A POWER EFFICIENT DISASTER DETECTION SCHEME USING WIRELESS SENSOR AND ACTUATOR NETWORKS

by

Salim Eryiğit

B.S., Industrial Engineering, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2008

A POWER EFFICIENT DISASTER DETECTION SCHEME USING WIRELESS SENSOR AND ACTUATOR NETWORKS

APPROVED BY:

Assist. Prof. Tuna Tuğcu (Thesis Supervisor)	.....
Assoc. Prof. Necati Aras (Thesis Co-supervisor)	.....
Assoc. Prof. Fatih Alagöz	.....
Prof. Cem Ersoy	.....
Prof. Sema Oktuğ	.....

DATE OF APPROVAL: 24.04.2008

ACKNOWLEDGEMENTS

First of all, I would like to thank Tuna Tuğcu for his endless support, kindness, and patience not only as an advisor but also as a friend. His helpful comments and brilliant ideas made this true. I am looking forward to work with him in my future studies. Secondly, I would like to express my gratitude to Necati Aras, Cem Ersoy, Fatih Alagöz, and Sema Oktuğ for taking part in my jury.

I also would like to thank my colleagues for their friendship and support. I am really thankful to Mehmet not only for his support and help but also being a brother to me. In addition, I am also grateful for friendship of Erinç, Deniz, and Umut.

I especially would like to thank my darling, Özge. She has been so supportive and caring, it is hard for me to choose the right words to express my feelings.

Finally, although they are not with me now, I want to thank my parents for everything. I am really proud to be your son and I would do my best to be a son you deserve. I am sure you are happy at where you are and wish the best for me. You will always be loved.

ABSTRACT

A POWER EFFICIENT DISASTER DETECTION SCHEME USING WIRELESS SENSOR AND ACTUATOR NETWORKS

In this work, a novel scheme is designed to exploit the fact that disasters occur rarely, but the sensor network should be up and available for a long time. The proposed scheme is devised for disaster recovery for wireless sensor and actuator networks. The main goal is to minimize battery usage, increase network lifetime, and minimize redeployment. The proposed model works even in the case that the disaster destroys all nodes within the disaster area and also blocks communication unless there is line of sight. The scheme works in an area where the actuators form a regular grid and the sensors are randomly deployed. The scheme tries to maximize network lifetime by reducing idle listening and protocol overhead while there is no disaster and rapidly organizes nodes when the disaster strikes. Operation under disaster includes an alarm phase, a grouping phase, and a routing phase. Grouping mechanism provides data aggregation and reduces the overhead of the actuators. Furthermore, routing mechanism allows nodes along the path to perform duty cycling and prevent them from becoming bottlenecks. The proposed scheme is compared with the well known T-MAC algorithm in terms of network lifetime and shown to produce better results.

ÖZET

KABLOSUZ ALGILAYICI VE ERİŞİM DÜZENEĞİ AĞLARINI KULLANAN GÜÇ VERİMLİ BİR AFET SEZİMİ ŞEMASI

Bu çalışmada, afetin az görüldüğü fakat algılayıcı ağın uzun bir süre çalışır ve müsait olması gerçeğinden yararlanılarak yeni bir şema tasarlanmıştır. Önerilen şema afet sezimi amaçlı kablosuz algılayıcı ve erişim düzeneği ağları için planlanmıştır. Ana amaçlarımız pil kullanımını azaltmak, ağ ömrünü arttırmak ve tekrar konuşlanma sayısını en aza indirmektir. Önerilen şema afetin afet alanındaki tüm düğümleri bozduğu ve göz görüşü yoksa iletişimi de engellediği durumlarda bile çalışabilmektedir. Şema erişim düzeneklerinin düzenli bir ızgara oluşturduğu ve algılayıcıların rastgele dağıtıldığı bir alanda çalışmaktadır. Şema herhangi bir afet yokken aylak dinlemeyi ve protokol ek yükünü azaltarak ağ ömrünü etkili bir şekilde azami mertebeye eriştirmeye, afet vurunca da düğümleri hızlı bir şekilde düzenlemeye çalışmaktadır. Afet sırasında operasyon uyarı evresi, gruplama evresi ve rota tespit etme evresini içerir. Gruplama mekanizması veri kümelemeye ve erişim düzeneklerinin ek yüklerinin azalmasına olanak sağlar. Ayrıca, rota tespit mekanizması yol üzerindeki düğümlerin görev döngüsü yapmasına izin verir ve onların darboğaz olmalarını engeller. Önerilen şema, iyi bilinen T-MAC algoritması ile ağ ömrü tabaninda karşılaştırılmış ve daha iyi sonuçlar ürettiği gözlenmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACK	Acknowledgement
ACW	Adaptive Contention Window
CH	Cluster Head
CRDP	Cluster Radius Decision Point
CTS	Clear to Send
DC	Duty Cycle
DCF	Distributed Coordination Function
DD	Delay Diameter
E-TDMA	Energy-Efficient TDMA
FCS	Forwarding Candidate Set
FLAMA	Flow Aware Medium Access
GPS	Global Positioning System
LEACH	Low Energy Adaptive Clustering Hierarchy
MAC	Medium Access Control
NEAN	North-East Actuator Node
NLP	Nonlinear Programming
NWAN	North-West Actuator Node
PEDD	Power Efficient Disaster Detection
RA	Random Access
ROI	Region of Interest
RTS	Ready to Send
SEAN	South-East Actuator Node
SWAN	South-West Actuator Node
TDMA	Time Division Multiple Access
TRAMA	Traffic Adaptive Medium Access
VC	Vector Computation
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network

