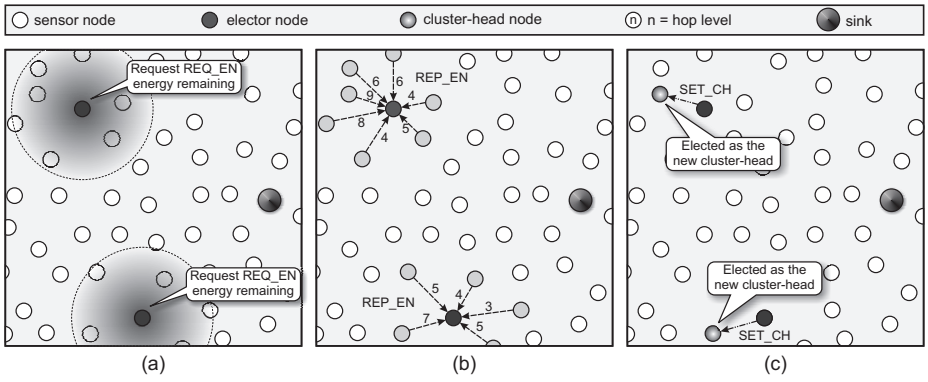


Figure 2.10: Cluster head selection in CPEQ (adopted from [83]).

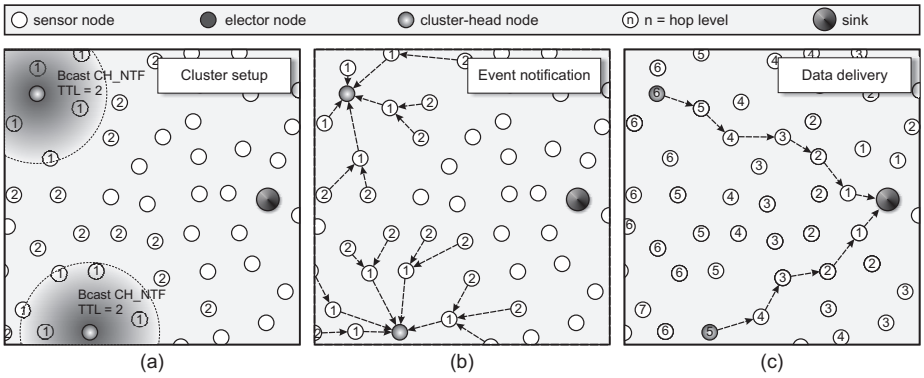


Figure 2.11: Data delivery in CPEQ (adopted from [83]).

Table 2.4: Comparison of features of selected clustering routing protocols for WSNs.

	Periodic, Event-Query-based	Single hop	Sink engaged in cluster formation	Data aggregation	Only CHs are relay nodes
LEACH [53]	*	*		*	*
TEEN [68]			*	*	*
APTEEN [69]	*		*	*	*
CPEQ [14]	*			*	
GAF [114]	*			*	
MRP [115]	*			*	

2.4.6 Comparison of Features of Routing Protocols

This section presents a discussion and comparison of features of the energy-aware clustering routing protocols. Table 2.4 lists the features of the clustering protocols for WSNs considered. As discussed earlier, LEACH [53] uses a single-hop routing when the cluster heads send event notifications to Sinks. All remaining protocols listed in the table use multi-hop routing for message propagation. This addresses the problem of scalability. In the majority of clustering routing protocols, only the cluster head nodes relay messages to the Sink, either in a single hop or in multiple hops, when the message is propagated from a cluster head to another cluster head until it eventually reaches the Sink. In CPEQ, GAF and MRP on the other hand, it is possible to use cluster heads, cluster nodes and the free nodes (nodes that do not belong to clusters) for propagating a message toward the Sink. This way, CHs are not involved in long transmissions. CH nodes already have a heavy load of configuring clusters and maintaining a schedule for the nodes' low-duty cycling. If the CHs are also engaged in long-distance transmissions, they will deplete their batteries sooner.

As already mentioned, the TEEN [68] cannot be used for periodic WSNs. The other protocols included in the table can be used in all modes of reporting the event notifications to the Sink, such as periodic, event-driven and query-based modes. In the TEEN and APTEEN [69] routing protocols, the cluster formation process involves the Sink, which implies a significant amount of network traffic. The other protocols included in the table

go through the cluster formation in a distributed manner, without engaging the Sink in the process.

Routing protocols achieve energy efficiency by employing low-duty cycling when the radio module of a sensor node is turned off. In LEACH, a CH node employs a TDMA schedule for the nodes in a cluster. GAF constructs a virtual grid in order to find equivalent nodes which become representative nodes of a given grid in order to communicate to nodes in the neighbouring grid. Those nodes are active in turns. Data aggregation is another way of reducing the number of event notification messages that need to be sent toward the Sink. The protocols presented in the comparison table accomplish data aggregation.

2.5 Summary

In this chapter, we presented the background and an overview of the algorithms, concerning the spatio-temporal context in WSNs in three areas: temporal event ordering, time synchronization and node localization. We also overviewed a selection of routing algorithms for WSNs. In this section, we briefly summarize the features of the discussed algorithms in each of the categories.

First, while some of the temporal event ordering algorithms for WSNs overviewed in this chapter suffered from the non-determinism of delay, other algorithms have overcome this problem. In the more recent algorithms, in order to reduce communication cost, the necessary control messages (that ensure there are no straggler messages in the network before the Sink orders events) are sent by the nodes after a request from the Sink is received and not periodically, as it was done in the earlier algorithms.

Second, the overviewed in this chapter synchronization algorithms for WSNs aimed at reducing various components of delay. While the sender-receiver based synchronization technique suffers from the uncertainty of delay, the receiver-receiver based synchronization technique achieves a smaller synchronization error by eliminating some of the

components of delay noted above. Timestamping synchronization pulses and replies at the MAC layer also aimed at reducing the impact of the components of delay and thus reducing the uncertainty of message delay.

Third, the localization algorithms for WSNs, discussed in this chapter, achieve a higher precision when not only connectivity information is used for localization purposes but also the ranging techniques. However, the overhead is thus increased as well. For DV-hop based algorithms, reducing the communication cost is critical in order to resolve the scalability issue. The iterative algorithms attempt to increase the precision of node localization, by employing a refinement stage, after the initial position estimates are obtained by the nodes.

Last, from the comparison of features of the clustering routing protocols discussed in the thesis, we can observe that an early clustering routing protocol LEACH could not be applied in WSNs that spanned large areas because of direct communication of cluster heads with the Sink or other cluster heads. The routing protocols that were developed later addressed the scalability issue. Multi-hop data propagation is employed by either using only cluster heads as relay nodes or by using the cluster heads, cluster nodes and the free nodes. Having the nodes other than the cluster heads participate in the message delivery has the objective of reducing the load on the cluster head nodes in order to preserve their energy. This, in turn, aims at achieving longer periods of network operation without cluster re-configuration and prolonging network lifetime.

Chapter 3

Preserving Temporal Relationships of Events

In this chapter, we present our algorithm on preserving temporal relationships of events sensed in Wireless Sensor Actor Networks (WSANs). The algorithm consists of two modules, which deal with the problems of temporal event ordering and time synchronization in WSANs. We propose to approach these two problems as a whole as they complement each other: in order to ensure that the events are temporally ordered, it is necessary to have a synchronized network. We here propose modifications to the Ordering by Confirmation [19] event ordering protocol for WSANs by introducing a clustering concept into the network's topology with the goal of reducing the overhead of event ordering. At the same time, we propose a hybrid synchronization scheme that is suited for the clustered topology. Moreover, the proposed algorithm utilizes the message exchange necessary in the event ordering algorithm and the underlying routing algorithm for time synchronization purposes by piggybacking them with synchronization pulses and replies in order to reduce the traffic needed for time synchronization.

As stated, the proposed algorithm on preserving temporal relationships of events sensed in WSANs consists of two modules - temporal event ordering and time synchronization. Even though both parts of the algorithm are executed concurrently, for clarity,

we choose to describe them in separate subsections. The clustering routing protocol discussed in Chapter 4 is used as an underlying routing protocol for message delivery while performing temporal event ordering and time synchronization.

3.0.1 Temporal Event Ordering

In this section we present our proposed algorithm on preserving temporal relationships of event sensed in WSANs. The initial objective of OBC event ordering algorithm presented by Boukerche *et al.* [19] (overviewed in Chapter 2) was to reduce the overhead of event ordering by reducing the number of messages that needed to be sent by the nodes as compared with periodic algorithms such as the *Heartbeat* algorithm [52]. In the latter algorithm, the nodes send heartbeat messages toward the Actor node periodically to ensure that if there were delayed messages in transit in the network, they get delivered to the Actor before the heartbeat messages do, due to the FIFO property of communication channels. The main modification to OBC method that we propose is using clustering concept in the network with the goal of further reducing the overhead of temporal ACK process of OBC. We will refer to the proposed algorithm as *Clustered Ordering by Confirmation* (COBC) hereafter.

As discussed in Chapter 2, OBC is a flushing algorithm, in which a consumer (Actor) first sends a flush request to the producer nodes (the nodes in the WSAN), to which the nodes reply with a flush message. In such algorithms, every node will have to send one flush message. Due to the FIFO property of communication channels this will ensure that the straggler messages will get delivered to the Actor before the flush messages do. Thus, event ordering is achieved in flushing algorithms with a lower overhead as compared with the periodic ones. In the proposed COBC algorithm, similarly, the nodes will have to send a flush message toward the Actor after hearing the flush request.

The rationale behind the idea of introducing a clustering concept into WSAN is the following. Clustering can further reduce the overhead of event ordering in terms of delay and energy dissipation. In OBC, a node on a branching path forwards a temporal ACK

message once it has received the messages from the nodes on the branches. To ensure that every node sends only a single flush message, every parent node in the constructed tree will have to wait to receive the flush messages from its children nodes before forwarding the flush message further toward the Actor. In a clustered topology, however, only a cluster head (CH) node will have to wait to receive the flush messages from its cluster nodes before sending the message further toward the Actor. This will decrease the delay of temporal event ordering process. It will also result in a reduced energy dissipation.

Compared with periodic event ordering algorithms (such as *Heartbeat*), clustering can reduce the number of messages that need to be sent by the nodes and thus decrease the overhead of event ordering and preserve energy at the same time. If there are m nodes in a cluster, the cluster head (CH) node will wait to receive temporal ACK messages from all the nodes and will then forward one message on behalf of all these nodes. In the periodic event ordering approach, if these nodes did not belong to a cluster, however, all m nodes would have to send a separate message toward the Actor.

In COBC, communication channels use FIFO as in the original version of OBC method. The setup phase of the algorithm consists of the cluster formation and path discovery that are accomplished by means of the clustering routing protocol described in Chapter 4. In COBC, the Actor node is viewed as a Sink node in the routing algorithm used. The event ordering procedure is initiated by the Actor by sending a temporal acknowledgment request and is followed by the reception of temporal acknowledgments (ACKs) from the nodes in the network. At this time, the first message in the list of received event notification messages is handed out to the application.

Setup. At the beginning of setup phase, a hop tree is built to find out the number of hops each node is away from the Actor. The Actor floods the area of the network with a message that is used to build the hop tree. The process iterates as long as the receiver nodes have heard the broadcast by the Actor: once the hop count message arrives at a node, the node checks whether it has previously received the broadcast by the Actor, i.e., whether it is in the transmission range of the Actor. If the answer is yes, then the

hop count process iterates; otherwise, the receiver node replies to the sender to inform it that it is beyond the transmission radius of the Actor. If a node gets replies from its neighboring nodes that they are out of the range of the Actor, then the sender node learns that it is a node with the maximum hop count. The process of learning the hop count terminates here. It is necessary to note that in the setup of the routing algorithm, in the implementation of the temporal event ordering and synchronization algorithm, every free node is associated with a cluster/CH node; there are no free nodes that are not associated with a cluster.

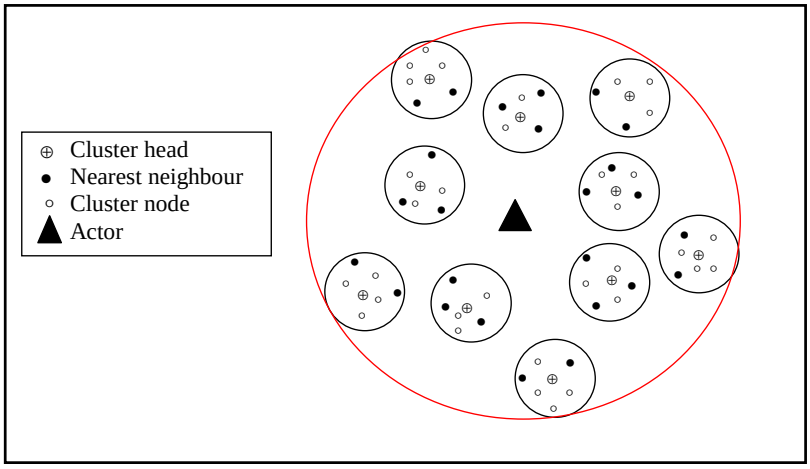


Figure 3.1: A WSAN with configured clusters.

After hearing a temporal acknowledgment request message from Actor, the nodes with the maximum hop count will initiate the process of sending the temporal ACK messages because they are located at the extremities of the geographic area of the Actor's monitoring. The cluster formation phase will result in constructing the clusters and finding the nodes that are responsible for inter-cluster communication as part of the routing algorithm. The nodes with the maximum hop count, if they belong to a cluster,

will indicate that the corresponding cluster is in the highest level of the hierarchy of clusters. Figure 3.1 demonstrates a WSAN with constructed clusters, after the setup phase of the routing protocol is complete.

Temporal Acknowledgment. This phase of the algorithm starts with Actor's broadcast of temporal acknowledgment request (temp_ACK_req) message. Once the message is heard by the free nodes with the highest hop count and the clusters that have cluster nodes with the highest hop count, they will initiate the process of sending temporal acknowledgment (temp_ACK) messages toward the Actor. After the temp_ACK request broadcast is heard, all free nodes will send temp_ACK messages toward the clusters with which they are associated. Thus, if a cluster belongs to the highest level of the hierarchy (i.e. it is at the extremity of the geographic area monitored by the Actor), then it will initiate the process of sending the temp_ACK message toward the Actor by means of nearest neighbor nodes and free nodes according to the routing algorithm, after first receiving the temp_ACK messages from associated free nodes (if any).

Once a CH receives these messages from all its associated free nodes and cluster nodes, the CH will forward the temporal ACK message to the nearest neighbor nodes to be forwarded toward the neighboring clusters. Upon receiving the message, a cluster node (the nearest neighbor node in the destination cluster) will send the message to its corresponding CH. The process will continue until the Actor is reached. The Actor waits for the temp_ACK messages from the nodes in all of the first-level clusters and/or the free nodes with the minimum hop count and then orders the events.

Figure 3.2 (a) demonstrates the initial phase of the temporal acknowledgment process. The clusters in the highest level of the hierarchy initiate the sending of the temp_ACK messages. In the figure, black circles represent nearest neighbour nodes, while the blank circles represent the remaining cluster nodes. Figure 3.2 (b) demonstrates the final phase of the temporal acknowledgment process when the clusters in the first level of the hierarchy forward the temp_ACK messages to the Actor. The propagation of a temp_ACK message will be accomplished by the routing protocol in the same way as an

Algorithm 1 PROPAGATION OF A TEMPORAL ACKNOWLEDGMENT TOWARD ACTOR.

{Upon receiving a temporal ACK request broadcast by Actor.}
1: **while** temporal ACK message is received **do**
2: **if** node's status is a cluster head **then**
3: **if** there is a node with a maximum hop count **then**
Action: Wait for temporal ACK from cluster nodes.
4: **end if**
5: **if** there are associated free nodes **then**
Action: Wait for temporal ACK from the associated free nodes.
6: **end if**
{Upon receiving the temporal ACK message(s) from the node(s)}
Action: Broadcast temporal ACK message for the nearest neighbour nodes to be forwarded toward the neighbouring cluster(s) (except from which received).
7: **else if** node's status is a free node **then**
Action: Send temporal ACK toward the cluster head node with which associated.
8: **else**
{This case corresponds to the status of a cluster node}
9: **if** node's status is a nearest neighbour node **then**
10: **if** the cluster has node(s) with the maximum hop count **then**
Action: Wait for temporal ACK message from cluster head.
{Upon receiving temporal ACK message from the cluster head}
Action: Send temporal ACK toward the neighbouring clusters.
11: **else**
Action: Wait for temporal ACK message from neighbouring clusters.
{Upon receiving temporal ACK message from neighbouring clusters}
Action: Send the temporal ACK to the cluster head node.
12: **end if**
13: **else**
Action: Send the temporal ACK to cluster head node.
14: **end if**
15: **end if**
16: **end while**

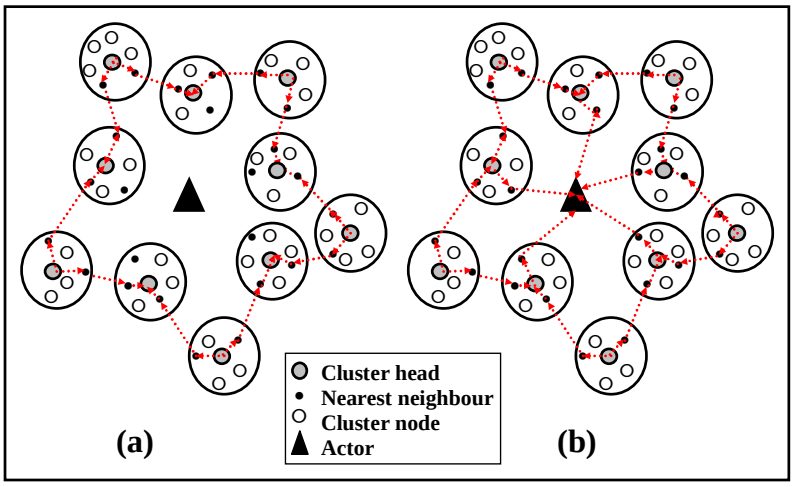


Figure 3.2: (a) Initiation of temporal ACK process. (b) Temporal ACKs reach Actor.

event notification's. However, when sending an event notification, a CH selects one of the possible nearest neighbors through which to forward the event notification.

Algorithm 1 shows a high-level pseudocode of temporal ACK message's propagation toward the Actor.

3.0.2 Time Synchronization

In the time synchronization module of the proposed algorithm, we propose to use two synchronization techniques. The first technique is RBS, which is employed for synchronizing CHs with each other. The second technique is RTS, which is used for synchronizing cluster nodes and associated free nodes with their corresponding CH.

The essential idea of the proposed synchronization scheme is not to use a global time scale as it is not well-suited for WSNs (as discussed in Chapter 2) but we propose to use local time scales at every CH. The Actor node keeps a table of the local times of

the CHs. The synchronization is achieved by time conversions of local times of different CHs at the Actor. We refer to the proposed synchronization scheme as Synchronization for Clustered Topology (SCT) hereafter. We use the version of RBS in which, after two nodes receive a sync pulse from a beacon node, they send messages back to the beacon node with their timestamps at the moment of the sync pulse's reception. The beacon node will store the information about the differences in local times between the two nodes.

Thus, the beacon node will store the information about the differences in local times of CHs. In our method, we use the Actor node as a beacon node that will broadcast the sync pulse to the CHs. The CHs in turn will reply to the Actor node with messages about their timestamps at the moment when the sync pulse was received.

In the proposed algorithm, after the CHs hear the broadcast by the Actor, they do not reply to the sync pulse received from the Actor right away. They will attach this information to an event notification message when an event is sensed by a node in the cluster. Actually, the Actor will only need this information when an event that originated inside that particular cluster needs to be ordered.

Every event notification coming from the network will have propagated through a CH node as described below. As part of the routing protocol, if an event is sensed by a node belonging to a cluster, then the event notification message will first be sent to the corresponding CH node and timestamped, with the latter taking into account the local time of the CH node and the delay computed according to the RTS synchronization. If an event is sensed by a free node, then the event notification message will first go through the CH node that has learned about the existence of that free node during the setup phase of the routing algorithm (the CH, with which the free node is associated). Again, the message will be timestamped accordingly: the node will be synchronized with the CH node by means of RTS synchronization.

Once the event notifications that originated inside clusters or in free nodes reach the Actor, the Actor will convert the local times of the CH nodes in order to synchronize the

CHs each other. At this point, the event ordering algorithm will be started (as described in the previous subsection) in order to identify whether there are delayed messages still in transit in the network.

Now we will describe how the information about the local times at the CHs is collected at the Actor. After the clusters are constructed, the Actor node broadcasts a message (sync pulse) at time a, to which all the CHs reply with their timestamps at the time when they received the pulse. At this time, the Actor constructs an initial table of local times at the CHs (see Table 3.1). The Actor, upon receiving an event notification, will

Table 3.1: Table with local times at the CHs kept at the Actor node.

Time	CH_1	CH_2	...	CH_n
a	h_{a1}	h_{a2}		h_{an}
b	h_{b1}	h_{b2}		h_{bn}
...				

also receive the time at which the sync pulse was received by the CH. The Actor will use this information for time conversions of the times at which the other CHs received the sync pulse. The sync pulses have a unique number so that the Actor can keep track of the information regarding the reception time of each sync pulse at the CHs.

Using this information, the Actor will then construct a table containing the phase offsets (time differences) of the CHs. Thus, the Actor will have complete information about the phase offsets between any two CHs. This information will be used for event ordering by ensuring that the timestamps of all events reflect the phase offsets and are accurate with respect to each other. The information in this table will later be incremented when more messages are exchanged between the Actor and the CHs. These messages will be exchanged as part of the event ordering algorithm described in the previous subsection. To keep the network synchronized, as we already noted, synchronization pulses are piggybacked onto the underlying event ordering protocol. For example, when the Actor broadcasts a temporal ACK request message to the CHs for the purpose of performing

event ordering, the CHs will reply with temporal ACKs together with their local times at the moment when they received the message broadcast by the Actor. The information about the CHs' local times at time b will thus be received at the Actor node. Then the table will be extended with the new data (see Table 3.1).

At this point, the information for the calculation of phase offsets between any two CHs will include the number of reference packets sent (i.e. the number of broadcasts by the Actor). The more messages are sent, the more accurate the phase offset calculation will be. Linear regression is used for calculating the phase offset between two CHs, taking into account the number of reference broadcasts. The phase differences between two clocks change over time due to the clocks' drift (the frequency of the clocks is not exactly the same). As Girod *et al.* [47] discussed in their work, linear regression allows the synchronization technique to take into account the clock skew of the receivers (in our case, the CHs).

Table 3.2 shows the time offsets of the CH nodes. The proposed method takes advantage of the Actor node's higher power and more powerful CPU as compared to the rest of the sensor nodes. The advantage of using this version of RBS for our algorithm is that the time conversion tables are kept at the Actor node. Also, all the time conversion operations will be performed at the Actor node. Consequently, because the Actor has a global picture of the local times at all the CHs, it can accurately convert the time for any two CHs and thus synchronize events that were sensed in the corresponding clusters.

Table 3.2: Table with phase offsets of the CHs

	CH_1	CH_2	...	CH_n
CH_1	–			
CH_2		–		
			–	
CH_n				–

Depending on the number of CHs that will be synchronized by means of the RBS

synchronization technique, a different number of beacon messages from the Actor will be required. Larger numbers of nodes will require a larger number of reference broadcasts. In the implementation of SCT, all message exchange between the CHs and the Actor is piggybacked with synchronization pulses and replies (for example, the message exchange necessary in the setup of the routing protocol). This approach will result in a reduced number of messages required for synchronization (including synchronization of CHs with each other). Besides, it is necessary to keep the network synchronized, i.e. the network needs to be resynchronized over time because of the clock skew of the nodes that need to be synchronized.

Algorithm 2 TIME SYNCHRONIZATION PERFORMED BY CLUSTER HEADS AND ACTOR.

1: **if** node's status is a cluster head **then**
{Upon hearing a temporal ACK request from Actor and receiving messages from cluster nodes and free nodes.}
2: **if** the message type is temporal ACK **then**
Action: Store the broadcast ID and the local time of the broadcast.
{Upon receiving the temporal ACK message(s) from cluster nodes and associated free node(s) (if any)}
Action: Piggyback the time of the Actor broadcast's reception onto the temporal ACK message to be forwarded toward the Actor via the neighbouring cluster(s) (except from which received).
3: **else**
{the message is an event notification (used in the underlying routing algorithm) or message used in the setup phase of routing and event ordering algorithms}
Action: Calculate round trip time based on the delay calculated from the time of the message's arrival and that of a previously sent message from the CH node to the free/cluster node.
Action: Store the delay of messages for the corresponding node.
4: **end if**
5: **end if**
6: **if** node's status is an Actor **then**
Action: Store the temporal ACK broadcast's ID and the time of the broadcast.
{Upon receiving temporal ACKs}
Action: Extract the information about cluster heads' local times of the broadcast's reception.
Action: Create/increment the Table of Phase offsets of cluster heads.
{Upon receiving temporal ACKs from all one-hop neighbours}
Action: Convert timestamps (taking into account the phase offsets of cluster heads) and order the events.
7: **end if**

As we mentioned, within a cluster, the nodes are synchronized with RTS synchronization technique. At the time when clusters are being configured, the CH and the

nodes exchange messages (such as the declaration of the CH and merge requests coming from the nodes). These messages are used for synchronization purposes as well. When a node declares itself a CH, it broadcasts a message to the surrounding nodes. The CH stores the time when the message was sent. The nodes that decide to join the cluster send merge requests that are piggybacked with their timestamps at the time when the CH declaration message was received. The CH then computes the Round Trip Time for each node and calculates the delay. A high-level pseudocode of time synchronization performed by cluster head nodes and the Actor node, each on their level, is described in Algorithm 2.

When a sensor node senses an event, the event-notification message is sent from the source node to its corresponding CH. It is then timestamped with the CH's local time, taking into account the delay between the node and the CH. Thus, synchronization on demand or event-driven synchronization is used for the sensor nodes. There is no need for permanent synchronization of the nodes in clusters that are not CHs. This scenario indicates that RTS synchronization is a reasonable choice for piggybacking the sync pulses and replies to the message exchange necessary in the underlying routing protocol. Thus, the synchronization scheme proposed in this thesis has a multi-modal structure: cluster nodes and free nodes are synchronized with their CH node using RTS synchronization, whereas, on the level of CHs, the RBS synchronization technique is used to synchronize the CHs with each other. The combination of synchronization techniques seems applicable and reasonable and is in accordance with the idea of tunable synchronization (as discussed by Elson [39] and Elson and Römer [38]).

Figure 3.3 shows the message exchange in the version of RTS that is used in the algorithm for synchronizing cluster nodes and free nodes with their corresponding CH node. Node N_j sends a sync pulse to node N_i, asking for the timestamp h_b^i. Node N_i does not, however, send the sync reply message back to node N_j right away but after some time D^i.

Node N_j calculates the Round Trip Time D^j that corresponds to the time elapsed

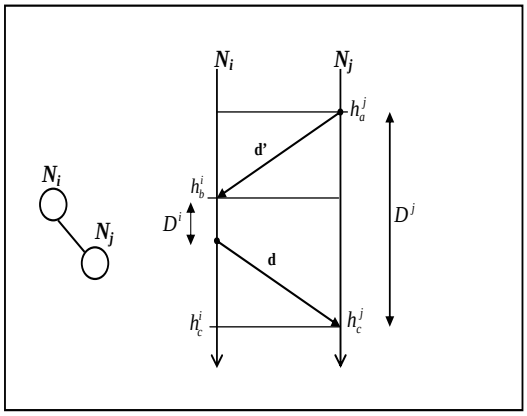


Figure 3.3: A version of Round Trip Synchronization.

between sending the sync pulse and receiving the sync reply as shown in equation 3.1, taking into account the time D^i that node N_i waited before sending the sync reply back to node N_j.

$$D^j = h_c^j - h_a^j - D^i \tag{3.1}$$

Then, node N_j is able to approximate its local time h_b^j that corresponds to the time when the sync pulse was received by node N_i (time h_b^i) as shown in equation 3.2.

$$h_b^j \approx h_c^j - D^j/2 \tag{3.2}$$

This way, node N_j learns of the time offset between the clocks of node N_i and its own. Due to using this version of RTS in our algorithm, the cluster nodes do not reply to CH node specifically with the sync reply after they hear a message from CH, but do it at a later time when they have a message to send to the CH node, by piggybacking the message with the synchronization reply. In the same manner, the free nodes are

synchronized with the CH node with which they are associated. For free nodes, a multi-hop time synchronization is used as they do not hear a broadcast of CH node directly because they do not belong to a cluster.

The method described above is sufficient when only relative synchronization is required (for instance, for timing the propagation delay of sound [46]), i.e. it is important to know which event occurred before or after a given event. However, some applications may require absolute time synchronization. For those applications, when relative times are not sufficient, we use the RBS version that includes synchronization with an external reference. As discussed by Girod and Estrin [47], for the case of synchronization with an external timescale, such as Coordinated Universal Time (UTC), when GPS sends a Pulse-Per-Second (PPS) at the beginning of every second, the nodes in its broadcast domain treat the PPS as a reference broadcast. They timestamp the PPS message with the local time at which the broadcast was received and compare the local time with the true GPS time. The nodes are then able to recover the phase offset and skew of the receiver nodes' clocks relative to the GPS node's. Illustrative example described in the next subsection will discuss both cases: relative and absolute time synchronization, as performed by our algorithm.

3.0.3 An Illustrative Example

In this subsection, we present an example that will illustrate how time synchronization is achieved in our algorithm. In the example, we consider n cluster head (CH) nodes. Let us assume that at the time a when the Actor node broadcasts its message, the local times recorded at the CH nodes are as shown in Table 3.3.

Table 3.3: Table with local times.

Time	CH_1	CH_2	...	CH_n
a	10 : 03	10 : 05		10 : 02

The table of time offsets that the Actor node creates will then look like it is shown

in Table 3.4. Obviously, the elements of the table corresponding to a relationship of the times at two CHs are the same, they only differ by their sign. For instance, the CH_1's clock is behind the clock of CH_2 by two minutes, which is shown in the table in two elements: CH_1CH_2 and CH_2CH_1 with opposite signs. We assume that the data in the table reflects the results of linear regression on multiple phase offsets.

Table 3.4: Table with phase offsets.

	CH_1	CH_2	...	CH_n
CH_1	–	-2		1
CH_2	2	–		3
...			–	
CH_n	-1	-3		–

Having the information above, let us assume that the Actor has received two events (E_1 and E_2) with their corresponding timestamps (as were timestamped at the corresponding CHs): E_1, timestamped at the CH_1 at the local time: 12:02 and E_2, timestamped at the CH_2 at the local time: 12:01. Thus, the CH nodes sensed different events at different times and the local times at which the events were sensed are presented in the table.

From these two events, we could conclude that E_2 happened before event E_1. However, after converting the local times at the Actor node, it is revealed that E_1 happened before E_2 as shown in Table 3.5. When an event E_3 timestamped by CH_n with its local time at 12:03 is received at the Actor, the latter can synchronize CH_n with either CH: CH_1 or CH_2. The element in the table of phase offsets corresponding to CH_nCH_1 is -1, i.e., the local time of CH_n is behind the local time of CH_1 by one minute. After conversion, the timestamp of event E_3 will be 12:02 when comparing with CH_1. Similarly, when comparing with CH_2, the timestamp of event E_3 will be modified to 12:00, as shown in Table 3.5.

An example of synchronization with an external timescale is presented in Table 3.6.

Table 3.5: Table with local times and converted times.

CH	Local times when events were sensed	Time after conversion
CH_1	12:02	12:00 (compared with CH_2)
CH_2	12:01	12:01
CH_n	12:03	12:02 (when comp. with CH_1) 12:00 (when comp. with CH_2)

In this case, the time offsets will be converted with regards to the Actor's time (supposing that the latter is equipped with a GPS). The Actor node stores a table of phase offsets for reception of PPS broadcasts. It is then able to convert the local times in accordance with the GPS true time.

Table 3.6: Table with local times and phase offsets.

Node	Local time	Phase offset
Actor	12:00	
CH_1	12:02	+2
CH_2	11:59	-1
CH_n	12:03	+3

Let us suppose that the Actor has received three events: E_1 timestamped at the CH_1 at the local time 13:01, E_2, timestamped at the CH_2 with the local time 13:01 and E_3 timestamped with the local time at the CH_n 13:00. Then, after the conversion, the times will be as follows: E_1 12:59; E_2 13:02 and E_3 12:57. This way, the receivers of the RBS sync pulses are synchronized to an external time scale that in our case is kept at the Actor node. It is important to note that all the time conversion operations are performed at the Actor node, which frees the rest of the energy-constrained nodes of the additional computations and storage.

3.1 Performance Evaluation

In this section, we present the results of simulation experiments that we conducted for each of the two modules of the proposed algorithm. We first discuss the results of the temporal event ordering module, followed by the simulation results of time synchronization module. We then present the results of combining the two modules and show the gain of the proposed methog of piggybacking synchronization pulses and replies onto the messages required in the underlying temporal event ordering and routing algorithms.

3.1.1 Experimental Results of Temporal Event Ordering Module

We used ns-2 network simulator [72] in order to evaluate performance of the proposed algorithm. In the experiments, the number of nodes ranged from 25 to 100, more precisely, we experimented with 25-, 50-, 75- and 100-node networks. In each case, we varied the percentage of the producer nodes (source nodes, that sense an event) from 12% to 100%. The initial energy of a sensor node was set to 2J. The simulation time was set to 1000s. The reception (Rx) power was set to 0.395W, transmission (Tx) power was set to 0.66W and the idle power was set to 0.035W [57].

We experimented with a grid topology, in which the producers (source nodes) were placed in the left bottom corner of the simulation area and the Actor was placed at the right top corner of the area. We intended to create a realistic scenario and test performance of the algorithm in this setting. We used transmission range of a sensor node of 25m. The simulation areas were set to $6400m^2$, $14400m^2$, $22400m^2$ and $30400m^2$ for 25-, 50-, 75- and 100-node networks, respectively. For each of the experimental groups we ran the experiments 33 times (based on the number of nodes and percentage of producer nodes) with a confidence interval of 95% (shown on the graphs).

Two main metrics that we used to evaluate performance of the algorithm were the delay and energy dissipation. We considered

- *delay of the temporal acknowledgment process* - the time elapsed from the moment when Actor broadcasts a temporal ACK request and the time when the temporal ACKs are received at the Actor node and the *delay per single temporally acknowledged event notification*;

- *energy dissipation of the temporal acknowledgment process* and the *energy dissipation per single temporally acknowledged event notification.*

These two metrics - delay and energy dissipation - constitute the overhead of any event ordering mechanism. This is the price that the algorithm pays in order to ensure that the messages (event notifications) are ordered at the Actor node according to the chronological order of their occurrence despite the possible violation of the order of their receipt at the Actor. COBC's proposed modifications to OBC aimed at reducing this overhead of event ordering process.

When more than one messages are temporally acknowledged during a single acknowledgment step, the algorithm decreases the overhead of event ordering. The two metrics measured per single event notification such as the *delay per single temporally acknowledged event notification* and the *energy dissipation per single temporally acknowledged event notification* demonstrate the delay and energy dissipation for the cases when more than one node (i.e., a specific percentage of producer nodes) generated event notifications and they were acknowledged during a single temporal acknowledgment step.

We compared performance of COBC with OBC and Heartbeat event ordering algorithms. The selection of these algorithms to compare COBC with stemmed from the initial objective of OBC (and later COBC) on improving upon the periodic event ordering algorithms that are based on sending periodic messages towards the Sink/Actor in order to ensure correct interpretation of events, such as Heartbeat algorithm. Both OBC and COBC are initiated on demand, i.e., when a temporal acknowledgment is sent by Actor. In the following subsections, we present the results of a comparative study of these two metrics - delay and energy dissipation of event ordering process - for the

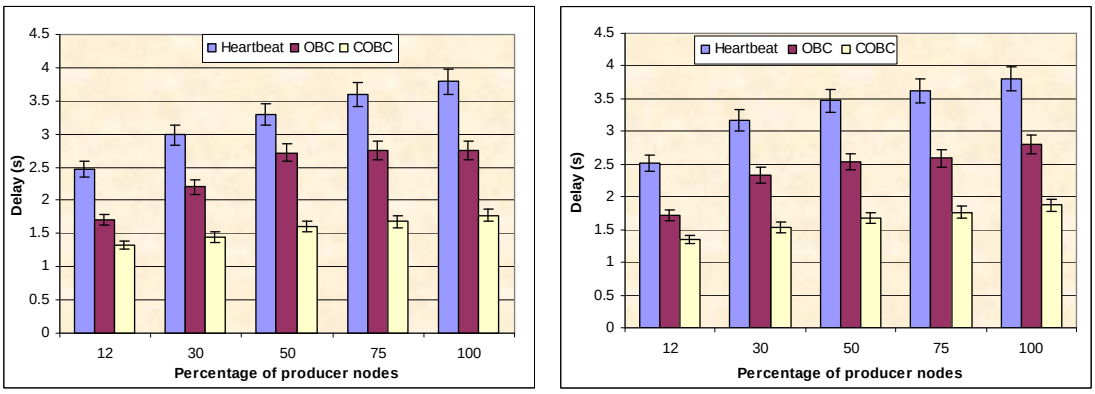


Figure 3.4: Delay of temporal ACK process in (a) 25-and (b) 50-node networks.

above-mentioned algorithms.

It is necessary to note that for Heartbeat, the delay of event ordering was calculated from the moment when an event notification was received at the Actor and the time when the Actor has received heartbeat messages (that are sent by the nodes periodically) from all the nodes in the network with a timestamp greater than the timestamp of the event notification message in question. Energy dissipation is similarly calculated for the same period. Message delivery in Heartbeat was accomplished by means of a shortest path algorithm. Initially, a hop tree was built and every node in the network stored the minimum hop neighbour on the path toward the Actor. It would send a message back toward the Actor via a route with the minimum hops.

Figures 3.4 and 3.5 show the graphs of delay in COBC, OBC and Heartbeat algorithms for different percentages of producer nodes when the number of nodes varies. As can be seen from the graphs, the delay is increasing for all the algorithms as the percentage of producer nodes increases. COBC outperforms both Heartbeat and OBC for all numbers of nodes and percentages of producer nodes.

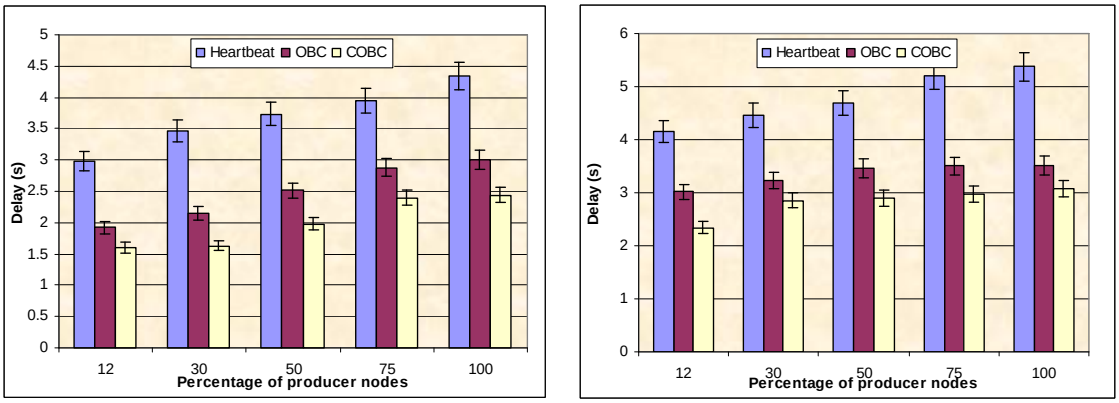


Figure 3.5: Delay of temporal ACK process in (a) 75- and (b) 100-node networks.

Obviously, Heartbeat takes the longest time for a message to be ordered as the Actor should wait to receive a heartbeat message from each node in the network with a timestamp greater than that of the event notification that needs to be ordered. In order to avoid collisions and lost messages, in the implementation of Heartbeat, we used random waiting times for each node before sending periodic messages so that the messages are not sent out in burst. Otherwise, it wouldn't have been possible to compare Heartbeat with the other algorithms considered.

With regards to OBC, the comparison of the results can be explained by the fact that in OBC, every parent node waits to receive temporal ACKs from the corresponding children nodes before forwarding the message toward the Actor node. In COBC, however, only the cluster head (CH) nodes wait to receive the ACK messages before sending the message further (to the cluster's nearest neighbour nodes as required by the underlying routing algorithm).

Figure 3.6 shows the combined delay of event ordering versus number of nodes. Here, all the cases of percentages of producer nodes are considered. COBC provides the lowest

delay of event ordering according to our experimental results.

As both COBC and OBC are "on demand" event ordering algorithms, as the percentage of producer nodes grows, the number of acknowledged event notifications in a single ACK step is increased as well. Thus, we compared the performance of our proposed algorithm with the OBC algorithm with regards to the delay per single temporally ACKed event notification. Figure 3.7 (a) presents the delay of temporal acknowledgment process per single event notification. As can be seen from the graph, the delay is decreasing as the number of nodes increases. This happens due to the fact that in a single temporal ACK step, a larger number (according to the percentages of producer nodes) of event notifications is temporally acknowledged, thus lowering the delay per single ACKed message. COBC consistently outperforms OBC in our experiments.

In Heartbeat's implementation, it is difficult to set the period *delta* that every node should wait before sending a heartbeat message toward the Actor in such a way that the required percentage of producer nodes' event notification messages would be delivered to the Sink before a heartbeat message is received from every node in the network with a timestamp greater than the last event notification message received at the Sink. The problem with Heartbeat is the non-determinism of delay, as discussed earlier. If a small interval *delta* is selected, the algorithm demonstrates message explosion as the messages will have to be sent out by each node with a higher frequency.

On the other hand, if a longer interval is selected, than the delay of event ordering is increased, as the Actor node will have to wait longer to receive heartbeat messages from every node in the network before ordering the event notifications. In our implementation of Heartbeat, there were cases when more than one event notification message, were ordered with a single heartbeat message coming from every node in the network. However, the percentage of producer nodes was not always the same as in OBC and COBC algorithms. Nevertheless, we included the Heartbeat's data on the graph in Figure 3.7 (b) to show the opposite trend in terms of the average delay of event ordering process per single ordered event notification. The delay is increasing as the number of nodes grows,

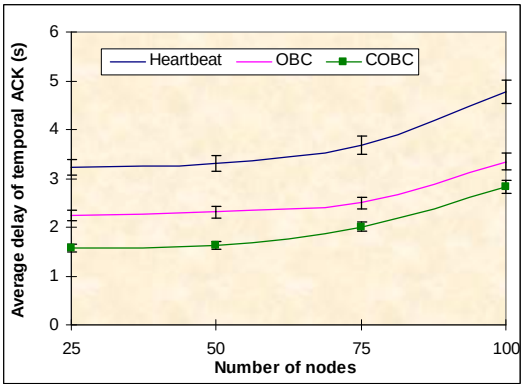


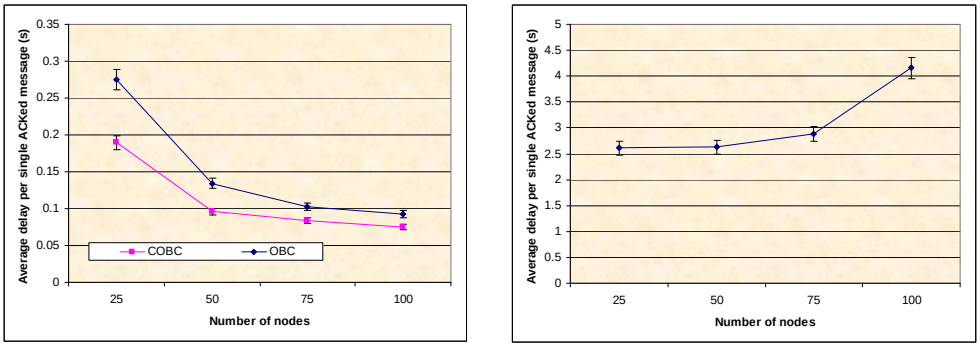
Figure 3.6: Delay of temporal ACK.

which confirms our hypothesis that on-demand event ordering algorithms have a lower overhead. Moreover, clustering resulted in a reduced delay of event ordering in COBC as compared with OBC.

Figures 3.8 and 3.9 show energy dissipation of event ordering algorithms when number of nodes grows. For OBC and COBC, it is the energy dissipation of the temporal ACK process. For Heartbeat, the energy dissipation is calculated for the period starting when an event notification message is received at the Actor and ending when it can be handed out to the application.

The most obvious observation from the graphs is that Heartbeat consumes significantly more energy for a message to be delivered to the application as compared with COBC and OBC for all network sizes and for all percentages of producer nodes considered. Figure 3.10 shows the combined energy dissipation versus number of nodes of event ordering process in the algorithms considered. Heartbeat's energy consumption is growing significantly as the number of nodes grows.

Moreover, as the percentage of producer nodes grows, i.e., more messages are received at the Actor, the possibility that the messages can be temporally acknowledged in a single step is increased in COBC and OBC. This in turn results in a gain in energy consumption as multiple messages are temporally acknowledged in a single ACK step. COBC provides



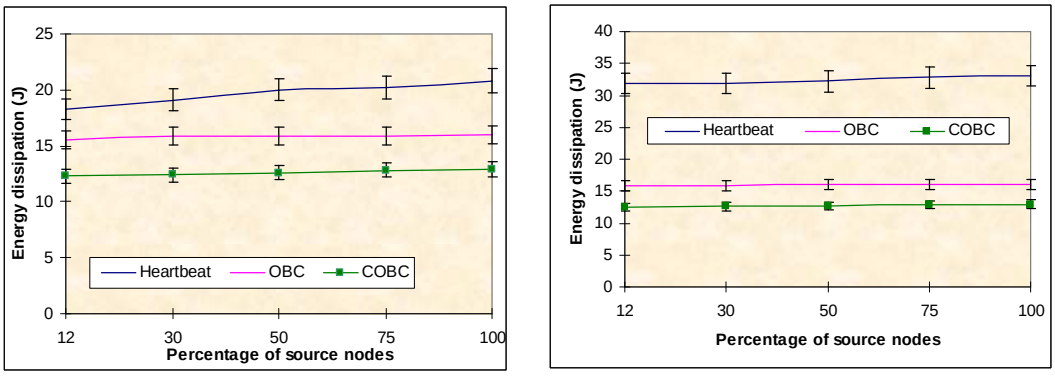
(a) **(b)**

Figure 3.7: Delay per single ACKed message: (a)OBC, COBC and (b)Heartbeat.

up to 15-20% reduction in energy dissipation as compared with OBC. This is mainly due to the lower delay of temporal ACK.

For COBC and OBC, to obtain the energy dissipation per single acknowledged event notification, we have calculated the energy dissipation per single event notification for each category of the experiments based on the percentage of producer nodes and then averaged the energy dissipation of the algorithm for the experiments grouped based on the number of nodes. Energy dissipation of temporal ACK process per single acknowledged event notification for different network sizes is presented in Figure 3.11 (a). As can be seen from the graph, COBC consumes less energy per single ACKed event notification as compared with OBC. We show energy dissipation per single ACKed event notification in Heartbeat in Figure 3.11 (b). As can be observed from the graph, similar to the delay of a single ACKed event notification, the energy dissipation per single ACKed event notification has an opposite trend to that of OBC and COBC.

The overall energy dissipation per single ACKed event notification is decreasing with the network size growing. This is explained by the fact that both OBC and COBC use



(a) **(b)**

Figure 3.8: Energy dissipation of temporal ACK in (a) 25- and (b) 50-node networks.

fewer acknowledgment steps, i.e., consume less energy to acknowledge more than one event notifications in a single acknowledgment step. Our results are with accordance with the results of OBC's performance presented by Boukerche *et. al* in [19].

3.1.2 Experimental Results of Time Synchronization Module

In this section, we present the results of experimental study of the time synchronization module SCT of the algorithm. To evaluate performance of SCT, we conducted simulation experiments in ns-2 network simulator. The experimental setting was the same as discussed in subsection 3.1.1. We experimented with a grid topology, in which the number of nodes varied from 25 to 100. Transmission range of a sensor node was set to 25m. In order to get an insight into the dependency of the number of CH nodes and the average node synchronization error, we experimented with varying percentages of CH nodes, ranging the percentage of CHs from 5% to 20%.

We conducted two groups of experiments to evaluate proposed algorithm's perfor-

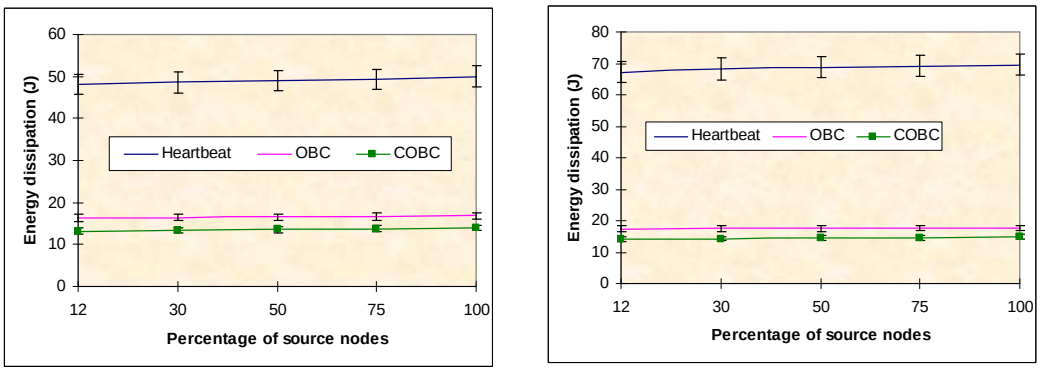


Figure 3.9: Energy dissipation of temporal ACK in (a) 75- and (b)100-node networks.

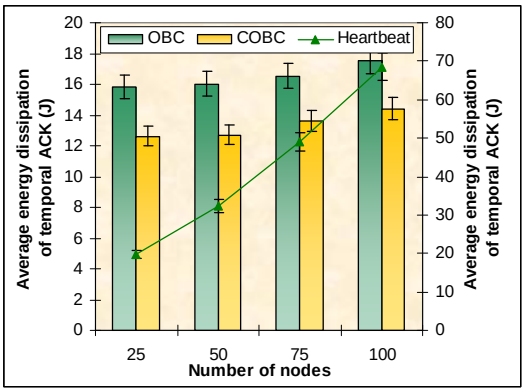
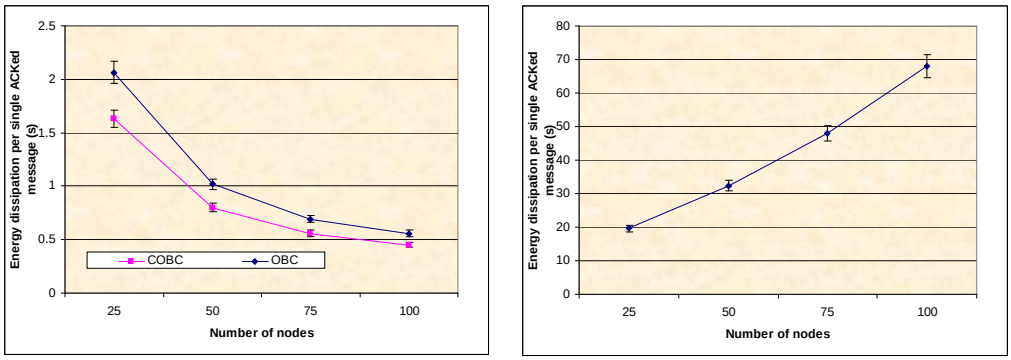


Figure 3.10: Energy dissipation of temporal ACK process.

mance. In the first group, for one-hop synchronization error in RBS we adopted 11μs and one-hop sync error in RTS 16.9μs [47], [89]. The second group of experiments considered timestamping of messages at the MAC layer in order to reduce the delay variability



(a) **(b)**

Figure 3.11: Energy dissipation per single message: (a)OBC, COBC and (b)Heartbeat.

by minimizing the *Send Time* and *Receive Time* components of delay as discussed in Chapter 2. For this group of experiments, we adopted a single hop synchronization error in RTS technique of 0.5μs (as was reported by Maroti *et al.* [71]). For RBS's single hop synchronization error in the second group of experiments in SCT we adopted the single hop synchronization error of 0.85μs (from the interval 1.85$\pm$1.28μs reported by Girod and Estrin in [47] for the case of kernel timestamping).

In addition, we present two versions of SCT in this chapter. In the first one - SCT - we combine the two synchronization techniques as discussed in Section 3.0.2. However, in order to evaluate the performance of the synchronization module of the algorithm, we also consider the case when nodes in the network are synchronized only by means of RTS synchronization technique. Thus, in the second version of the algorithm we used RTS synchronization method to synchronize cluster nodes with CH nodes and, in a similar way, CH nodes are synchronized with the Actor by means of RTS technique. This version of the algorithm is referred to as SCT+. In addition, we compared the performance of our proposed scheme with Baseline and FTSP [44]. Baseline algorithm synchronized all

the nodes in the network with RTS synchronization method, where every node would be synchronized with the Actor in a multi-hop fashion, i.e., every node is synchronized with a single reference point - the Actor. In the second group of experiments, we compared the results of our algorithm's performance with FTSP [71] algorithm.

In both groups of experiments, we considered the following metrics in order to evaluate performance of SCT and SCT+:

- *average node synchronization error versus number of nodes*;
- *average node synchronization error versus percentage of CHs*;
- *average node synchronization error versus single hop error*;
- *average node synchronization error versus network density.*

Figure 3.12 shows average node synchronization error versus number of nodes in the first group of experiments. From the graph, one can observe that the average node synchronization error is increasing as the number of nodes grows. SCT outperforms SCT+ in the experiments that we conducted. Average node error in Baseline algorithm is incomparably higher than that of SCT and SCT+.

The dependency of the percentage of CHs on the node average synchronization accuracy in SCT algorithm is depicted in Figure 3.13. We can observe from the graph that clustering results in a smaller average node synchronization error. This happens due to the fact that more nodes in the network are synchronized in a single hop, i.e., the reference point (which is a corresponding CH node) can be reached in a single hop. For a multi-hop synchronization, it implies shorter paths to the reference point and thus - a smaller accumulated synchronization error. This affects the overall node synchronization error. As the error accumulates from hop to hop, the idea of increasing the percentage of CH nodes seems reasonable in a clustered topology WSANs.

Actually, clustering reduces the average node error in both versions of the proposed algorithm - SCT and SCT+. This happens due to the fact that more reference points

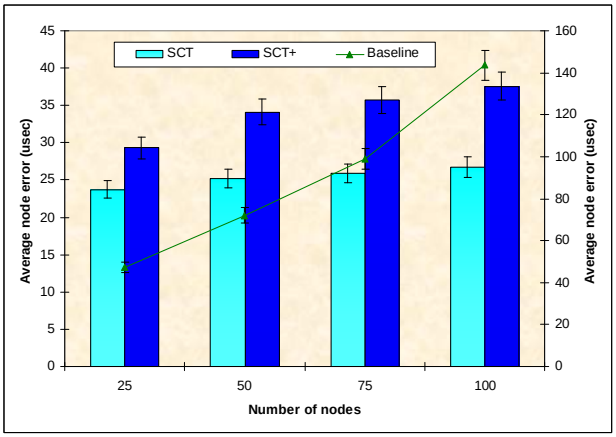


Figure 3.12: Average node error versus number of nodes.

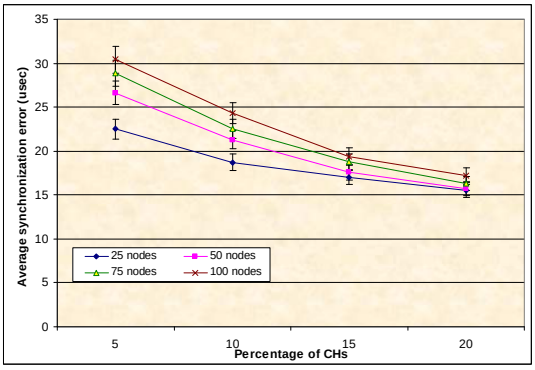


Figure 3.13: Average node error versus percentage of CHs in SCT algorithm.

are available for the nodes in the network to be synchronized with. CHs synchronize their corresponding nodes in a single hop in both versions of the algorithm. Synchronization error accumulates when a synchronization pulse or reply travels multiple hops between the sender and receiver nodes. When a larger portion of nodes, however, are synchronized in a single hop, the resulting average node synchronization error is reduced. Consequently, also the resulting cumulative error decreases for multi-hop synchroniza-

tion. While in SCT, the CHs are synchronized with each other with RBS technique, an additional gain in the average hop synchronization error is obtained as compared with SCT+.

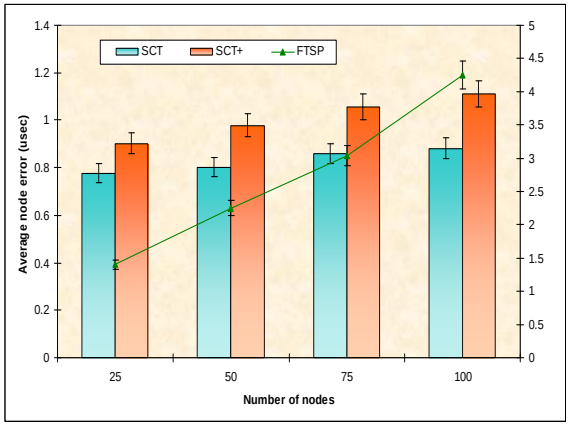


Figure 3.14: Average node error versus number of nodes (MAC timestamp).

We now discuss the simulation results in the second group of experiments. Figure 3.14 shows average node synchronization error versus number of hops in the second group of experiments. Similarly to the results obtained in the first group of experiments, the average node synchronization error is increasing as the number of nodes grows. An increasing percentage of CHs results in a decreased average node synchronization error. With regards to the difference in performance of SCT and SCT+, one can observe that it is not significant in the second group of experiments: even though SCT performs better than SCT+, the gain is not significant. Both SCT and SCT+ scale well, whereas FTSP demonstrates poor scalability. When the number of nodes grows, the average node synchronization error grows significantly.

To obtain an insight into the impact of a single hop synchronization error on the synchronization accuracy of the nodes, we varied the single-hop error in our simulation experiments from 5μs to 20μs. The results are presented in Figure 3.15. The single hop

synchronization error has a considerable impact on the average node synchronization error for all the algorithms considered in our simulation. However, we can see that the FTSP algorithm heavily relies on the single hop synchronization error, whereas in SCT and SCT+, even though the average node synchronization error increases when the single hop error does, the dependency is not that significant as compared with FTSP's performance in our set of simulations.

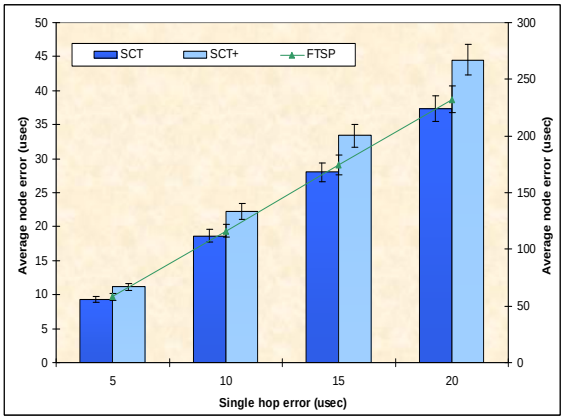
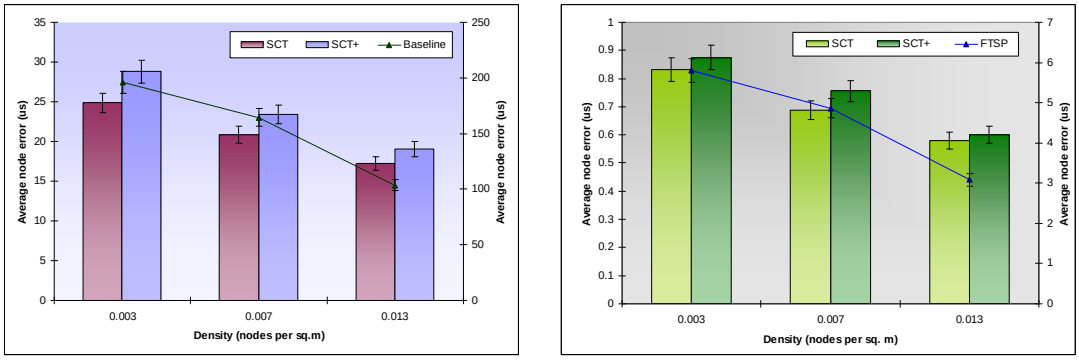


Figure 3.15: Impact of a single hop sync error on average error in 100-node networks.

In the next set of experiments, we varied the density of the network to examine the dependency of the average node error on node density. Figure 3.16 shows the graphs for both groups of experiments that we conducted for 100-node networks. The graphs for 25-, 50- and 75-node networks demonstrate a similar trend to the ones presented in Figure 3.16. We can see from the graphs that as the density grows (i.e., the number of nodes remains the same but the network area is decreased), the average node error decreases. This happens mainly due to the fact that as the nodes become closer to each other, more nodes will be included into clusters and thus they will be synchronized with their corresponding CH nodes in a single hop. Another reason is that free nodes are synchronized with CH nodes in a shorter hop-length path. This all contributes to

a decreased length of multi-hop synchronization in general, which has an immediate impact on the average node synchronization error. We can see from both graphs that in the second group of experiments, the dependency of the average node error on the density is mimicking that of the first group of experiments. As already discussed above, the difference in the performance of SCT and SCT+ is not significant in the second group of experiments and it is demonstrated once again in the graphs in Figure 3.16.



(a) **(b)**

Figure 3.16: Density vs node error in (a) first and (b) second groups of experiments.

When varying the node density, we ran experiments with a varying percentage of CH nodes. Figure 3.17 and Figure 3.18 show the dependency of the average node synchronization error on the node density when percentage of CH nodes varies in the first group of experiments. The results of the second group experiments are shown in Figure 3.19 and Figure 3.20.

The observation from the experiments was that as the density is increased, and the percentage of CH nodes grows, the resulting average node error is decreased. This again supports the above observation that in a denser network as a percentage of CH nodes grows, the number of cluster nodes is increased, thus resulting in a smaller average node

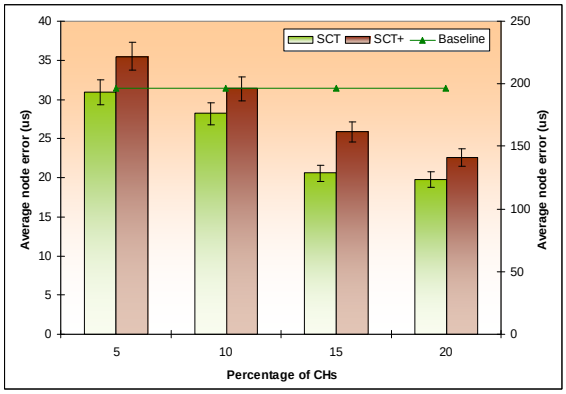
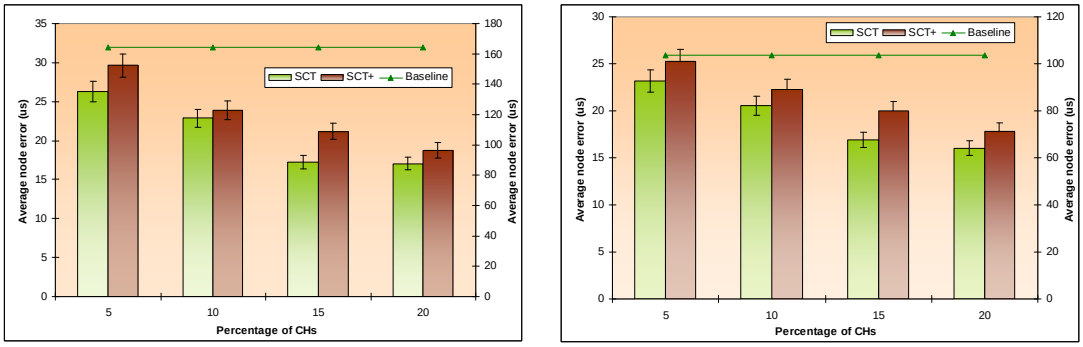


Figure 3.17: Node error in $30400m^2$ network area.



(a) **(b)**

Figure 3.18: Node error in (a)$15200m^2$ and (b)$7600m^2$ network areas.

synchronization error. As the number of CHs grows, in turn, the number of cluster nodes is increasing, thus contributing to an increased number of nodes that are synchronized in a single hop and a shorter path length for a multi-hop synchronization to a reference point (which is a CH) for free node's synchronization. The trend of a decreasing average node error is evident when comparing the graphs in the figures. With regards to performance of

Baseline and, similarly, FTSP, one can conclude that the average node error is decreasing when the density of the nodes grows due to the fact that in a denser network, the resulting paths to the reference point (the Actor) are shorter than those in sparse networks. This results in an overall lower node synchronization error in denser networks.

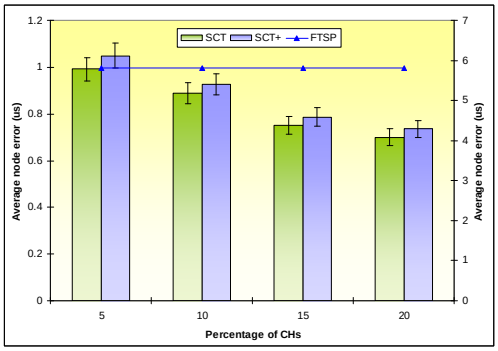
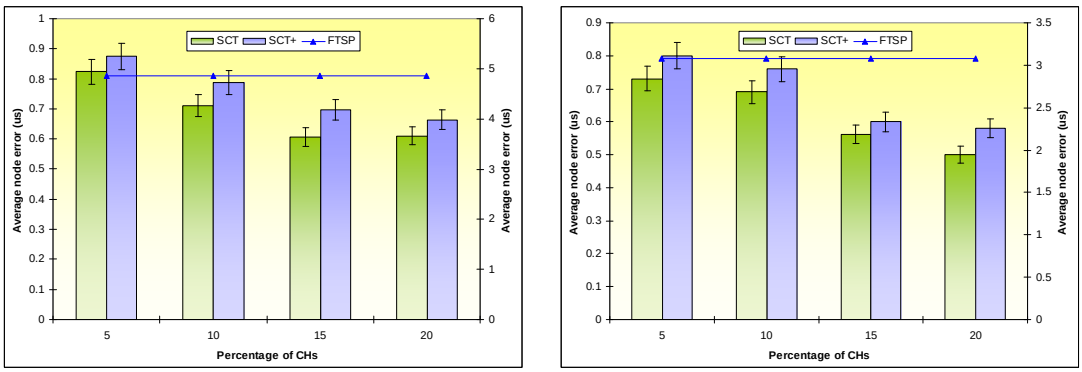


Figure 3.19: Node error in $30400m^2$ network area (MAC timestamp).



(a) **(b)**

Figure 3.20: Node error in (a)$15200m^2$ and (b)$7600m^2$ network areas (MAC timestamp).

Due to piggybacking of synchronization pulses and replies on the messages exchanged

in the underlying routing and temporal event ordering algorithms, the algorithm gained in terms of the number of messages sent. Figure 3.21 presents the number of messages that were used to piggyback synchronization pulses and replies in both the routing and event ordering algorithms during simulation time of 1000s. As can be seen from the graph, the number of messages piggybacked with sync pulses and replies grows as the network size grows.

The results include average number of messages for different percentages of producer nodes as discussed in subsection 3.1.1. The messages that were piggybacked with synchronization pulses for RBS technique included the broadcasts of Actor (for sending subscription messages in the routing algorithm as discussed in Chapter 4 as well as sending temporal ACK requests as discussed in section 3.0.1); synchronization replies for the nodes synchronized with RBS technique were piggybacked on the event notification messages traversing via a CH node as well as temporal ACKs traversing via a CH node. For the cluster nodes and free nodes that were synchronized with RTS technique, the message exchange necessary during cluster formation was used, as well as the messages travelling from cluster nodes to their corresponding CH nodes during event notification phase of the routing algorithm as well as temporal ACK messages travelling from cluster nodes to their corresponding CH nodes. Consequently, the proposed approach saved energy on the synchronization pulses and replies, which would have been sent explicitly for synchronization purposes if they hadn't been piggybacked.

Time synchronization needs to be performed in a WSN continuously, for the entire period of network's operation in order to cope with the clock drift of the nodes. Thus, the approach of piggybacking sync pulses and replies on the message exchange necessary in underlying protocols is a logical and efficient way to synchronize nodes while consering network resources.

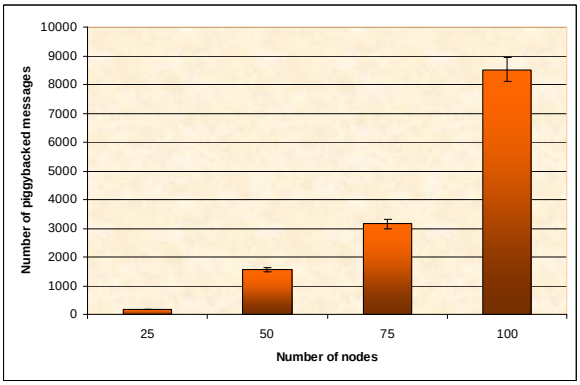


Figure 3.21: Messages piggybacked with synchronization pulses and replies.

3.1.3 Summary

In this chapter, we presented an efficient algorithm for preserving temporal relationships of the events sensed in WSANs that consisted of two modules - temporal event ordering and time synchronization. COBC event ordering algorithm for WSANs aimed at correct interpretation of events at the Actor; it considered modifications to OBC event ordering algorithm by introducing clustering concept into network topology with the goal of decreasing the overall overhead of temporal ordering mechanism: decreasing latency of event ordering process and improving its energy cost. The simulation results demonstrated that the proposed COBC algorithm is capable of obtaining the set goals. The delay of the temporal ACK process in COBC was reduced due to the proposed modifications to OBC, while the energy dissipation of the process was reduced as well. Comparison with Heartbeat algorithm showed that COBC, being an "on demand" event ordering algorithm, has the ability to significantly reduce energy dissipation and delay of event ordering as compared with the periodic approach to event ordering. Thus, COBC provides a tool for temporally ordering events that are sensed in WSANs with a smaller

overhead.

The discussed in this chapter SCT algorithm, which is tailored to the clustered topology of WSANs, employs two synchronization techniques - RTS and RBS. The combination of the techniques proved to be appropriate for the topology. SCT outperformed SCT+ (SCT algorithm's variation, in which only RTS synchronization technique was used) in all the sets of experiments, which demonstarted that the selection of the two synchronization techniques in our algorithm was tuned to the communication type of the clustered topology. The gain in the performance of SCT over SCT+ in the first group of experiments was greater than that in the second group. The latter can be explained by a lower single hop error in RTS technique in the case of MAC timestamping. We presented simulation results of SCT and SCT+ algorithms and compared their performance with FTSP and Baseline algorithms. SCT outperforms the other algorithms in terms of providing a higher synchronization accuracy. As the network scaled, the degradation in synchronization accuracy was not as considerable in SCT as in Baseline and FTSP algorithm.

With regards to the varying percentages of CHs, our experiments demonstrated that SCT's accuracy increased as the percentage of CH nodes grew due to the availability of an increased percentage of reference points and, consequently, the increased number of nodes that were synchronized in a single hop. Experimental results also have shown that FTSP algorithm heavily relies on the single hop synchronization error, whereas the dependency of the average node synchronization error on the single hop error is not as strong in SCT and SCT+. Synchronization accuracy benefited from an increased node density and increased percentage of CH nodes.

Furthermore, due to the approach of piggybacking synchronization pulses and replies on the messages exchanged in underlying event ordering and routing algorithms, the proposed algorithm achieved a considerable gain which we demonstrated in the number of messages that were piggybacked for synchronization purposes.

Chapter 4

Inter-Cluster Communication based Routing in WSNs

In this chapter, we present our proposed routing algorithm for WSNs. The *Inter-Cluster communication based Energy aware and fault tolerant protocol for WSNs* (ICE) is a clustering routing protocol that employs *Publish/Subscribe* paradigm [41]. The algorithm can be divided into three separate phases: setup, subscription propagation and event notification propagation. In the following subsections, we describe each of the phases.

4.1 Setup Phase

The setup phase starts with the selection of cluster head (CH) nodes. We adopted the idea presented in LEACH [53] based on the probability for each node in the network to become a CH/aggregator node. The algorithm selects 5% of the total number of nodes to be CHs. In LEACH, after each node generates a random number between 0 and 1, a node becomes a CH if it generates a number less than the threshold shown in equation 4.1, where p is the desired percentage of CHs (e.g., 0.05), r is the current round, and G is the set of nodes that have not been selected CHs in the last $1/p$ rounds. The idea behind the method of selecting CHs in LEACH is that CHs change over time in order to

balance the energy dissipation of the nodes.

$$T(n) = \begin{cases} p/(1 - p * (r mod 1/p)) & \text{if } n \in G\,, \\ 0 & \text{otherwise}\,. \end{cases} \tag{4.1}$$

In our algorithm, however, the selected nodes do not become CHs, we call these nodes beacon nodes since they are used as network explorers in the algorithm as described below. A beacon node requests the energy level from its surrounding nodes (which are one-hop away from it). It then selects the node with the highest energy level to be a CH. The CHs then generate clusters. CH nodes will be used as aggregator nodes, where the sensed data will be aggregated (for instance, if the sensors sense temperature, their readings may be averaged at the CH node and a single message will be sent toward the Sink instead of sending separate messages from all the cluster nodes that have sensed the event).

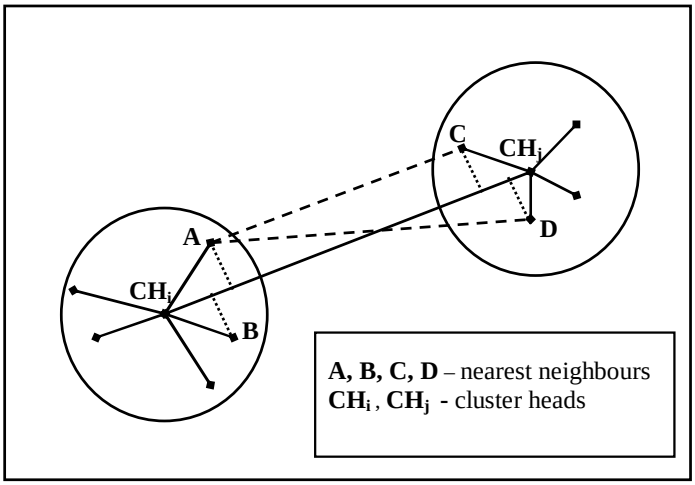


Figure 4.1: Finding nearest neighbours between clusters.

The next part of the setup phase is to learn about neighbouring clusters. The beacon nodes are used as explorers: they discover the neighbouring CHs within their transmission radius. The beacon node then broadcasts the information (the coordinates) about the CHs in its transmission radius. Once a CH receives this information, it learns of its neighbouring clusters and builds nearest neighbour (NN) tables to these clusters.

To find NNs, every CH identified by the beacon node draws a logical line between itself and the other CHs and projects the nodes in its cluster to that line (e.g., nodes A and C in Figure 4.1). Thus, two neighbouring clusters communicate with each other by means of nearest neighbour nodes as it ensures the shortest transmissions between two neighbouring clusters. Then, in its table of nearest neighbours, the CH stores the first two NNs (i.e., the first NN and the second NN) to $Cluster_2$ (see Table 4.1). The CH uses this information for routing and providing energy efficiency. The neighbours responsible for inter-cluster communication will be active in turns (e.g., nodes A and B in $Cluster_1$) as described below.

Table 4.1: Table of Nearest Neighbors to $Cluster_2$ from $Cluster_1$ (kept at CH_1).

	Nearest neighbours
To Cluster$_2$	{A, B}

Every CH will stay in that role for a certain period of time. In other words, the algorithm goes through rounds in order to cope with the dynamics of the network. It is necessary to note that during a round, if a CH's energy drops below a certain threshold, the CH selects a node from its cluster to be a CH. The last phase of the setup is the discovery of free nodes (the nodes that do not belong to any cluster). The free nodes broadcast a search message. Any node belonging to a cluster, when it hears the message, forwards it to the CH node. The message contains the coordinates of the free node. This way, the free node associates itself with the CH node. Overhearing is used in the algorithm, which helps the discovery process of CHs. For instance, when a node sends a merge request to a CH node, free nodes overhear the message and ask the prospective

cluster node to be associated with the CH node. The latter then forwards the message to the CH node.

4.2 Subscription Propagation

As part of the *Publish/Subscribe* paradigm, the Sink sends its subscription message to the network. The message contains the information in which the Sink is interested. For instance, the Sink could be interested in the event "Temperature is greater than 35 degrees". Thus, when any node senses temperature greater than 35 degrees, it will send a notification message toward the Sink (as discussed in Chapter 1). The propagation of the message is accomplished by using the constructed clusters. Thus, only a subset of nodes is involved in the message's propagation, resulting in energy savings compared to the case when the whole network is flooded with the message. When a subscription message leaves a cluster, one node from the pair of NN nodes will go to sleep. The CH will broadcast a TDMA schedule for these nodes. The nodes will stay asleep unless they sense an event or wake up according to the TDMA schedule.

When a cluster node hears a message from the Sink, it forwards the subscription message to its CH. The CH has information about all of its nodes' coordinates. It identifies whether the subscription is intended for any of its nodes. If yes, then the message propagation stops after the CH forwards the subscription message to the node. Otherwise, the message will be forwarded to other clusters by using NN nodes and intermediate nodes as described below.

Every time when a message leaves a cluster, a timeout is set up. If the sending node does not receive an ACK message from a node in the destination cluster, the algorithm uses intermediate nodes for the further relaying of the message (nodes A and B in Figure 4.2). This can happen due to insufficient transmission radius.

The message is then retransmitted with the flag "intermediate = ON" in its header. Every message going out from a cluster has its sender and destination cluster IDs (coor-

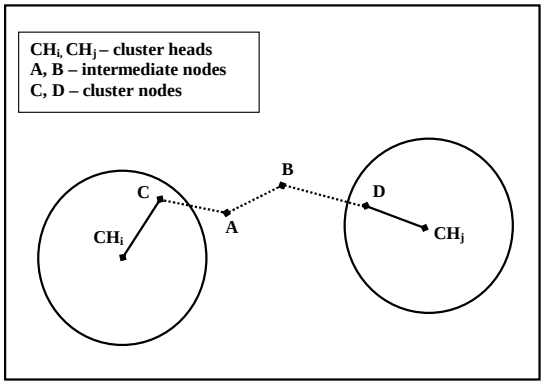


Figure 4.2: Using intermediate nodes for inter-cluster communication.

dinates of the corresponding CH) as shown in Figure 4.3.

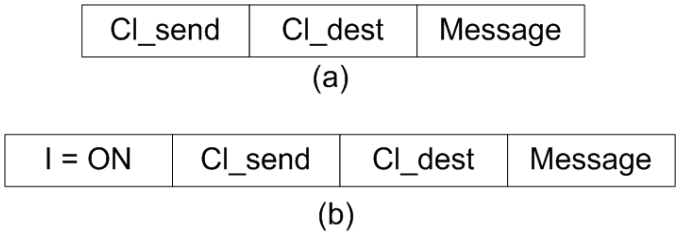


Figure 4.3: Message format when (a) NNs and (b) intermediate nodes are used.

When a message is relayed from $Cluster_i$ to $Cluster_j$, the NN node in $Cluster_j$ hears the message (the header contains the information about the sender) and relays it to its CH. The subscription message will be relayed further to neighbouring clusters. Any cluster node that receives a message with the flag, relays the message to the CH.

Table 4.2: Table of Subscriptions (kept at CH_1).

EventID	SinkID	SenderID
	$Sink_1$	$Cluster_2$, $Cluster_3$

The CH then takes care of relaying it further to the neighbouring clusters in a similar way. The CHs store information about every subscription message that goes through them. It consists of the *eventID*, *SinkID* and the *SenderID* (the cluster from which the message has arrived at the current cluster).

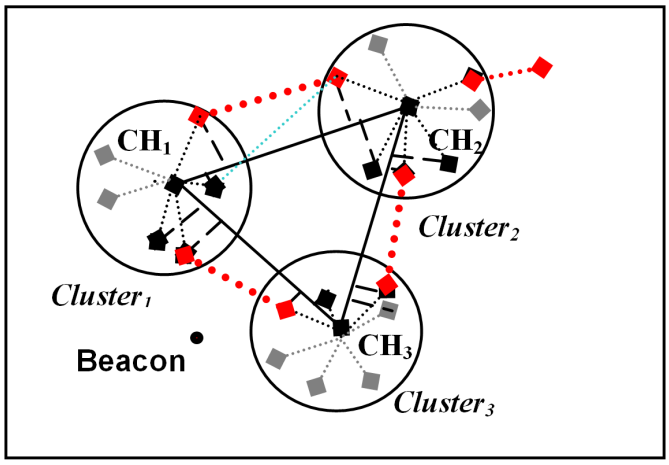


Figure 4.4: Nearest neighbours are used as the cluster's representative nodes.

For instance, if the same subscription message reaches $Cluster_1$ from $Cluster_2$ and $Cluster_3$ (as shown in Figure 4.4), then the table of subscriptions kept at CH_1 will store that information (Table 4.2). The routes through different clusters will be used later on for sending event notification messages toward the Sink as discussed in the following subsection.

When a subscription message propagates throughout the clusters, the CH checks not only whether the subscription is intended for the nodes within the cluster, but also whether the subscription is intended for any free node(s) that it knows about. If it is, the CH will use its cluster nodes to inform the free node about the subscription. The free node will thus learn about the Sink's interest.

When free nodes that are not associated with clusters receive a subscription message, they broadcast the message with an intermediate flag on. This message will propagate the network further. The message is then counting the number of hops that it propagates.

This way, when a free node receives a message that was broadcast by another free node, it updates its minimum hop neighbour and stores the minimum hop neighbour's address. A reverse path to this path is used later on, during notification's propagation via free nodes that are not associated with clusters. Algorithm 3 shows a high-level pseudocode for subscription message's propagation through WSN.

Algorithm 3 A HIGH LEVEL PSEUDOCODE OF SUBSCRIPTION PROPAGATION.

{Sink sends a subscription message containing the coordinates of the node, for which it is intended.}

1: **while** subscription message is received **do**
2: **if** the receiver node is a cluster node **then**
Action: Send subscription message to the cluster head (CH) node.
3: **else if** the receiver node is a cluster head **then**
Action: Store sender clusterID, sinkID, messageID.
4: **if** has not yet forwarded the subscription message **then**
{check whether the subscription message is intended for its cluster node}
5: **if** subscription message is intended for its cluster node **then**
Action: Send subscription message to that cluster node.
{check whether the subscription message is intended for an associated free node}
6: **else if** subscription message is intended for an associated free node **then**
Action: Send subscription message to the cluster node via which the free node was associated.
7: **else**
Action: Broadcast the subscription message for the nearest neighbour nodes to be forwarded toward neighbouring cluster(s) (except from which received).
8: **end if**
9: **end if**
10: **else if** the receiver node is an associated free node **then**
Action: Send the message toward the associated cluster head via the cluster node with which the free node is associated.
11: **else**
{the receiver node is a free node}
Action: Broadcast the subscription message.
12: **end if**
13: **end while**

4.2.1 An Example of Subscription Propagation

We now demonstrate with the help of an example how the subscription propagation process is accomplished in our protocol. In the example in Figure 4.5, $Sink_1$ broadcasts

a subscription message intended for node S and the receiver nodes are cluster nodes. Let us suppose that the three CHs (CH_1, CH_2 and CH_3) were discovered by a single beacon node and the CHs have learned about each other in the setup phase of the algorithm. The steps that the algorithm goes through for subscription propagation are shown below:

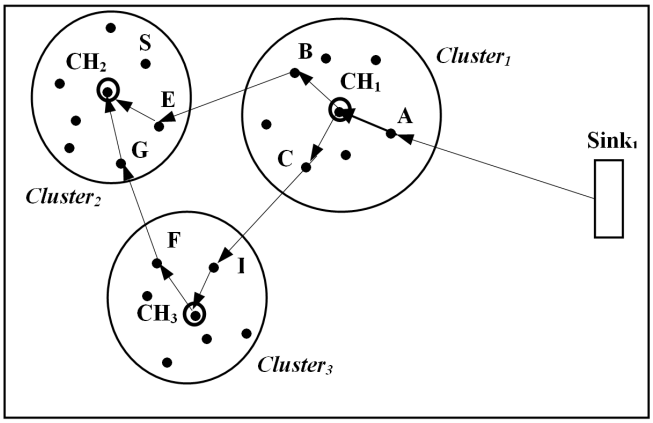


Figure 4.5: Subscription's propagation when receiver nodes belong to clusters.

1. $Sink_1$ broadcasts a subscription message intended for node S;

2. Cluster node A receives the message and forwards the message to its cluster head CH_1;

3. CH_1 checks whether the subscription is intended for its nodes and the free nodes that it knows about; CH_1 stores the *eventID*, *SinkID* and the *SenderID* in its Table of Subscriptions;

4. CH_1 sends the message to nearest neighbour nodes B and C to forward it to neighbouring clusters as the subscription message was not intended for the cluster's nodes and free nodes that CH_1 knows about;

5. Nodes B and C - nearest neighbour nodes which are responsible for communicating with the neighbouring $Cluster_2$ and $Cluster_3$, respectively, receive the message and send it toward those clusters;

6. Nodes E and I (which are the active nearest neighbour nodes responsible for communicating with $Cluster_1$) receive the message and forward it to their cluster head nodes: CH_2 and CH_3, respectively;

7. CH_2 checks the subscription message and finds that it is intended for a node in its cluster - node S. Subscription message's propagation stops here; CH_2 adds the information about $Cluster_1$ and $Sink_1$ into its Table of Subscriptions;

8. CH_3 node sends the message to its nodes, which are responsible for communicating with neighbouring clusters (except for the cluster, from which the message was received). CH_3 sends the message to node F responsible for communicating with neighbouring $Cluster_2$; CH_3 adds the information about CH_1 and $Sink_1$ into its Table of Subscriptions;

9. Node F sends the message toward $Cluster_2$;

10. The active nearest neighbour node G in $Cluster_2$ receives the message and sends it to its cluster head node CH_2;

11. CH_2 checks the *messageID* and sees that the message has already been received from $Cluster_1$. Propagation stops here; CH_2 adds the information about $Sink_1$ and $Cluster_3$ into its Table of Subscriptions;

The proposed algorithm provides QoS in that it prioritizes messages by finding a least cost path toward the Sink for high-priority event notifications. How it is achieved, will be discussed in the next subsection. However, we need to note that during subscription's propagation, CH nodes store residual energy of the path that was traversed by the subscription, as well as the number of hops that the message has traversed. This information is included into the subscription messages, it is modified while the message hops from node to node, thus incrementing the residual energy of the path traversed as well as the number of hops that the message has propagated.

4.3 Event Notification Propagation

The proposed algorithm aims at achieving even energy dissipation of the nodes. Thus, event notification is accomplished by using alternative routes every time a message is relayed toward the Sink. When $Cluster_i$ has a message to send toward the Sink, it chooses one of its neighbouring clusters (which it has learned about from the beacon node). It composes the message accordingly, by including that *destination clusterID* into the message.

The algorithm ensures that at any given point in time there is an active node from the NN table that is responsible for inter-cluster communication. The message will be received by that node and further relayed to its CH. The intermediate nodes will be involved in the message relaying when the "intermediate" flag is on in the message. When a free node senses an event, depending on the status of the free node, whether it is associated with a CH node or not, it will do the following. If the free node is associated, the node will forward the event notification toward the CH with which it has associated itself in the setup phase of the algorithm. Otherwise, the node will use the reverse path to the path traversed by the subscription message for sending the event notification. As we described in the previous subsection, it will have stored the node from which it had received the subscription.

As noted, during the subscription propagation phase, the CH stores the *senderID* of the cluster that has forwarded that subscription message to the current cluster. Once the event occurs, the CH assigns the node that is responsible for communicating to that cluster to forward the event notification message toward the Sink. Once the neighbouring cluster's node receives that message, it will in turn forward it to its CH. By checking the *SinkID* field in the message, this CH will know to which cluster to forward the message so that it eventually reaches the Sink.

Every CH has a cache of a few events last sent to the Sink. If a CH identifies redundant event notifications, they are dropped to avoid unnecessary network traffic.

Every event notification has a unique ID. The uniqueness of an event notification is achieved by concatenating a CH's coordinates with the timestamp of the event.

As already mentioned, ICE provides QoS by prioritizing event notifications. The algorithm selects a path with the least cost for a high priority event notification message. When an event notification message propagates via a CH node, the latter checks the priority level of the message. If the message has a high priority (for the cases when the observed in the network parameters reach critical or near-critical values), every CH that the message goes through on the way from a source node to the Sink identifies a path with the least cost and forwards the message accordingly via that path.

As there are multiple paths that are created during the subscription propagation phase of the algorithm, it is possible to select the path with the least cost for a high-priority event notification's delivery to the Sink. The cost function that we use is simple yet efficient. It takes into account the number of hops that the subscription message has propagated in the second phase of the algorithm as well as the residual energy of that path. The cost function of path i is calculated as shown in equation 4.2,

$$C_i = Hop\_Count_i / Eres_i \tag{4.2}$$

where Hop_Count_i is the number of hops from the Sink to the CH node and $Eres_i$ is the residual energy of that path. Algorithm 4 shows a high-level pseudocode for event notification message's propagation through WSN.

4.4 Properties of the Algorithm

In this section, we highlight the main properties of the algorithm, focusing on energy-efficiency, fault tolerance, QoS and network connectivity.

Algorithm 4 A HIGH LEVEL PSEUDOCODE OF NOTIFICATION PROPAGATION.

{A source node senses an event and sends an event notification message toward the Sink that subscribed to the event.}

1: **if** the source node is a cluster node **then**

Action: Send notification message to the cluster head node.

2: **else if** the source node is a cluster head OR an event notification is received at the cluster head node **then**

{check the priority level of the event notification}

3: **if** the event notification has a high priority **then**

Action: Consult the Table of Subscriptions and select the least-cost path toward the Sink.

4: **else**

Action: Consult the Table of Subscriptions and select a path toward the Sink.

5: **end if**

Action: Send the notification message to the nearest neighbour node belonging to that path to be forwarded toward a neighbouring cluster.

6: **else if** the source node is an associated free node **then**

Action: Send the message toward the associated cluster head via the cluster node with which the free node is associated.

7: **else**

{receiver node is a free node}

Action: Send the event notification to the minimum hop neighbour (using the reverse path to subscription message's propagation).

8: **end if**

4.4.1 Energy-Efficiency

- *In our algorithm, two neighbouring clusters communicate with each other by means of nearest neighbour nodes as this implies shortest distance possible between two neighbouring clusters.*

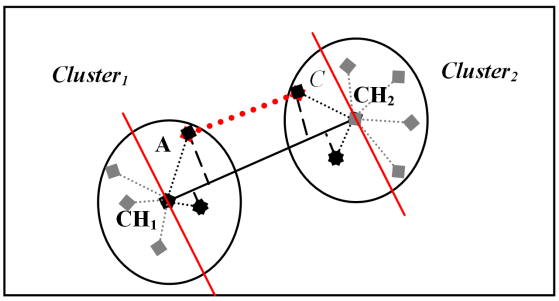


Figure 4.6: Nearest neighbour nodes for $Cluster_1$ and $Cluster_2$.

To find a NN node from CH_1 to CH_2, we project the cluster nodes onto the line

CH_1CH_2. Node A (Figure 4.6) is the NN from $Cluster_1$ to the CH_2 node. Node A is also a NN node to any node in $Cluster_2$ that is closer than CH_2. In the same way, we find that the NN node from $Cluster_2$ to CH_1 is node C. Thus, nodes A and C are NNs for $Cluster_1$ and $Cluster_2$.

- *Due to the energy efficient approach of alternating the nodes responsible for inter-cluster communication, the algorithm reduces the number of active nodes by $N_{sleeping}$ during the event notification phase.*

  Let m_i be the number of neighbouring clusters for a given cluster and k be the total number of clusters. Then

$$N_{sleeping} = \sum_{i=1}^{k} m_i \tag{4.3}$$

  represents the number of sleeping nodes in all of the clusters. The other aspects of ICE's energy efficiency is data aggregation: the CHs are also aggregator nodes. In addition, the process of subscription propagation also aims at preserving energy by using a subset of nodes as opposed to flooding subscription messages.

4.4.2 Fault Tolerance

Having two nodes be responsible for inter-cluster communication aims at providing fault tolerance. If one of the two nodes dies, the other node is still available for inter-cluster communication. Also, the algorithm takes advantage of the multiple paths created during the propagation of the subscription message. If one of these paths has faulty nodes, the other ones will be used for message propagation. If there is a broken path somewhere along the source-Sink route, it will be identified when the message is not acknowledged to the sender node.

- *The algorithm provides an alternative path when there is a faulty node on the route from source to Sink due to the storing the paths proactively.*

Let r=$\{r_1,r_2\}$ be a set of alternative routes to the Sink from CH_j. Let msg_j be the message broadcast by nearest neighbour node A, where $A \epsilon r_1$ from $Cluster_j$, and let the message be not acknowledged. Then msg_j will be sent toward the Sink through nearest neighbour node B, such that $B \epsilon r_2$.

4.4.3 Quality of Service

- *The algorithm uses alternate routes for event notifications' delivery to the Sink. The delay of some routes can be greater than that of the others and, to cope with this, the algorithm provides QoS by associating a cost with each of the paths. A path with the least cost is used for high-priority messages.*

  Let r be a set of alternative routes to the Sink from CH_j, created during subscription's propagation, where r_1, $r_2 \epsilon$ r; and Countr_1 and Countr_2 represent the number of hops on the paths r_1 and r_2, respectively. Let Eres_1 be the residual energy of the route r_1 and Eres_2 be the residual energy of the route r_2. Then C_1 = Countr_1/Eres_1 and C_2 = Countr_2/Eres_2 are the costs of routes r_1 and r_2 respectively. A high-priority message will always be sent through r_1 route if $C_1 < C_2$.

  The metric that we use for finding the cost C of a path reflects the requirements of a path to have a high residual energy and a small hop count. Delay of a message is highly affected by the delay associated with the MAC layer on the sender's side as well as the variable delays introduced by the operating system on both the sender's and receiver's sides as discussed in Chapter 2. The smaller the hop count, the fewer nodes will affect the delay of the path. At the same time, when delivering a high-priority event notification, it is desirable to select the path with a sufficient residual energy.

4.4.4 Network Connectivity

- *In our algorithm, nearest neighbours provide network connectivity on the level of clusters.*

  For every $Cluster_i$, at any given point in time t the algorithm ensures that there is one active NN node $n_i \in \{N_{ij}, N_{ij'}\}$, where N_{ij} and $N_{ij'}$ are the NN nodes responsible for communication with neighbouring $Cluster_j$. Thus, the clusters' connectivity is provided. Whenever the NN nodes of the neighbouring clusters are out of each other's range, intermediate nodes alleviate the problem of connectivity.

  When beacon nodes explore the neighbouring clusters, the network is divided into islands of clusters as shown in Figure 4.7, each of them being bounded by the transmission radius of the beacon.

  Definition 1: An island of clusters is a set of clusters whose cluster head nodes were found by one beacon node.

- *In our algorithm, the overlapping cluster heads identified by beacon nodes provide network connectivity between the islands of clusters.*

  If a beacon node B_1 within its transmission radius R_1 discovers a set of CHs s_1 such that $s_1 = \{CH_i, CH_j, CH_k\}$, and beacon node B_2 within its transmission radius R_2 discovers the set of cluster heads $s_2 = \{CH_l, CH_j, CH_m\}$, then the CH_j node provides network connectivity on the level of islands of clusters, whose CH nodes are discovered by beacon nodes B_1 and B_2.

  It is possible that there will not be overlapping CHs in two islands (e.g., in Figure 4.7 beacon nodes B_2 and B_6 do not have overlapping CHs). In that case, communication will be achieved by using intermediate nodes as described above.

  Definition 2: Any node that does not belong to a cluster is a free node.

- *When there does not exist an overlapping cluster head node in the sets of cluster head nodes found by two beacon nodes, the proposed algorithm ensures network*

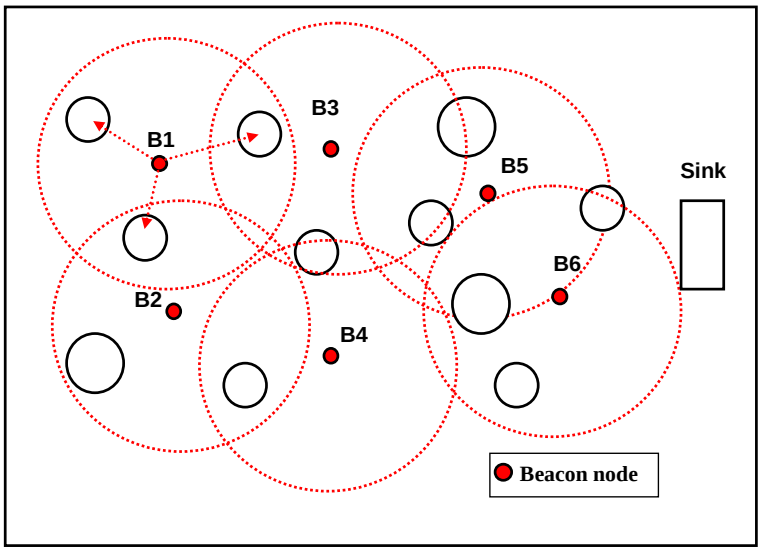


Figure 4.7: Neighboring clusters' discovery by using beacon nodes.

connectivity by means of free nodes and messages with "intermediate" flags.

We need to consider two cases depending on the status of the receiver node.

Case 1: The receiver node is a free node. A free node f that hears the message with an "intermediate" flag retransmits the message again with the flag. Any node that hears the message will transmit the message to its neighbouring nodes.

Case 2: The receiver node belongs to a cluster. The receiver node will forward the message to its CH. The CH will consult its Table of Subscriptions to identify the route through which to forward the message toward the Sink.

4.5 Extension to the Routing Algorithm involving Multiple Sinks

In this section, we present the extensions to ICE routing algorithm for multiple sinks. In what follows, we will describe the additional component of the routing protocol that deals with the propagation of messages in a WSN monitored by multiple Sinks. Figure 4.8 presents a WSN that is monitored by multiple Sinks. The proposed extensions have two main objectives: to improve QoS of the routing algorithm and provide an increased fault tolerance.

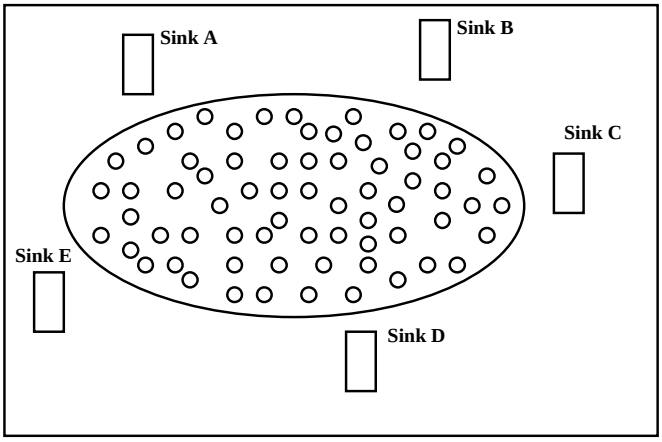


Figure 4.8: A WSN that is monitored by multiple Sinks.

4.5.1 Subscription Propagation

In the subscription propagation phase, each Sink subscribes to an event in the form of an interest subscription. When subscription messages from different Sinks propagate throughout the network, the paths created during the messages' propagation are stored, in a way similar to the single Sink scenario described earlier. Every subscription message

is unique, which is provided by the Sink that expresses an interest in an event. Also, the Sink includes a priority level into each subscription message so that, once the event occurs, the event notification will be attached with the corresponding priority level. The *eventID* is stored to ensure that the same subscription message will not be sent repeatedly to neighbouring clusters.

If a cluster head (CH) node already has a subscription message to send to its nearest neighbour node (in order for the node to send it toward a neighbouring cluster) and another subscription message is received, the messages will be sent out according to their priority (i.e. a high-priority message will be sent out first). If the messages are of the same priority, they will be treated according to their timestamps, in which case, the earlier event subscription message will be sent out first.

4.5.2 Event Notification Propagation

The multi-path property of the protocol comes in handy when more than one messages need to be sent from a CH node toward the Sink. The latency associated with sending multiple event notifications by means of using these paths will be reduced compared to a single path routing protocol. At the same time, the event notifications will be treated according to their priority as described below.

Similar to a single Sink scenario, interest subscription messages propagate throughout the network until they reach the area of interest. All these paths are stored at the CH nodes through which the subscription message propagates (basically, only the *SinkID*, *eventID* and *senderID* of a sending cluster). Reverse path(s) are used for sending an event notification message once the event of interest is sensed. Once an event notification is received at the CH node, it is placed in a proper buffer depending on the priority field in the event notification message.

If the event notification messages that need to be forwarded by a CH have different priorities, the CH will consult its Table of Subscriptions to find out whether it has more than one entry in the field of *ClusterIDs* of the clusters via which this node has received

the subscription message in question. If there is more than one entry in the field, the CH will forward an event notification with a high priority via a route with the least cost. As we mentioned above, the cost function that we have chosen for selecting a path is based on the residual energy and hop count of a path. Thus, the route will be chosen accordingly: the event notification with a high priority will be forwarded toward a nearest neighbour node that belongs to the corresponding route. Every CH node will do the same when the event notification traverses through it. Thus, the event notification with a high priority will be forwarded toward the corresponding Sink via a route with the least cost.

If the Sinks all have access to the Internet (i.e. they are gateway nodes), we propose to build a distributed DBMS (DDBMS) over them and in the global schema of the DB, have *SinkID* as one of the attributes. In this case, every Sink can query the DDBMS and get the results in which it is interested, i.e. it can see whether an event, in which the Sink has expressed an interest, has already been sensed and the corresponding event notification message has been delivered to one of the Sinks. The main purpose of the proposed scheme is to provide QoS by using the paths toward the multiple Sinks with the goal of selecting the least cost path. These paths will be selected for high-priority event notification messages in order to reduce the resulting delay of the path used for delivering the event notification. The details on the implementation of the DDBMS are out of scope of this thesis.

Also, regarding low-priority event notifications, if none of the paths from a CH toward the Sink that has subscribed to an event is available (i.e. the paths have faulty nodes), a path toward a different Sink than the one that has subscribed to the event is selected for the event notification to be sent through. This feature also provides increased fault tolerance. At the same time, all the Sinks responsible for monitoring the WSN will have a global view of the conditions of the network. Their interests, in fact, might overlap: some Sinks can be replicated to increase the network's fault tolerance. A high-level pseudocode for event notification's propagation through a cluster head node in a

multiple Sink scenario is shown in Algorithm 5.

Algorithm 5 A HIGH LEVEL PSEUDOCODE OF NOTIFICATION PROPAGATION VIA A CH NODE IN MULTI-SINK WSN.

{A source node sends an event notification toward a Sink; it arrives at a cluster head node.}
1: **if** receiver node is a cluster head **then**
2: **if** has not yet forwarded the event notification message **then**
{check the priority level of the event notification message}
3: **if** the event notification has a high priority level **then**
Action: Consult the Table of Subscriptions and find a least cost path (from the paths toward all the Sinks).
Action: Send the event notification message toward the Sink via a nearest neighbour node belonging to that path.
Action: Set ACK timer and wait.
4: **else**
{the event notification has a low priority level}
Action: Consult the Table of Subscriptions and find a path toward the Sink that subscribed to the event.
Action: Send the event notification message toward the Sink via a nearest neighbour node belonging to that path.
Action: Set ACK timer and wait.
5: **end if**
{Upon ACK timer expiration}
6: **if** no ACK received **then**
7: **if** the event notification has a high priority level **then**
Action: Consult the Table of Subscriptions and find a least cost path (from the paths toward the remaining Sinks).
Action: Send the event notification message toward the Sink via a nearest neighbour node belonging to that path.
Action: Set ACK timer and wait.
8: **else**
Action: Consult the Table of Subscriptions and find a path toward alternative Sink.
Action: Send the event notification message toward the Sink via a nearest neighbour node belonging to that path.
Action: Set ACK timer and wait.
9: **end if**
10: **end if**
11: **end if**
12: **end if**

Applications that can benefit from the proposed scheme, may include the monitoring of critical conditions, such as fires. Firemen carrying PDAs at a site of fire can be viewed as multiple Sinks/Actors. Having a Distributed DBMS, consisting of the event

notifications of the events sensed at the site, in such a particular scenario will allow the firefighters to have a global picture of the conditions at the site, i.e. the readings of the sensors for which a particular firefighter (Sink) is not responsible can be available to the firefighter for making decisions about the surrounding area, taking preventive actions etc.

Another possible application is in pervasive health monitoring systems, particularly those in emergency rooms, retirement homes etc. The medical personnel carrying mobile devices such as PDAs can be viewed as multiple Sinks as well. Patients wearing vital signs monitoring sensors (e.g., for heartbeat, temperature, blood pressure etc.) create a WSN. The data gathered from the sensors will be delivered to the Sinks. For high-priority event notification messages, when the observed parameters of the health conditions of a patient reach a critical value, the availability of multiple Sinks will allow the message to be delivered to the closest PDA through a more reliable path, thus reducing the delay and resulting in a prompt response from the medical community.

4.6 Performance Evaluation

We used ns-2 network simulator in order to evaluate the performance of the routing protocol. In the experiments, the number of nodes ranged from 25 to 100. The metrics that we used in our experiments were:

- *average energy dissipation (per node)*;
- *average success rate* (measured as the ratio of messages sent from source node and received by the Sink);
- *average delay of event notification messages*.

The simulation time was set to 1600s. The duration of a round was set to 800s. In each case, we varied the percentage of the producer nodes (source nodes, that sense an

event) from 12% to 100%. The reception (Rx) power was set to 0.395W, transmission (Tx) power was set to 0.66W and the idle power was set to 0.035W [57]. We used the transmission range of a sensor node of 25m. The simulation areas were set to $6400m^2$, $14400m^2$, $22400m^2$ and $30400m^2$ for 25-, 50-, 75- and 100-node networks, respectively. The experiments were executed 33 times with a confidence interval of 95%. The average rate of messages sent was set to one message per 0.8 seconds for subscription messages. For sending notification messages, we used a random delay (between 0 and 1 seconds) for each of the source nodes after the notification phase of the algorithm started.

4.6.1 Average Energy Dissipation

Figure 4.9 presents the average energy dissipation per node in the algorithm as the simulation time progressed. The graph shows the combined results of simulation experiments with varying percentages of source nodes. We here show the average energy dissipation per node during a single round of the algorithm. As can be seen from the graphs, the energy dissipation per node is very similar for all network sizes, with which we experimented. This is explained by the fact that the simulated settings were the same in all scenarios for different network sizes (such as the percentage of source nodes was the same, the percentage of CHs etc.). As the number of nodes in the network increased, so did the energy dissipation. However, the average energy dissipation per node remained similar.

Figure 4.10 shows energy dissipation per node in ICE and CPEQ algorithms. As can be seen from the graph, ICE outperforms CPEQ in terms of energy dissipation in our experiments. Energy dissipation is smaller in ICE mainly due to the energy-efficient approach of using low-duty cycling for the nodes responsible for inter-cluster communication. Also, in the subscription propagation phase, CPEQ employs a flooding of the network with a subscription message, whereas ICE utilizes the constructed clusters for disseminating the Sink's interest. Another factor that affects the energy dissipation is the way that ICE and CPEQ treat a situation when a broken path is identified on the

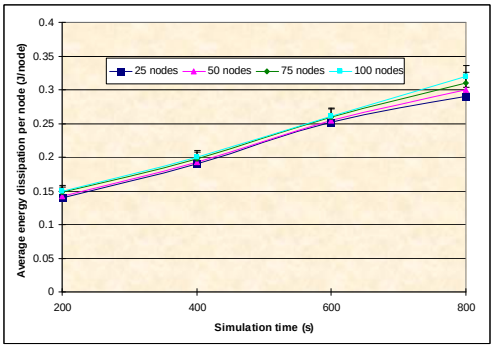


Figure 4.9: Energy dissipation per node in ICE.

route from source node to the Sink. In CPEQ, a SEARCH packet is broadcast when a node sends a packet and does not receive an ACK from the receiver node. An alternative node is then sought for. In ICE, on the other hand, the paths that were created and proactively stored during subscription message's propagation are used in this case. In other words, the time spent on selecting a different path is reduced in ICE as compared with CPEQ. This all resulted in a better energy dissipation of the nodes in ICE as compared with CPEQ. We can see from the graph that ICE shows a decrease in energy dissipation of 20% as compared with CPEQ.

4.6.2 Average Rate of Successfully Delivered Messages

Figure 4.11 graphs the dependency of the success rate on the percentage of source nodes. As the percentage of source nodes grows, the algorithm performs well under the increasing number of event notification messages being sent to the Sink node. The high success rate can be explained by the availability of proactively created and stored multiple paths from source nodes towards the Sink. This resulted in a high percentage of delivered event notification messages.

Figure 4.12 shows combined average rate of successfully delivered messages for differ-

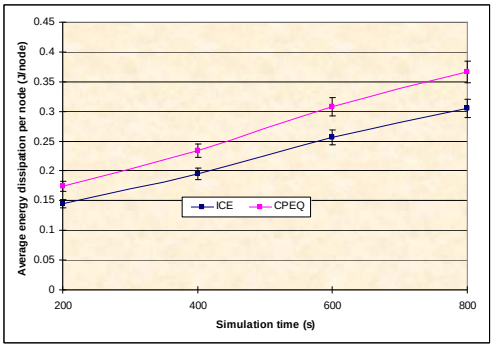
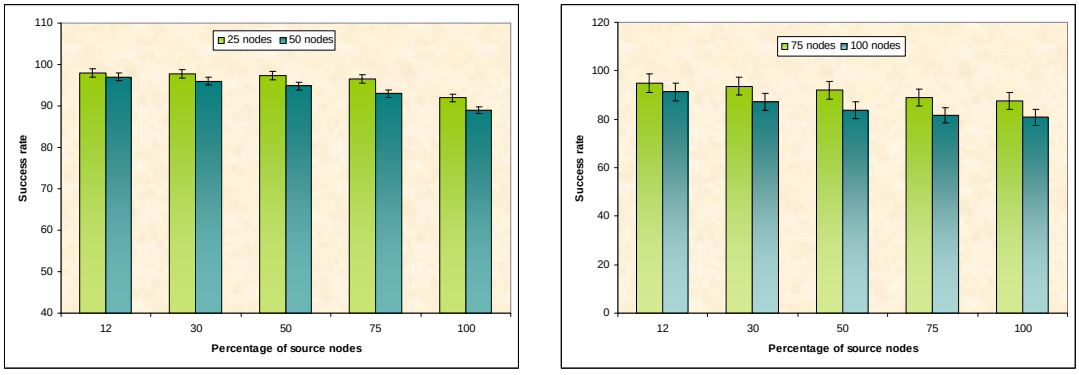


Figure 4.10: Energy dissipation in ICE and CPEQ.



(a) **(b)**

Figure 4.11: Success rate in (a) 25/50- and (b) 75/100-node networks.

ent percentages of source nodes. It is the highest for smallest networks and is gradually decreasing as the network size grows. As the number of nodes in the network as well as the percentage of source nodes grows, the probability of collisions and lost messages is increased. As a result, the success rate is decreased, however, ICE provides a high

success rate even under such stressful conditions such as having all 100% of nodes to be source nodes. This happens also due to the fact of selecting random waiting times before a source node sends its event notification toward the Sink. With regards to CPEQ, our simulation experiments showed that while ICE demonstrates a higher success rate of event notifications' delivery to the Sink, the gain is not significant as shown in Figure 4.12. Both the algorithms achieved a high success rate of event notifications' delivery to the Sink in our experiments.

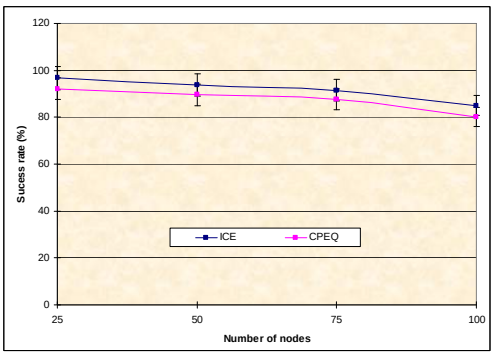


Figure 4.12: Success rate.

4.6.3 Average Delay of Messages

Figure 4.13 shows the delay of event notifications in ICE and CPEQ. We can observe that the difference in the delay of both algorithms is not big. This can be explained by the fact that ICE employs multiple paths alternatively for sending event notifications. The objective of this, as was discussed above, is load balancing. CPEQ, on the other hand, always forwards packets via the shortest path (minimal hop count) toward the Sink. It is necessary to note that ICE selects a path with the least cost (in which path length is one of the metrics considered, as already discussed) only for the event notifications with a high priority.

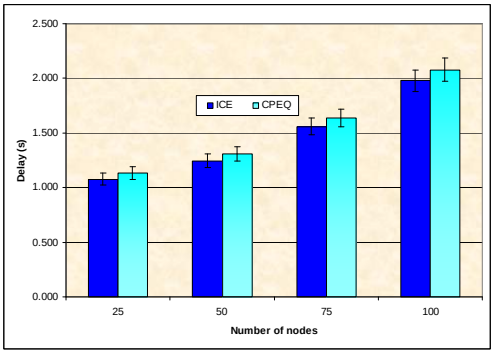


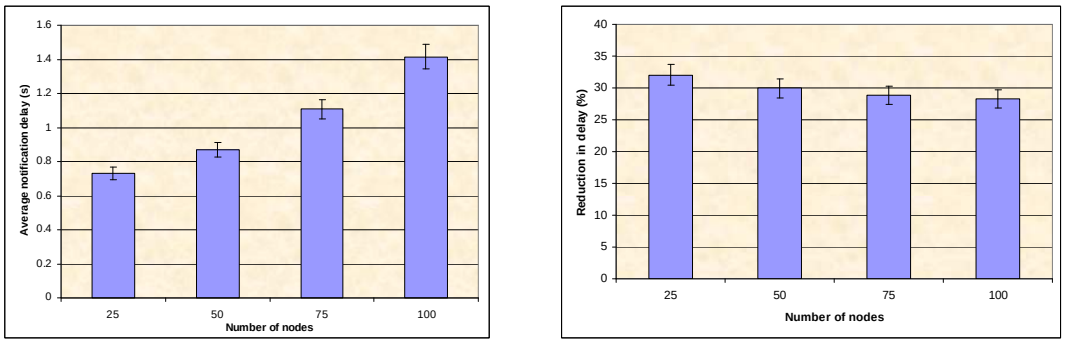
Figure 4.13: Notification delay.

4.6.4 QoS Feature

Figure 4.14 (a) shows the average notification delay of high-priority messages. The delay is decreased due to the employing the least cost path for the messages' delivery: the event notifications with a high priority were forwarded via routes with the least cost when traversing through CH nodes, as described in section 4.4.3. Figure 4.14 (b) demonstrates the reduction in delay of the event notification messages delivered to the Sink due to providing QoS in the set of experiments that we conducted.

CPEQ does not provide QoS, so we do not include the CPEQ's data on the graph in Figure 4.14. However, in CPEQ, a path with the least number of hops is selected every time a source node sends an event notification message toward the Sink. ICE, on the other hand, forwards only the event notification messages with a high priority via a path with the least cost (where the number of hops of the path is one of the parameters considered in the cost function). A comparison of the delay of high-priority event notifications with the delay of CPEQ confirms that ICE outperforms CPEQ when using a least-cost path for forwarding high-priority event notifications.

As can be seen from the graph, the QoS property of the algorithm allows to achieve a considerable reduction in the delay of event notification messages - approximately



(a) **(b)**

Figure 4.14: (a) Delay of high-priority messages; (b) Reduction in delay due to QoS.

30% reduction of delay. The gain is maximal in the smallest networks and is gradually decreasing as the number of nodes grows and, consequently, the average path length grows.

4.6.5 Extension for Multiple Sinks Scenario

For the extension of the routing algorithm, we experimented with the scenarios that aimed to investigate the variance of the event notification delay for the different settings of WSNs. The subscription delay is not affected by the availability of multiple Sinks and thus cannot be used as an indication of gain/loss due to the fact of employing multiple Sinks in our algorithm. Every Sink is subscribed to the event(s) that it is interested in. Only at the time when event notifications are sent toward the Sink(s) is the availability of multiple Sinks utilized by the algorithm to provide QoS and improve its fault tolerance, resulting in a higher reliability of the algorithm. Similar with the results reported in section 4.6.4, we observed a significant reduction in the event notification's delay for the

extended version of the routing algorithm as can be seen below.

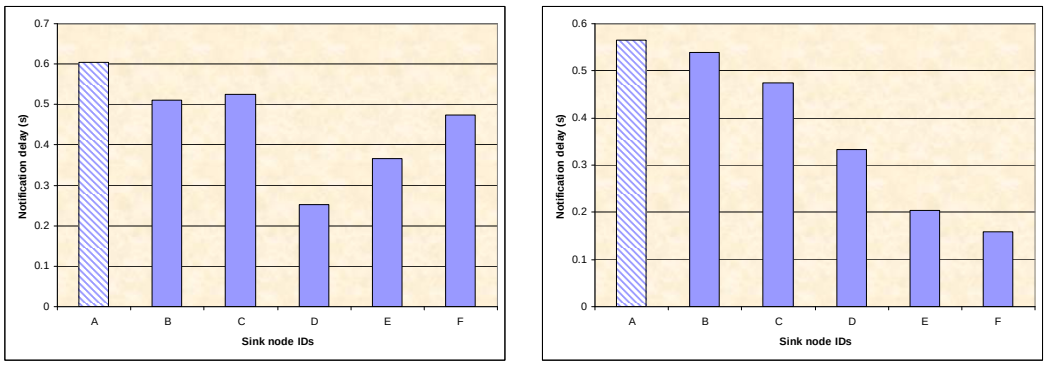
In the simulation experiments, we had the Sink A located at the right top corner of the network area that has subscribed to every event, whereas the alternative Sinks B, C, D, E and F were randomly selected among the other nodes in the network. This implied that the paths traversed by a subscription message from Sink A to the source node(s) could likely have the maximum number of hops. The graphs in Figures 4.15 - 4.17 show the average notification delays on the available paths from source nodes to different Sinks for 25-node networks. A similar trend in the results was observed for 50-, 75- and 100-node networks.

Each of the graphs corresponds to a different percentage of source nodes and the number of sensed events corresponds to the percentage of source nodes. QoS feature is extended by means of considering the available Sinks for notification messages' delivery. The least cost path is selected for delivering high-priority messages to one of those Sinks.

We selected simulation experiments such that the Sink that has subscribed to an event would likely have a maximum average delay in order to demonstrate the possibility and advantage of selecting an alternative Sink for message delivery. In the graphs below, the bar with the patterned fill corresponds to the Sink A that subscribed to the events.

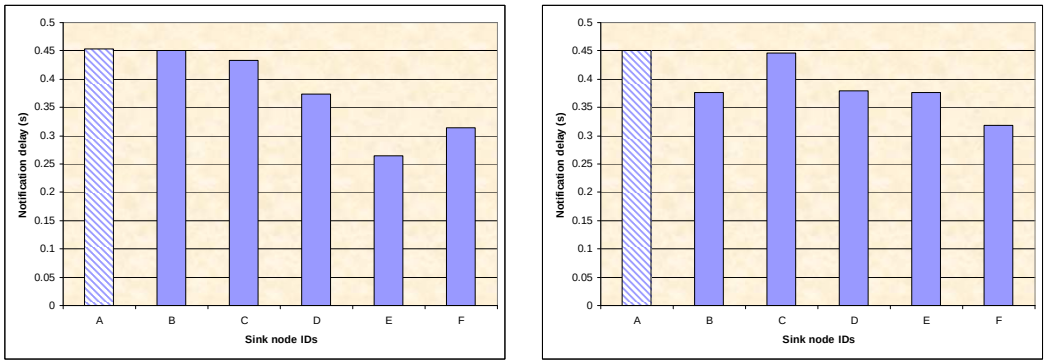
Figure 4.18 demonstrates the reduction in average notification delay when the number of nodes varies in our set of experiments. The improvement in the notification delay reaches from 60.3% to 77.4% for 25-node networks; from 60.4% to 77.6% for 50-node networks; from 60.1% to 79.9% for 75-node networks and from 47.6% to 68.7% for 100-node networks. In the graph, the average reduction of delay is presented.

The simulation experiments have shown that due to the proposed approach of considering multiple Sinks for event delivery, the algorithm has great potential of reducing the average event notification delay and thus ensuring a prompt response from monitoring centre Sink.



(a) **(b)**

Figure 4.15: Notification delay to Sinks with (a)12% and (b)30% source nodes.



(a) **(b)**

Figure 4.16: Notification delay to Sinks with (a)50% and (b)75% source nodes.

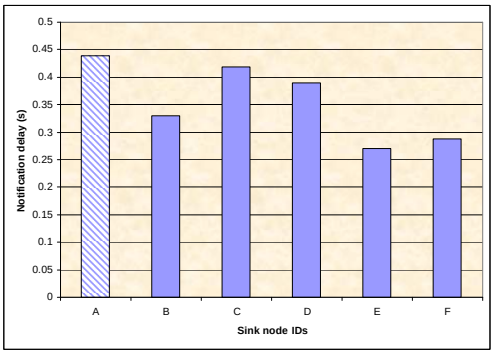


Figure 4.17: Notification delay to Sinks with 100% source nodes.

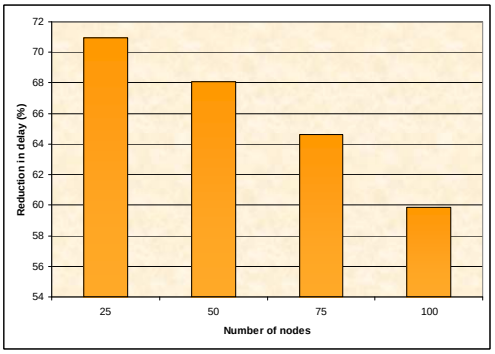


Figure 4.18: Reduction in average notification delay in multiple Sinks scenario.

4.6.6 Summary

In this chapter, we have presented a clustering routing protocol ICE for event driven, query-based and periodic WSNs. The protocol emphasizes short range transmissions by using nearest neighbors among the clusters to preserve energy. It also aims at even energy dissipation among the nodes; it makes use of alternative routes to the Sink. Multi-path routing aims at balancing the load on sensor nodes and increases the network

lifetime. It also provides fault tolerance. At the same time, the algorithm aims at avoiding congested paths. The algorithm provides QoS by prioritizing event notifications and selecting the path with the least cost in terms of low delay and high residual energy for event notifications with high priority. The algorithm provides network connectivity by ensuring that, at any moment in time, neighboring clusters can communicate (provided that NNs are within each other's transmission ranges). The algorithm also performs in-network data aggregation to further reduce energy consumption.

Due to the proposed extensions to the routing protocol, it is capable of handling multiple events' propagation through a WSN that is monitored by multiple Sinks. The first objective of the protocol was to decrease the overall latency of the multiple events' propagation. To achieve this goal, the proposed extensions to the routing protocol took advantage of the multiple paths available toward Sinks that are created during the propagation of interest subscriptions. The second objective was to provide QoS in the presence of multiple events. The protocol proposed having a Distributed DBMS over the multiple Sinks in order to select the least cost path from all of the paths available, including paths to Sinks other than the Sink that has subscribed to the event of interest. An event notification with a high priority was then forwarded along that path. Once the event notification was received at a Sink, it will be available to every Sink monitoring the WSN, including the one that subscribed to the event. This will also reduce duplicated event notifications that would need to be sent toward all the Sinks with the same interests. Also, the reliability of the routing protocol is increased due to application of this scheme. If path(s) toward the Sink that has subscribed to an event have faulty nodes, an alternative path toward one of the other Sinks will be used for the message's delivery. Thus, the overall responsiveness of the network is increased.

Experimental results have shown that ICE continues to provide a high success rate as the percentage of source nodes and the number of nodes grow. The algorithm benefits from the proactively stored paths from source nodes toward the Sink. Comparison with CPEQ clustering algorithm has shown that ICE is capable of achieving a 20% reduction

in energy dissipation. It also provides a lower delay of event notification messages with a high priority, as compared with CPEQ. Due to the QoS feature of the algorithm, a considerable reduction in the delay of event notifications with a high priority is achieved. Extensions of ICE considering multiple Sinks are capable of reducing the average notification delay and thus further improving QoS feature of the algorithm.

Chapter 5

Lightweight Iterative Positioning in WSNs

In this chapter, we present our proposed Lightweight Iterative Positioning (LIP) algorithm for WSNs, which consists of two main phases - initial position estimation and iterative refinement. The objective of the algorithm is to reduce the overhead of DV-hop based localization algorithms without significantly affecting their accuracy. It is achieved in a number of ways in both phases of the algorithm as described below. While the proposed lightweight solution to node localization is beneficial for any WSN application, it is well suited especially for the applications that require repeated localization of nodes during WSN's operation (such as military troop's relocation).

In the first phase of the algorithm, the nodes collect information about the coordinates of beacon nodes and by using that information come to an initial, rough estimate of their positions. In the second phase, the nodes iteratively refine their position estimates, after having learned the position estimates of their immediate neighbors and measured ranges between the nodes. Unlike iterative location algorithm [93] (overviewed in Chapter 2), our algorithm approaches each node individually. That is, not all nodes enter and exit the phases of the algorithm at the same time. The two phases of the algorithm may run in the network in parallel. While some nodes are still collecting the information about

beacons, the other ones that were successful in finding that information faster, will start the refinement phase of the algorithm.

5.1 Initial Position Estimation

The purpose of the first phase of the algorithm is for every node to obtain an initial rough position estimate. The proposed algorithm attempts to decrease communication overhead of DV-hop based algorithms by randomizing the process of propagation of beacons' position information to the nodes in the network by using a random Time-To-Live (TTL) - the number of hops the message propagates - for the most of the messages with beacons' position information.

Similarly to DV-hop [77] and Hop-Terrain [93], our algorithm starts with the beacon nodes flooding the network with their position information. However, only a fraction of beacons floods the network, while the remainder of the beacons use a random TTL when sending a message with their coordinates. Basically, we employ a limited flood for the majority of beacons. The number of beacon nodes is set to 5% of the nodes in the network, which was reported to be sufficient in the previous work [93], [66].

The method that we use for each beacon node to decide whether or not it is going to flood the network or use the limited flood, is similar to the distributed way that nodes are selected to be cluster heads in LEACH routing protocol [53]. Depending on the number of beacon nodes in the network, the algorithm will select a third of the beacon nodes to flood the whole network (having a minimum of three/four beacon nodes to flood the network), while the rest will be using random TTLs to disseminate their coordinates. Figure 5.1 demonstrates the process of flooding the WSN with complete and limited to the TTL floods. Beacons flood the network with their coordinates. The limited floods are initiated by most of the beacons (represented by a black triangle), they propagate the nodes inside the corresponding curved line, whereas the complete floods by the other beacons (represented by a white triangle) propagate the whole network.

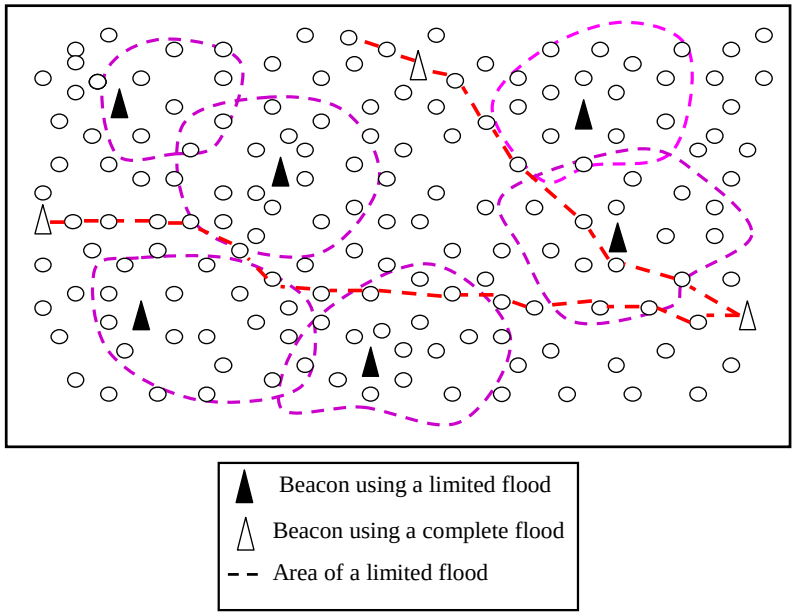


Figure 5.1: Beacons use complete and limited floods to propagate their coordinates.

While a beacon message traverses the network, the number of hops it has gone through is accumulated, stored at the nodes and added to the message. If a node has received a message about the same beacon, it will store the minimum number of hops to the corresponding beacon. As in RPA algorithm, a node broadcasts the information about beacon nodes if it has not yet done so, or if the hop count is smaller than the one sent before. In other words, a controlled flooding is used.

Savarese *et al.* [93] make a simplifying assumption that "message loss or corruption does not occur and that each message is delivered at the neighbors within a fixed radio range from a sending node". However, when large-scale networks are targeted and the algorithm uses floods extensively, this assumption is not realistic. Even using 5% beacon nodes in a 400-node network, implies that 20 beacon nodes will have to first flood the network with their position estimates and then with the average hop distances. Also, having every node go through the separate phases of the algorithm at the same time,

implies that collisions are very likely to occur.

Any beacon node that receives a message flooded by another beacon and thus learns of the coordinates of the other beacon, will calculate the average hop distance and send the message containing the average hop distance to the network. The average hop distance is calculated according to the equation 5.1. The calculation is based on the coordinates of two beacons - $Beacon_i$ and $Beacon_j$ - and the number of hops h_{ij} that the beacon message traversed from one beacon to the other.

$$average\_hop \approx \sum_{j \neq i} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} / \sum_{j \neq i} h_{ij} \tag{5.1}$$

It may happen that not all nodes will have the required information about three/four beacon nodes at the time when the message containing the average hop distance arrives at the nodes. We propose to relax the constraint of DV-hop algorithm about having the three stages of node localization non-overlapping [77]. Thus, in our algorithm a beacon node broadcasts an average hop shortly after it has obtained coordinates of another beacon. In fact, the message containing average hop distance being broadcast by a beacon, will be used to collect the information about the known beacon's coordinates: when the message propagates the nodes on the path from $Beacon_1$ to $Beacon_2$, the information about three closest beacons (the ones that can be reached in a fewer number of hops) that the nodes are aware of is piggybacked to the message.

Once a node has received the information about three/four beacons and the average hop size, it performs trilateration and obtains its position estimate. The trilateration procedure is described in section 2.3.3 and is depicted in Figure 2.8. The unknown node U needs to know the coordinates of the beacon nodes as well as the distances to the beacon nodes in order to be able to perform the trilateration procedure as shown below. The distances to the beacon nodes are calculated based on the number of hops and the average hop distance. Algorithms 6 and 7 present high-level pseudocodes of the *Initial Position Estimation* phase of the algorithm - the phase of beacon nodes flooding WSN with their coordinates and that of the propagation of the message containing average

hop distance.

Algorithm 6 A HIGH LEVEL PSEUDOCODE OF BEACON NODES FLOODING WSN WITH THEIR COORDINATES.

```
1: if node status is a beacon then
     {A beacon decides on the scope of its message.}
2:   if The beacon is going to use a complete flood then
   Action:    Broadcast beacon's coordinate and hop count of 0.
3:   else
       {The beacon is going to use a limited flood.}
   Action:    Broadcast beacons coordinates, random TTL and hop count of 0.
4:   end if
5: else
     {The receiver is not a beacon, but an ordinary node.}
6:   if received a message containing beacon's coordinates then
7:     if The node has not yet received a message about this beacon OR hop count contained in the message for this
       beacon is smaller than the one stored at the node then
   Action:      Increment the current hop count.
8:       if received a beacon message with a complete flood then
   Action:        Store the rank of the beacon (current hop count corresponding to this beacon).
   Action:        Broadcast the message containing the beacon's coordinates and current hop count.
9:       else
           {The node received a beacon message with a limited flood.}
10:        if current hop count is less than or equal to the TTL in the message then
   Action:          Store the rank of the beacon (current hop count corresponding to this beacon).
   Action:          Broadcast the message containing the beacon's coordinates, current hop count and the TTL.
11:        end if
12:      end if
13:    else
   Action:      Do nothing.
14:    end if
15:  end if
16: end if
```

Moreover, if a node was already successful in obtaining the initial position estimate, it attaches the message with the information about its initial position. Thus, when a receiver node gets the message, it is able (if needed) to retrieve that information and learn about beacons. The receiver node also receives the initial position of the neighboring node.

This will speed up the process of learning of the beacons and thus estimating nodes'

Algorithm 7 A HIGH LEVEL PSEUDOCODE OF THE PROPAGATION OF THE MESSAGE CONTAINING AVERAGE HOP DISTANCE.

1: **if** node status is a beacon **then**
{Upon receiving a message originated at a different beacon node.}
Action: Calculate average hop distance.
Action: Decide on the scope of the message containing the average hop distance.
2: **if** the beacon is going to use a complete flood **then**
Action: Broadcast a message containing the average hop distance and coordinates of the known beacon node(s).
3: **else**
{This case corresponds to a limited flood of the message containing the average hop distance.}
Action: Broadcast a message containing the average hop distance, hop count of 0, random TTL of the message and coordinates of the known beacon node(s).
4: **end if**
5: **else**
{The receiver is not a beacon, but an ordinary node.}
6: **if** the message uses a complete flood **then**
Action: Store the average hop distance.
7: **if** the number of known beacons is less than (the dimension of space + 1) **then**
Action: Extract beacon information from the message; look for higher ranked beacons in the message.
8: **else**
Action: Trilaterate and store the position estimate.
Action: Broadcast the message containing average hop distance, coordinates of the three known beacons and the initial position estimate.
9: **end if**
10: **else**
Action: Increment the message hop count.
11: **if** the incremented message hop count is less than or equal to TTL **then**
Action: Store the average hop distance.
12: **if** the number of known beacons is less than (the dimension of space + 1) **then**
Action: Extract beacon information from the message; look for higher ranked beacons in the message.
13: **else**
Action: Trilaterate and store the position estimate.
Action: Broadcast a message containing average hop distance, incremented message hop count, random TTL of the message, coordinates of the known beacon node(s) and the initial position estimate.
14: **end if**
15: **end if**
16: **end if**
17: **end if**

positions. Basically, it is a flood that is used to do more than one jobs: disseminate the average hop distance; propagate the beacon coordinate information stored at the nodes; and propagate initial position estimations (if any).

Every beacon information received at the nodes will be ranked according to the number of hops it took the message to reach the node in question. The unknown nodes will have received the beacon messages that were flooded (with a complete flood) by one third of the beacons. Having the beacon messages, containing average hop distance collect the information about the beacons, may help the nodes to select the beacons with a higher rank (fewer number of hops) from them to consider these beacons when performing a trilateration. It is desirable for the nodes to have information about the beacons that are relatively close to the nodes. Localization error accumulates with every hop and thus, considering the beacons that are located closer (can be reached in fewer hops from the node in question) will yield a more accurate position estimate as opposed to the beacons located further away from the node.

Also, when a beacon broadcasts an average hop distance, it piggybacks the message with the information about the beacon nodes that it has learned about. If a node receives messages containing different average hop sizes (it may happen because different beacons may have different distances for the average hop), then the node will compute an average of the distances to obtain an average hop distance.

When beacon nodes broadcast an average hop distance message, they do not flood the whole network with the messages, similarly to the initial floods of the algorithm. Again, a portion of the beacons floods the whole network, while the remainder of the beacons uses a limited flood with a randomly selected TTL of the message. Depending on the TTL of the message, it will either be forwarded to the neighboring nodes as defined by the TTL or will be flooded further with the adjusted average hop size.

5.2 Iterative Refinement

Iterative Refinement phase has the goal of refining the positioning information, given the rough position estimates obtained by the nodes in the initial phase of the algorithm. The refinement is achieved by using one of the ranging techniques to estimate the ranges between the nodes, such as RSSI, AoA, ToA, TDoA, discussed in section 2.3.2. The proposed algorithm does not directly depend on the technique used for range measurement, similarly to previous iterative approaches. However, each one of the techniques will affect localization accuracy and cost by introducing errors and computational cost inherent to it.

Every node associates a confidence level with its position estimate. The nodes that are close to beacon nodes will set their confidence level to a higher value (in the range from 0 to 1) compared to the nodes which are further away. Also, when nodes start the refinement phase, their confidence levels will change depending on whether or not a lateration procedure was successful every time they perform it.

In this phase of the algorithm, we propose to introduce random waiting times for the nodes, before they refine their position estimates based on the position information of neighboring nodes and the distance measurements to them, obtained by means of one of the above-mentioned range-measurement techniques. Every node will have a randomly chosen "listening" timer. While the timer is pending, a node will listen to the position correction messages coming from its neighbouring nodes. It may happen that a node hears a few position estimate messages (along with the corresponding confidence levels) coming from the same neighbouring node during that time.

Then, Weighted Moving Average (WMA) that is weighted according to the confidence levels of the position estimates is used to emphasize the estimates with higher confidences. In equation 5.2, $p_M ... p$ are the multiple position estimates of a node, and $C_M, ..., C_1$ are decreasing weights (corresponding confidence levels). WMA is calculated for both X and Y coordinates of the neighbouring nodes.

$$WMA_M = (C_M p_M + C_{M-1} p_{M-1} + ... + C_2 p_2 + C_1 p_1)/(C_M + C_{M-1} + ... + C_2 + C_1) \quad (5.2)$$

After the timer expires, the node performs a lateration procedure (either trilateration or multilateration, depending on the number of neighboring nodes) and broadcasts its corrected position estimate. Thus, in the proposed approach, a lateration is not performed every time a node hears a position correction from its neighboring nodes.Rather, the node collects the information and calculates a WMA on the position estimates of the neighboring node, from which there were received multiple position corrections. Then, a miltilateration is performed taking into account the calculated position estimate of the node, from which more than one position estimates were received.

In addition, in the proposed algorithm, at the beginning of each round (after the first round) of the iterative refinement phase, if a node has a confidence below a certain threshold, it sets a timer to use low-duty cycling. After the timer expires, the node wakes up and listens to its neighbors' position updates. If the node succeeds in performing a lateration, or if some of its neighbors' confidence levels have increased while the node was using low-duty cycling, the node's confidence level will eventually increase after performing a successful lateration. Depending on the value of the confidence level, the node will decide on using low-duty cycling again.

By putting the nodes with a very low confidence to sleep, the localization process should not be greatly affected. The thing is that when neighbouring nodes receive position updates from their neighbors, they will disregard position estimates of the nodes with low confidences anyway. We propose to conserve energy by putting such nodes to sleep until more successful nodes obtain position estimates with a higher confidence. The information will be then used by the disadvantages nodes to perform multilateration once they wake up. Thus, the goal of the proposed scheme is to first rely on the nodes that are more confident in their position estimates to settle on their position estimates. Also, the rationale behind this is to ensure that the nodes that aren't confident in their position estimates do not affect the multilateration results of the neighbouring

nodes with their potentially erroneous position estimates. Meanwhile, the nodes using low duty cycling will conserve energy and avoid computation. Algorithm 8 presents a high-level pseudocode of the Iterative Refinement phase of LIP.

Algorithm 8 A HIGH LEVEL PSEUDOCODE OF ITERATIVE REFINEMENT PHASE.

{In the Refinement phase}
Action: Set a random listening timer.
1: **while** timer pending **do**
2: **if** received a position update from the same neighbouring node **then**
Action: Store the position estimate of the node and the confidence level in an array.
3: **else**
Action: Store the position estimate of the node and the confidence level.
4: **end if**
5: **end while**
6: **if** there are multiple position estimates received from the same neighbouring node **then**
Action: Calculate Weighted Moving Average on the position estimates.
7: **end if**
Action: Multilaterate.
8: **if** performed a successful multilateration procedure AND the change in the position estimate is greater than a certain threshold **then**
Action: Increase the confidence in the position estimate.
Action: Broadcast a refined position estimate together with the confidence.
9: **else**
10: **if** the confidence is lower than a certain threshold **then**
Action: Set a timer for using low duty cycling.
11: **end if**
12: **end if**

5.3 Properties of the Algorithm

In this section we discuss main properties of the Lightweight Iterative Positioning algorithm such as communication cost, energy efficiency, and computation and accuracy.

5.3.1 Communication

- *The proposed algorithm reduces the total number of complete floods performed by beacon nodes in the phases of initial position estimation and refinement by two*

thirds as compared with previous DV-hop based approaches.

Let the number of floods in DV-hop based algorithms be B (which corresponds to the number of beacon nodes). Let the number of nodes in the network be N, then the number of messages will be NlogB (due to the controlled flooding, communication cost is logarithmic).

In the proposed algorithm, let the number of beacon nodes that used a limited flood be (2/3)B and the number of nodes that they propagate be N_l, then the number of messages during the floods will be Nlog(B/3) + $N_l$log(2B/3). Moreover, due to de-synchronizing the phases of the algorithm (the initial position estimation and the refinement) for all the nodes in the network, the number of messages propagating the network will further be reduced. As mentioned above, the algorithm employs piggybacking for disseminating the information about an average hop distance due to relaxing the constraint of having the phases of the algorithm non-overlapping. This also results in a reduced number of messages propagating the network.

- *When a beacon's message, containing an average hop distance is piggybacked with beacon information, the algorithm preserves the constraint of storing shortest paths from beacon nodes to ensure conformity with the way that is used to estimate hop distance.*

  Let us consider the case depicted in Figure 5.2, where node A has received beacon information from the beacon node B_2 due to a limited flood. Let us also assume that beacon node B_1 sends a message, containing an average hop distance with a complete flood. A message containing average hop distance is sent by the beacon node B_1, which is piggybacked with beacon information. It may reach the node A via different paths, with a different resulting hop count. Node A stores minimal hop count corresponding to the beacon B_1.

  When the message from beacon node B_1 traverses the nodes on the way to the node A, the nodes piggyback the message with the information about the beacons

that are known to them. In the proposed algorithm, while the message is flooded, the same constraint about the nodes keeping the shortest path from beacons is preserved. Thus, the resulting hop number for the message containing the information about the beacon B_1, that is stored at the node A is minimal.

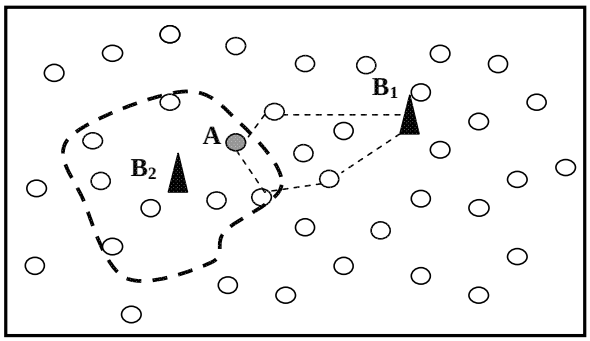


Figure 5.2: Node A stores minimal hop count to B_1 when gets a piggybacked message.

5.3.2 Energy-efficiency

- *The nodes that use low duty cycling for the duration of the set timers will conserve energy that is equal to at least* $E_{conserved}$.

  Let N_{sl} be the number of nodes that have a confidence level lower than a certain threshold value C_{thresh}. Let the duration of low duty cycling timer for node i be Tsl_i and the number of nodes using low duty cycling be N_{sl}. Then, the energy that the nodes will have saved during the sleeping mode will be at least $E_{conserved}$ as shown in equation 5.3. The calculation considers energy dissipation of a node in the idle state. However, it is possible that a node could also receive or send messages if it weren't using low duty cycling. In this case, the energy conserved by the nodes in sleeping mode will be greater than in the case of being idle.

$$E_{conserved} \geq E_{idle} \sum_{i=1}^{N_{sl}} Tsl_i, \tag{5.3}$$

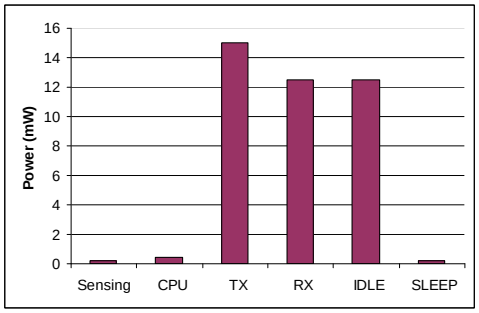


Figure 5.3: A sensor node's energy consumption, as reported in [40].

where E_{idle} is the energy dissipated in the idle state, which will be conserved by a node using low duty cycling in a unit of time.

The other way, in which the proposed algorithm provides energy-efficiency is that the messages about the average hop distances, instead of being flooded to the network, may be piggybacked with the information about known beacons and the initial position estimates if nodes along the path that the message follows already have the information. Transmitting and receiving are among the most power-consuming operations for a sensor node (as shown in Figure 5.3, where TX and RX are transmission and reception powers, respectively), thus, energy is preserved on eliminated transmissions. The overhead of piggybacking the information to the messages is reported to be incomparably lower than the cost of sending and receiving messages [40], [42].

5.3.3 Computation and Accuracy

- *A node that has hears multiple position updates from a neighboring node, while the listening timer is pending, instead of performing multilateration every time the position update is received, will compute WMA according to the confidence levels of the position estimates before performing a multilateration.*

  Let node N that set its listening timer T in the Refinement phase of the algorithm have one of the neighboring nodes refine its position estimate during that time m

times. Instead of performing m laterations, as it would happen in previous iterative algorithms, the node will calculate WMA on the position estimates of that node and then perform a lateration procedure. Thus, computation cost will be reduced. The latency of the localization process will be reduced as well, as most of the time is spent on lateration procedures as discussed in previous work. WMA, on the other hand, has been used by the research community for node localization to filter outliers in order to deal with inaccurate range data [81]. Certainly, WMA does not replace lateration procedure but is used to emphasize the position estimates with higher confidences before a lateration is performed.

- *Due to using WMA, the error accumulation is reduced in the Refinement phase.*

  By emphasizing the position estimates with high confidences, the erroneous position estimates are filtered, what, in turn, affects the accumulated position error in the network. As has been reported by Savarese *et al.* [93], errors accumulate and propagate the network very quickly. If a network has a diameter d, then an error that is introduced by some node in step n will indirectly affect each node in the network by step $n+d$ due to the sequence of a message hopping from a node to node and trilaterations performed at every node.

5.4 Performance Evaluation

We conducted simulation experiments in network simulator ns-2 in order to evaluate performance of our algorithm. We experimented with different size networks up to 400 nodes. The transmission radius of the nodes was set to 25m. The nodes were uniformly distributed in a grid. The fraction of beacon nodes was randomly selected (from 5% to 12% of the total number of nodes). Simulation time was set to 1500s. The simulation area was ranging from 30400m^2 to 108300m^2. Similar to [93], the position errors were normalized to the transmission range (i.e., 25% of position error corresponds to a quarter of the transmission range of the nodes' radio). We compared performance of LIP with

RPA localization algorithm. The experiments were executed 33 times with a confidence interval of 95%.

The metrics that we used in our experiments were:

- *number of messages in initial phase of localization*;
- *average localization error versus percentage of beacon nodes*;
- *average localization error versus network size*;
- *average localization error versus RSSI error*;
- *energy consumption during a single iteration of Refinement.*

In order to demonstrate the gain of LIP in terms of the numbers of messages in the Initial Position Estimation phase of the algorithm, we show the number of messages in this phase in both LIP and RPA in Figure 5.4. We also show the results of DV-hop's performance. As can be observed from the graph, DV-hop performs significantly worse than both RPA and LIP. We can observe that in LIP the number of messages in the first phase is consistently lower than that in RPA. Thus, LIP achieves its primary objective with regards to reducing communication cost of RPA. It is necessary to note that similar savings in terms of the number of messages sent are achieved during the average hop size's dissemination in the first phase of LIP as discussed above.

Figure 5.5 shows the position error in 400-node networks when the percentage of beacon nodes varied from 5% to 12%. As we can observe from the graph, the error is decreasing as the percentage of beacon nodes grows. This is explained by the fact that the accumulated error in distances between the nodes and the beacons is decreased with a growing percentage of beacon nodes. The average hop distance is more accurate in this case, which results in more accurate initial position estimates before the Refinement phase of the algorithm begins. As can be seen from the graph, LIP and RPA performed similarly in terms of average node localization error. While RPA performed slightly better, the difference is not significant.

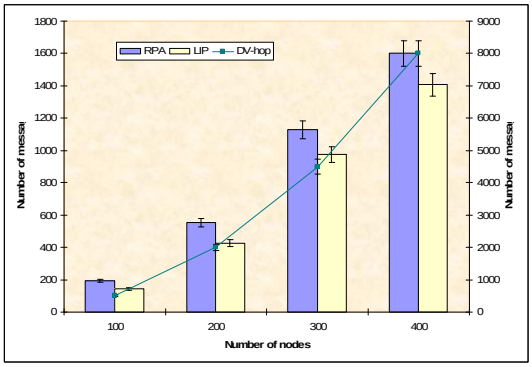


Figure 5.4: Average number of messages during beacon information dissemination.

The small difference in performance of the two algorithms with regards to the average localization error can be explained by the fact that even though the number of floods was reduced in LIP compared with RPA, other features of LIP (such as limited floods, piggybacking, random listening timers and WMA) compensated well the missing information about the beacon nodes. In Refinement phase of LIP, the employed random listening timers resulted in a reduced number of collisions of messages: the nodes broadcast their refined positions at different times, thus increasing the probability of reception of position updates of their neighbouring nodes.

WMA's property of filtering the outliers in position estimates contributed to the resulting comparable average localization error when multiple position estimates were received from a neighbouring node. The advantage of LIP is that the number of computation-expensive lateration procedures is reduced due to using WMA, whereas in RPA, every time a node receives a position update from a neighbouring node, it performs lateration. We would like to note that from our experiments we observed that in most of the cases, when during Refinement a node performed multilateration instead of trilateration, it would yield to a less accurate position estimate. Thus, in our experiments,

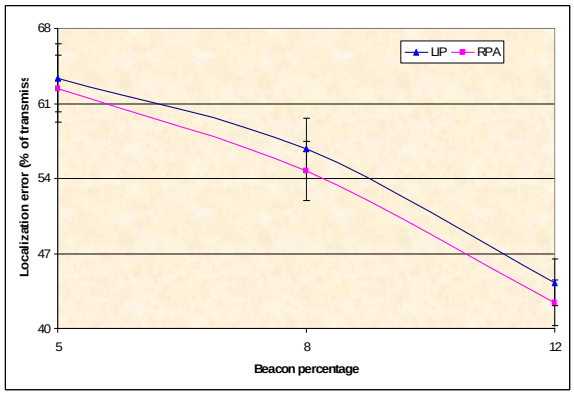


Figure 5.5: Average position error when percentage of beacons varies.

during Refinement phase of the algorithm, a node performed trilateration to the three neighbouring nodes with highest confidences in their position estimates.

The graph presented in Figure 5.6 shows that while LIP demonstrates a comparable performance in terms of average localization error, it also scales better than RPA. These results can be explained by the reduced number of floods in LIP, as well as the randomization of the Refinement phase due to employing random listening timers. For 300- and 400-node networks, the difference in average node localization error between LIP and RPA becomes even smaller than in smaller networks.

In the experiments discussed above, in Refinement phase of LIP, the RSSI error was set to 5% of the true distance between the neighbouring nodes. In order to observe the impact of RSSI error on the resulting average node localization error, in Refinement phase of the algorithm, we experimented with varying percentages of RSSI error. In our experiments, RSSI error varied from 5% to 20% of the true distance between the neighbouring nodes. Figure 5.7 shows the dependency of the average node error on the error in range measurements inherent to RSSI in 400-node networks. The resulting node error grows with RSSI's growth, as shown on the graph.

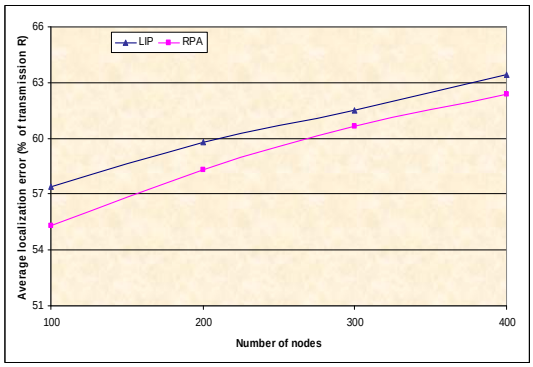


Figure 5.6: Average localization error versus network size.

In LIP, the nodes with a very low confidence in their position estimates used low-duty cycling, as described above. In order to get an insight into energy conservation feature of LIP, we conducted experiments with and without using low duty cycling in Refinement phase. Figure 5.8 shows energy consumption of LIP and RPA during one iteration of Refinement in 400-node networks. We simulated a scenario, in which every node went through one iteration of Refinement in both LIP and RPA. As can be seen from the graph, LIP achieves around 10% energy savings as compared with RPA during a single iteration of Refinement phase. It is necessary to note that the average localization error is not, however, significantly aggravated despite using low-duty cycling for the nodes with a low confidence level as seen above.

5.5 Summary

In this chapter, we presented a lightweight iterative positioning algorithm for WSNs, which consisted of two main phases such as initial position estimation and iterative refinement. The algorithm aimed at reducing the inherent overhead of DV-hop based

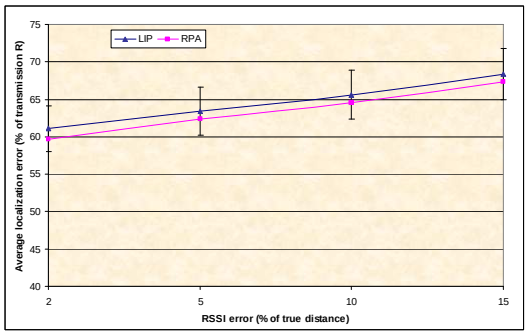


Figure 5.7: Impact of RSSI error on average node localization error.

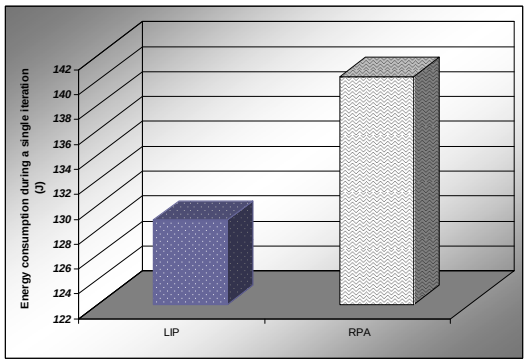


Figure 5.8: Energy consumption during a single iteration in LIP and RPA.

localization algorithms primarily in terms of the communication cost. To achieve this goal, LIP algorithm proposed to decrease the number of floods performed by the beacon nodes in the initial position estimation phase both for propagating beacons' coordinates and average hop distance. This was achieved by having the majority of the beacon nodes select random TTL for their messages' propagation, while only a fraction of the beacons utilized a complete flood.

Furthermore, both phases of the algorithm could run in the network in parallel due to eliminating the synchronism of DV-hop based algorithms. This allowed to use pig-

gybacking of messages with the information about known beacons and initial position estimation if it was available at the nodes. In the iterative refinement phase, the algorithm used random waiting times for the nodes before they were to broadcast their new position estimates with the goal of avoiding collisions and reducing the number of position correction messages. While the timers were pending, the nodes collected position corrections from neighbouring nodes and performed WMA on the position estimates of neighbouring nodes from which there were received multiple position corrections to emphasize the position estimates with high confidences. Another feature of LIP was energy-efficiency. It employed low duty cycling for the nodes that had a low confidence in their position estimates in the iterative refinemet phase. This resulted in energy conservation while the accuracy of the node localization was not compromised.

The employed random listening timers allowed to de-synchronize the nodes and aimed at reducing possible collisions and lost mesages. WMA, on the other hand, filtered out the outliers and assisted in providing a comparable with RPA localization error despite the reduced number of messages. Energy-efficient feature of LIP allowed to conserve energy. Meanwhile, the nodes with low confidences were eliminated from the process of position corrections and thus their impact on the average localization error was reduced.

From simulation experiments, we observed that LIP provides a comparable positioning accuracy with RPA, while it is capable of reducing the number of messages. In addition, LIP demonstrates around 10% energy savings in the Refinement phase of the algorithm due to employing low-duty cycling. LIP showed an improved scalability compared with RPA, for the difference in average node localization error between LIP and RPA is decreasing as the network size grows.

Chapter 6

Conclusion and Future Work

In this chapter, we first draw conclusions from this thesis. We then identify some of the research directions for future work.

6.0.1 Conclusion

In this thesis, we presented a suite of lightweight protocols for context-aware WSNs. The context comprised of location and time information about the sensed in the network events. To ensure the correct interpretation of events, we proposed a temporal event ordering algorithm, which was combined with a time synchronization algorithm suited for the clustering topology. We also introduced a lightweight node localization algorithm for WSNs. As an underlying routing protocol in the temporal algorithm, we used our proposed clustering routing algorithm. We summarize the features and experimental results of the algorithms discussed in this thesis below.

The proposed event ordering algorithm for WSANs views the temporal relationships between messages in the network as a whole by ensuring both time synchronization and temporal event ordering. Thus, we first proposed modifications to the Ordering by Confirmation event ordering protocol for WSANs, which were based on introducing a clustering concept into the network's topology with the objective of reducing the over-

head of event ordering. Second, we proposed a hybrid synchronization method that used a combination of synchronization techniques for keeping WSANs synchronized in a clustered topology. This synchronization scheme contributes to the energy efficiency of the algorithm in that it employs local time-scaled synchronization and is tailored to the clustered topology. Third, we proposed using the message exchange necessary for event ordering, as well as routing protocols for piggybacking the messages required for time synchronization. These elements contributed to further reducing the energy expenditure of the network by reducing the network traffic needed for synchronizing the network and keeping it synchronized over its time of operation.

The event ordering module of the algorithm had a goal of reducing the overall overhead of temporal ordering mechanism and it obtained the set goal: decreasing latency of event ordering process and improving its energy cost. Comparison with a periodic event ordering algorithm Heartbeat showed that COBC, being an "on demand" event ordering algorithm, outperformed the latter as it has the ability to significantly reduce energy expenditure and delay of event ordering as compared with the periodic approach to event ordering.

The combination of RBS and RTS techniques in SCT synchronization algorithm proved to be appropriate for the clustered topology. SCT's performance in both groups of simulation experiments that we conducted has demonstrated that it provides a higher synchronization accuracy while at the same time, it is not as vulnerable to the single hop synchronization error as compared with FTSP synchronization algorithm. SCT contributed from a varying percentage of CH nodes in terms of a lower average node synchronization error as the CH nodes were used as reference points.

For message delivery in the temporal algorithm, we proposed to use the presented clustering routing protocol ICE for WSNs. It used nearest neighbors between clusters as gateway nodes for communication with neighbouring clusters. It also aimed at even energy dissipation among the nodes by making use of alternative routes to the Sink. ICE's performance in our simulation experiments demonstrated that it is capable of providing

a high success rate of message delivery when the percentage of nodes sensing events and sending event notifications increases. The proactively stored multiple paths from source node to Sink enabled ICE with a low event notification delay. Comparison with CPEQ clustering algorithm has shown that ICE is capable of achieving a 20% reduction in energy dissipation. It also provides a lower delay of event notification messages with a high priority, as compared with CPEQ. QoS feature, as in the single Sink as well as in the multiple Sinks scenarios, allowed to achieve a significant reduction in notification delay of events with a high priority.

To obtain location information, we proposed the lightweight iterative positioning LIP algorithm for WSNs. During the initial position estimation, the number of floods has been reduced. In the refinement phase, LIP used random waiting times to collect information from neighboring nodes with the goal of eliminating synchronism that is present in DV-hop based algorithms and reducing the number of messages exchanged. An important feature of the proposed algorithm is that its phases may be run in parallel in the network, to utilize piggybacking of information on the messages exchanged. The proposed algorithm employed low-duty cycling for the nodes that have low confidence in their position estimates so that the nodes with a low confidence do not affect other nodes' position estimates while at the same time preserve energy. Simulation experiments showed that LIP provided a comparable positioning accuracy with RPA, while it was capable of reducing the number of messages. In addition, LIP demonstrated around 10% energy savings in the Refinement phase of the algorithm due to employing low-duty cycling. LIP showed an improved scalability compared with RPA, for the difference in average node localization error between LIP and RPA was decreasing as the network size grew.

To sum up, in order to make WSNs "intelligent", it is necessary to incorporate context into the information about the sensed in WSNs phenomena. In this thesis, we concentrated on the spatio-temporal information that constituted context. To satisfy the stringent constraints of WSNs in terms of energy, the proposed algorithms' primary objective

was energy-efficiency. First, the algorithms used low-duty cycling to conserve energy in the lightweight localization algorithm and routing algorithm. Next, nearest neighbour communication was emphasized in the routing algorithm as it is energy-efficient by ensuring short range transmissions. Moreover, we proposed to combine some components of context-awareness such as time synchronization and temporal event ordering in a single algorithm. The synchronization algorithm was tailored to the clustered topology and contributed from piggybacking synchronization pulses and replies on the messages exchanged in underlying algorithms. The algorithms proposed in this thesis used piggybacking consistently to avoid additional transmissions wherever it was possible with the goal of preserving scarce energy resources of WSNs. The algorithms also aimed at providing low latency and fault tolerance, which suggests them being suitable for emergency response class of applications and ubiquitous computing applications.

To conclude this thesis, consideration of the spatio-temporal context in WSNs enables the building of a foundation for context-aware systems. Then, the concept of context can be extended past the limits of space and time.

6.0.2 Future Work

In what follows, we will identify the directions of research that we are planning to follow in the future.

- We are planning to work on making the localization algorithm proposed in this thesis secure.
- We are planning to run the proposed in this thesis algorithms for a larger number of nodes in the simulation experiments.
- In the future work, we would like to evaluate performance of the proposed in this thesis algorithms on real WSNs by using Mica motes.
- In the event ordering algorithm for WSANs, we would like to experiment with

scenarios including multiple Actors. The algorithm could be extended to account for Actor-Actor communication, as well as Actor-Sink communication.

- In the multi-Actor scenario, we would like to examine performance of the proposed synchronization algorithm in terms of average node synchronization error.

- From the experimental setups of the routing protocol, we observed that the distribution of CH nodes is generally not even due to the distributed way of selecting CH nodes. It would be beneficial, before running the routing protocol, to have a highly uniform cluster formation. One option would be to run existing algorithms for getting a highly uniform cluster formation [26]. We would also be interested to investigate the research area of cluster formation in order to reduce the overhead of the existing solutions. It would be interesting to test performance of the routing algorithm after such an algorithm were used.

- We are going to investigate the possibility of extending the proposed routing algorithm to account for node/Sink mobility.

- We would like to compare performance of the multiple Sink extension of our proposed algorithm with existing routing protocols for multiple Sinks.

- We are going to investigate the possibility of tuning the proposed in this thesis algorithms into Vehicular Sensor Networks.

- In the future, we also intend to work on the design of a knowledge discovery algorithm for WSNs. In order to collect the information about the object being monitored that constitutes context, Knowledge Discovery/Data Mining can be applied. We propose to use Information Theory approach - Point-Wise Mutual Information (PMI) - for this purpose. In the context of WSNs, we are interested in finding patterns of sensed events/phenomena. The PMI is a powerful technique that is able to identify patterns of events that tend to co-occur as well as find events that

tend not to co-occur. This information will allow building context-aware applications that will be able to predict the evolution of events with a certain probability based on the previously processed knowledge. This in turn may lead to preventing the occurrence of critical events. Moreover, the knowledge obtained through Data Mining may be used in order to tailor an application to the particular object of monitoring. For instance, in a context-aware pervasive health monitoring system, the system will identify frequently occurring events that reflect the physical condition of a patient. It will thus be able to detect unusual development of events and respond to the condition promptly, thus preventing critical health conditions of the patient. Similarly, WSNs used for critical condition monitoring (such as gas leaks, explosions etc.) would benefit from the previously collected knowledge about the area of monitoring and respond timely to abnormal conditions and their development. Last, the context information will also assist the monitoring personnel in identifying the interests in *Publish/Subscribe* paradigm.

Bibliography

[1] K. Akkaya and M. Younis, An Energy-Aware QoS Routing Protocol for Wireless Sensor Networks, IEEE Workshop on Mobile and Wireless Networks, pp.710-715, May 2003.

[2] K Akkaya, M. Younis, Energy and QoS Aware Routing in Wireless Sensor Networks, Cluster Computing 8(2-3), 2005, pp. 179-188.

[3] K. Akkaya and M. Younis, Sink Repositioning for Enhanced Performance in Wireless Sensor Networks, Computer Networks, vol. 49/4, pp. 512-534, 2005.

[4] K. Akkaya, M. Younis, A survey of routing protocols in wireless sensor networks, Elsevier Ad Hoc Network 3/3, 2005, pp. 325-349.

[5] I. F. Akyildiz , W. Su , Y. Sankarasubramaniam , E. Cayirci, Wireless Sensor Networks: A Survey, Computer Networks: The International Journal of Computer and Telecommunications Networking, v.38 n.4, pp. 393-422, March 2002.

[6] I. F. Akyildiz, I. H. Kasimoglu, Wireless Sensor and Actor Networks: Research Challenges, Ad Hoc Networks Journal (Elsevier), Vol. 2(4), pp. 351-367, 2004.

[7] S. Alagar and S. Venkatesan, Causal Ordering in Distributed Mobile Systems. IEEE Transactions on Computers, 46(3), pp. 353-361, 1997.

[8] J. Albowitz, A. Chen, and L. Zhang, Recursive position estimation in sensor networks. In Proceedings of IEEE ICNP, pages 3541, 2001.

[9] L. Badia, M. Miozzo, M. Ross, M. Zorzi, Routing schemes in heterogeneous wireless networks based on access advertisement and backward utilities for QoS support, IEEE Communications Magazine 45(2),pp. 67-73 (2007).

[10] Bahl, P. and Padmanabhan, V. N.: Radar: An In-Building RF-Based User Location and Tracking System. In Proceedings of IEEE Infocom, 775-784 (2000).

[11] S. Bandyopadhyay and E. Coyle, An Energy-Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks, in Proceedings of IEEE INFOCOM, 2003.

[12] S. Banerjee and S. Khuller, A Clustering Scheme for Hierarchical Control in Multihop Wireless Networks, in Proceedings of IEEE INFOCOM, 2001.

[13] E. Biagioni and K. Bridges, The Application of Remote Sensor Technology to Assist the Recovery of Rare and Endangered Species, In Special Issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications, Vol. 16, No. 3, pp. 315-324, 2002.

[14] A. Boukerche, R. W. Pazzi, and R.B Araujo, Fault-tolerant wireless sensor network routing protocols for the supervision of context-aware physical environments, Journal of Parallel and Distributed Computing, Volume 66, Issue 4, Algorithms for Wireless and Ad-Hoc Networks, 586-599 (2006).

[15] A. Boukerche, H. A. B. F. Oliveira, E. F. Nakamura, and A. A. F. Loureiro. A voronoi approach for scalable and robust dv-hop localization system for sensor networks. In 16th IEEE International Conference on Computer Communications and Networks, 2007.

[16] A. Boukerche, H. A. B. F. Oliveira, E. F. Nakamura, A. Alfredo F. Loureiro: A Novel Location-Free Greedy Forward Algorithm for Wireless Sensor Networks. ICC 2008: 2096-2101.

[17] A. Boukerche, Algorithms and protocols for wireless sensor networks, Wiley Series on Parallel and Distributed Computing, 2008.

[18] A. Boukerche, H. Oliveira, E. Nakamura, A. Loureiro: Vehicular Ad Hoc Networks: A New Challenge for Localization-Based Systems. Computer Communications 31(12), pp. 2838-2849 (2008).

[19] A. Boukerche, F. H. S. Silva, R. B. Araujo, R. W. N. Pazzi, A low latency and energy aware event ordering algorithm for wireless actor and sensor networks, 8th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 111-117, 2005.

[20] A. Boukerche, I. Chatzigiannakis, S. E. Nikoletseas: A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range, Computer Communications 29(4): 477-489 (2006).

[21] A. Boukerche, R. W. Nelem Pazzi, R. Borges de Araujo, Fault-tolerant wireless sensor network routing protocols for the supervision of context-aware physical environments, Journal of Parallel and Distributed Computing (JPDC), v. 66, n. 4, pp. 586-599, (2006).

[22] A. Boukerche, Handbook of Algorithms for Wireless Networking and Mobile Computing, Chapman and Hall/CRC, 2005.

[23] G.D. Caro, M. Dorigo, AntNet: Distributed Stigmergetic Control for Communications Networks, Journal of Artificial Intelligence Res. 1998, 9, pp.317-365.

[24] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton and J. Zhao, Habitat Monitoring: Application Driver for Wireless Communications Technology, In proceedings ACM SIGCOMM Workshop Data Communication in Latin America and the Caribbean, ACM Press, pp. 20-41, 2001.

[25] H. Chan, M. Luk, A. Perrig, Using clustering information for sensor network localization. Lecture Notes in Computer Science, Springer Berlin/ Heidelberg, pp. 109-125 (2005).

[26] H. Chan and A. Perrig, ACE: An Emergent Algorithm for Highly Uniform Cluster Formation, In Proceedings of the First European Workshop on Sensor Networks (EWSN), pp. 154-171, 2004.

[27] P. Chandra and A.D. Kshemkalyani, Causal Multicast in Mobile Networks. In Proceedings of 12th IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer and Communication Systems, pp. 213-220, 2004.

[28] J. Chang and L. Tassiulas, Routing for Maximum System Lifetime in Wireless Ad-Hoc Networks, Proceedings of the 37th Annual Allerton Conference Communication, Control, and Computing, 1999.

[29] D. Chen and P.K. Varshney, QoS Support in Wireless Sensor Network: A Survey, Proceedings of the International Conference on Wireless Networks (ICWN), CSREA Press, 2004.

[30] H. Chen, K. Sezakil, P. Deng and H. C. So, An Improved DV-Hop Localization Algorithm with Reduced Node Location Error for Wireless Sensor Networks, IECE Trans. Fundamentals, vol. E91-A, number 8 (2008).

[31] D. Chen and P. K. Varshney, QoS Support in Wireless Sensor Networks: A Survey, International Conference on Wireless Networks, 2004.

[32] P. Ciciriello, L. Mottola, and G. P. Picco, Efficient Routing from Multiple Sources to Multiple Sinks in Wireless Sensor Networks, 4th European Conference on Wireless Sensor Networks, 2007.

[33] Crossbow, MicaZ Wireless Measurement System: http://www.xbow.com.

[34] A. Dey, Understanding and Using Context, Personal and Ubiquitous Computing, v.5, n.1. (2001).

[35] Y. Ding, C. Wang, and L. Xiao, A static-node assisted adaptive routing protocol in vehicular networks, In Proceedings of the Fourth ACM international Workshop on Vehicular Ad Hoc Networks VANET, pp 59-68, 2007.

[36] L. Doherty, L. El Ghaoui and K.S.J. Pister, Convex position estimation in wireless sensor networks, In Proceedings of Infocom 2001, pp. 1655-1663.

[37] R. Draves, J. Padhye, and B. Zill, Comparison of Routing Metrics for Static Multi-Hop Wireless Networks, ACM SIGCOMM, pp. 133144, 2004.

[38] J. Elson, K. Römer, Wireless sensor networks: a new regime for time synchronization, ACM SIGCOMM, Computer Communication Review, Vol. 33, No. 1, pp. 149-154, 2003.

[39] J. Elson, Time Synchronization for Wireless Sensor Networks, PhD thesis, University of California, Los Angeles, 2003.

[40] D. Estrin et al., http://nesl.ee.ucla.edu/tutorials/mobicom02.

[41] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.M. Kermarrec, The many faces of publish/subscribe. ACM Computing Surveys, 35(2), 114-131 (2003).

[42] L. M. Feeney, An energy-consumption model for performance analysis of routing protocols for mobile ad hoc networks, Mobile Networks and Applications, Vol. 3, No. 6, pp. 239-249 (2001).

[43] S.R. Gandham, M. Dawande, R. Prakash, S. Venkatesan, Energy efficient schemes for wireless sensor networks with multiple mobile base stations, Global Telecommunications Conference, pp. 377- 381 Vol.1 (2003).

[44] S. Ganeriwal, R. Kumar, M. B. Srivastava, Timing-sync Protocol for Sensor Networks, ACM SenSys 2003.

[45] H. Garcia-Molina and A. Spauster. Ordered and Reliable Multicast Communication. ACM Transactions on Computer Systems, 9(3), pp. 242-271 (1991).

[46] Girod, L. and Estrin, D.: Robust range estimation using acoustic and multimodal sensing. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1312-1320 (2001).

[47] Girod, J. L. and Estrin, D.: Fine-Grained Network Time Synchronization using Reference Broadcasts. In Proceedings of the Fifth Symposium on Operating systems Design and Implementation, 147-163 (2002).

[48] G. H. Golub and C.F. Van Loan. Matrix Computations, 3rd edition, Johns Hopkins University Press, 1996.

[49] M. Guenes, U. Sorges, I. Bouazizi, ARA-the Ant-Colony Based Routing Algorithm for MANETs, In Proceedings of International Workshop on AD Hoc Networking (WAHN 2002), pp. 79-85, 2002.

[50] J. Y. Halpern, I. Suzuki, Clock synchronization and the Power of Broadcasting, Distributed Computing, 5(2), pp. 7382, 1991.

[51] B. Han, G. Simon, Fair Capacity Sharing Among Multiple Sinks in Wireless Sensor Networks, in Proceedings of IEEE MASS Conference, pp. 19. to MASS 2007.

[52] R. Hayton, OASIS: An Open Architecture for Secure Interworking Services, PhD thesis, University of Cambridge, 1996.

[53] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, Energy-Efficient Communication Protocols for Wireless Sensor Networks. In Proceedings of the 33rd Annual International Conference on System Sciences, 3005-3014, 2000.

[54] T. He, J. Stankovic, C. Lu and T. Abdelzaher, SPEED: A stateless Protocol for Real-Time Communication in Sensor Networks, in the Proceedings of International Conference on Distributed Computing Systems, 2003.

[55] J. Hightower, G. Borriello, Location Systems for Ubiquitous Computing, Computer, v.34 n.8, p.57-66, 2001.

[56] J. Hill and D. Culler, A wireless embedded sensor architecture for system-level optimization, Technical report, University of California, Berkley, 2002.

[57] C. Imatagonwiwat, R. Govindan, and D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking, 2000.

[58] R. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, John Wiley and Sons, New York, 1991.

[59] X. Jia. A total Ordering Multicast Protocol Using Propagation Trees. IEEE Transactions on Parallel and Distributed Systems, 6(6): pp. 617-627, 1995.

[60] H. Kim, Y. Seok, N. Choi, Y. Choi and T. Kwon, Optimal Multi-sink Positioning and Energy-efficient Routing in Wireless Sensor Networks, Lecture Notes in Computer Science, Springer-Verlag, Number 3391, 2005.

[61] H. Kopetz and W. Schwabl: Global Time in Distributed Real-Time Systems. Technische Universitat Wien (1989).

[62] H. Kopetz and W. Ochsenreiter: Clock Synchronization in Distributed Real-Time Systems. IEEE Transactions on Computers, C-36(8), 933–939 (1987).

[63] D. Koutsonikolas, S.M. Das, Y.C. Hu. Path planning of mobile landmarks for localization in wireless sensor networks. Computer Commun., 30(13):25772592 (2007).

[64] A. Krohn, M. Beigl, C. Decker, T. Zimmer, TOMAC - Real-Time Message Ordering in Wireless Sensor Networks Using the MAC Layer, In Proceedings of the 2nd International Workshop on Networked Sensing Systems (INSS), 2005.

[65] L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, pp. 558-565, 1978.

[66] K. Langendoen and N. Reijers, Distributed Localization in Wireless Sensor Networks: a Quantitative Comparison, Computer Networks, vol. 43, no. 4, pp. 499-518, 2003.

[67] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson, Wireless Sensor Networks for Habitat Monitoring, In ACM International Workshop on Wireless Sensor Networks and Applications (WSNA02), pp. 88-97, 2002.

[68] A. Manjeshwar and D. P. Agrawal, TEEN: A Protocol for Enhanced Efficiency in Wireless Sensor Networks. In Proceedings of the 15th International Symposium on Parallel and Distributed Processing, pp. 2009-2015, 2001.

[69] A. Manjeshwar and D. P. Agrawal, APTEEN: A Hybrid Protocol for Efficient Routing and Comprehensive Information Retrieval in Wireless Sensor Networks, 2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, pp. 195-202, 2002.

[70] Mansouri-Samani, M., Sloman, M.: GEM A Generalized Event Monitoring Language for Distributed Systems. IEE/IOP/BCS Distributed Systems Engineering Journal, 4(25), 96-108 (1997).

[71] Maroti, M., Kusy, B., Simon, G. and Ledeszi, A.: The flooding time synchronization protocol. In ACM Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, 39-49 (2004).

[72] S. McCanne, S. Floyd, ns network simulator: http://isi.edu/nsnam/ns.

[73] Mills, D. L.: Internet Time Synchronization: The Network Time Protocol. In Zhonghua Yang and T. Anthony Marsland, editors, Global States and Time in Distributed Systems. IEEE Computer Society Press (1994).

[74] Y. H. Nam, Z. Halm, Y. J. Chee, and K. S. Park, Development of remote diagnosis system integrating digital telemetry for medicine, International Conference IEEE-EMBS, Hong Kong, 1998, pp. 11701173.

[75] M. Nekovee, Sensor networks on the road: the promises and challenges of vehicular ad hoc networks and grids, Proceedings of the Workshop on Ubiquitous Computing and e-Research, 2005.

[76] G. J. Nelson. Context-Aware and Location Systems. PhD thesis, University of Cambridge, 1998.

[77] Niculescu, D. and Nath, B.: Ad Hoc Positioning System. In Proceedings of the IEEE International Conference on Global communications (GlobeCom ' 01), 2926-2931 (2001).

[78] N. Noury, T. Herve, V. Rialle, G. Virone, E. Mercier, G. Morey, A. Moro, T. Porcheron, Monitoring behavior in home using a smart fall sensor, IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology, pp. 607610, 2000.

[79] N. Noury, T. Herve, V. Rialle, G. Virone, E. Mercier, G. Morey, A. Moro, T. Porcheron, Monitoring behavior in home using a smart fall sensor, IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology, pp. 607610, 2000.

[80] E.I. Oyman, C. Ersoy, Multiple sink network design problem in large scale wireless sensor networks. 1st International Conference on Communications, 2004.

[81] T. Parker, K. Langendoen, Refined statistic-based localization for ad-hoc sensor networks, in: 47th IEEE Global Telecommunications Conference, 2004.

[82] B. Parkinson and J. Spilker, Global Positioning System: Theory and Application (Aican Institute for Aeronautics and Astronautics, 1996).

[83] R. Pazzi, Design and Performance Evaluation of Data Gathering and Streaming Protocols for Wireless Multimedia Sensor Networks. PhD thesis, University of Ottawa, 2007.

[84] E.M. Petriu, N.D. Georganas, D.C. Petriu, D. Makrakis, V.Z. Groza, Sensor-based information appliances, IEEE Instrumentation and Measurement Magazine, pp. 3135, 2000.

[85] R. Prakash, M. Raynal and M. Singhal, An Efficient causal ordering algorithm for mobile computing environments. In Proceedings of 16th International Conference on Distributed Computing Systems, Hong Kong, pp. 744-751, 1996.

[86] Priyantha, N. B., Balakrishnan, H. Demaine, E. and Teller, S.: Mobile-Assisted Localization in Wireless Sensor Networks. In IEEE INFOCOM, pp. 172-183 (2005).

[87] S. Quaireau and P. Laumay. Ensuring Applicative Causal Ordering in Autonomous Mobile Computing. In Workshop on Middleware for Mobile Computing, 2001.

[88] M. Raynal, A. Schiper, S. Toueg, The causal ordering abstraction and a simple way to implement it, Information Processing Letters archive, Volume 39, Issue 6, pp. 343-350, 1991.

[89] K. Römer, Blum, P., Meier, L.: Time Synchronization and Calibration in Wireless Sensor Networks, In I. Stojmenovic, editor, Handbook of Sensor Networks: Algorithms and Architectures, John Wiley and Sons, 199-237 (2005).

[90] K. Römer, Temporal Message Ordering in Wireless Sensor Networks. IFIP Mediterranean Workshop on Ad-Hoc Networks, pp. 131-142, 2003.

[91] K. Römer, Time Synchronization in Ad Hoc Networks. In ACM Symposium on Mobile Adhoc Networking and Computing, pp. 0-1 (2001).

[92] S. Samarah, Knowledge Discovery for Behavioral Patterns in Wireless Sensor Networks. PhD thesis, University of Ottawa, 2008.

[93] C. Savarese, J.M. Rabaey, K. Langendoen, Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks, In Proceedings of the General Track: 2002 USENIX Annual, 317-327, 2002.

[94] C. Savarese, J. M. Rabaey, and J. Beutel. Locationing in distributed ad-hoc wireless sensor networks. In ICASSP, pp. 20372040, 2001.

[95] A. Savvides, H. Park and M. Srivastava, The n-Hop Multilateration Primitive for Node Localization Problems, Mobile Networks and Applications, vol 8, 443-451, 2003.

[96] A. Savvides, C. Han, M. B. Strivastava, Dynamic fine-grained localization in Ad-Hoc networks of sensors, In Proceedings of the 7th annual International Conference on Mobile computing and networking, 2001, pp. 166-179.

[97] B. Schilit, N. Adams, R. Want, Context-Aware Computing Applications. In Proceedings of the First International Workshop on Mobile Computing Systems and Applications, pp. 85-90 1994.

[98] A. Schiper, J. Eggli, A. Sandoz. A New algorithm to implement causal ordering. In Workshop on Distributed Algorithms, 1989, pp. 219-232.

[99] Y. Shang , W. Ruml , Y. Zhang , M. Fromherz, Localization from Connectivity in Sensor Networks, IEEE Transactions on Parallel and Distributed Systems, v.15 n.11, pp. 961-974, 2004.

[100] Y. C. Shim, C. V Ramamoorthy, Monitoring and Control of Distributed Systems. 1st International Conference of Systems Integration, pp. 672-681, 1990.

[101] Sichitiu, M. L. and Ramadurai, V.: Localization of Wireless Sensor Networks with a Mobile Beacon. Center for Advances Computing Communications, North Carolina State University, Technical Report. TR-03/06 (2003).

[102] K. Sohrabi et al., Protocols for self-organization of a wireless sensor network, IEEE Personal Communications, Vol. 7, No.5, pp. 16-27, 2000.

[103] P. Sommer , R. Wattenhofer, Gradient clock synchronization in wireless sensor networks, In Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, pp.37-48, 2009.

[104] K.-F. Ssu, C.-H. Ou and H.C. Jiau, Localization with mobile anchor points in wireless sensor networks, IEEE Transactions on Vehicular Technology, pp. 11871197, 2005.

[105] I. Stojmenovic, Handbook of Sensor Networks, Algorithms and Architectures, Wiley and Sons, 2005.

[106] Stoleru, R., Stankovic, J. A., Son, S. H.: Robust node localization for wireless sensor networks. In Proceedings of the 4th workshop on Embedded networked sensors, 2007.

[107] W. Su and I. F. Akyildiz, Time-diffusion synchronization protocol for wireless sensor networks, IEEE/ACM Transactions on Networking, vol 13, no. 2, pp.384-398, 2005.

[108] J. Van Greunen and J. Rabaey, Lightweight time synchronization for sensor networks, in ACM WSNA, 2003, pp. 1119.

[109] Z. Vincze, D. Vass, R. Vida and A. Vidacs, Adaptive Sink Mobility in Event-driven Clustered Single-hop Wireless Sensor Networks, 6th International Network Conference, 2006.

[110] G. Wang, D.Turgut, L.Boloni, Y. Ji, and D. Marinescu, Improving routing performance through m-limited forwarding in power-constrained wireless networks, Journal of Parallel and Distributed Computing (JPDC), 68(4):501-514, (2008).

[111] B. Warneke, M. Last, B. Liebowitz, K.S.J. Pister, Smart Dust: Communication with a Cubic-Millimeter Computer, IEEE Computer Magazine, Vol. 34, 44-51 (2001).

[112] Whitehouse, K., Culler, D.: Calibration as Parameter Estimation in Sensor Networks. In Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications, 59-67 (2002).

[113] B. Xiao, H. Chen, and S. Zhou, A Walking Beacon-Assisted Localization in Wireless Sensor Networks, Proceedings of the IEEE International Conference on Communications (ICC '07), 3070-3075 (2007).

[114] Y. Xu, J. Heidemann, and D. Estrin, Geography-informed energy conservation for ad hoc routing, ACM MobiCom, pp. 70-84, 2001.

[115] J. Yang, M. Xu, W. Zhao, B. Xu, A Multipath Routing Protocol Based on Clustering and Ant Colony Optimization for Wireless Sensor Networks, Sensors 2010; 10(5):4521-4540.

[116] E. Yoneki. Temporal Ordering of Wireless Sensor Events. In Ubicomp - Workshop on Ubiquitous Wireless Communication, pp. 7-8, 2005.

[117] E. Yoneki, J. Bacon, Determination of time and order for event-based middleware in mobile peer-to-peer environments. In Proceedings of the 3rd International Conference on Pervasive Computing and Communications Workshops, PerCom 2005 Workshops.