1. INTRODUCTION

Wireless sensors are battery operated low cost devices with limited processing and communication capabilities. A wireless sensor mainly consists of four components: sensing unit, processing unit, power unit, and transceiver. They differ from traditional sensors since they can process the sensed data and communicate through wireless channel. The phenomenon to be sensed can be ambient temperature, humidity, motion, seismic activity, etc.

Large number of these nodes are deployed to form a WSN (Wireless Sensor Network). In a WSN, sensor nodes sense the environment for some sort of activity and report the collected data to a sink node. Sink nodes have more processing power than an ordinary sensor and are usually assumed to have unlimited power. A WSN should have at least one sink.

A WSAN (Wireless Sensor and Actuator network) is a WSN that contains actuator nodes besides sensor nodes and sink. Actuator nodes have more processing power and more power than an ordinary sensor. Moreover, they can communicate to much longer distances than regular sensor nodes. An actuator node costs less than a sink and may be used for coordination in WSNs that are deployed for monitoring a large area in a cost effective way.

A WSN can operate in different ways depending on the context it is used for. An application may require constant monitoring in which nodes should report collected data to the sink periodically. Some WSNs operate in a query based fashion where sink propagates a query into the network and some relevant sensors report back to the sink. Another type of WSN operation is for event detection. In this type, collected data is usually irrelevant to the sink unless some sort of event occurs.

In this thesis, a new scheme, Power Efficient Disaster Detection (PEDD), is proposed for detecting rarely occurring events such as forest fires. PEDD includes

network deployment and initialization methods in addition to energy efficient operation before and during disaster. Group formation is postponed till the disaster strikes to reduce energy consumption. When a fire starts, the network reports an alarm about the fire to the sink within a reasonable amount of time in addition to periodic measurement reports subsequently. Furthermore, we want to extend the lifetime of the network to minimize cost and harm to the environment due to the batteries in the sensor nodes.

Most of proposed protocols for WSNs assume that communication unit of nodes are always on. However, it is known that idle listening (listening the channel for possible communication) consumes nearly the same energy as reception. Therefore, in order to obtain a network with a longer lifetime, sensors' transceiver units should be turned off to save energy. The trade-off for this action is increased packet latency. Even if transceiver unit is shut down, sensing and processing units still consume power. To have a long lived network, one should also shut down those units whenever possible, putting nodes into sleep mode in which they consume negligibly small amount of energy.

Event detection using WSNs has been studied in the literature. In these works events are assumed to have small effect area and sensor nodes can operate as usual during an event. However, an event like fire that can cover a large area may destroy the sensor circuitry and make them inoperable if they are within the area enclosed by fire. Moreover, it can affect the communication channel (depending on the fire) so that two nodes separated by fire may not be able to communicate although they are within each other's transmission range.

The contribution of this thesis is the introduction of a complete solution for disaster recovery with the following features:

- Communication, and thus, power expenditure is kept at minimum as long as there is no observed disaster to prolong network lifetime.
- In the case of a disaster, an alarm is produced and transmitted over several paths to guarantee early warning.

- Group formation is delayed until the disaster and applied only in the area of disaster.
- A new routing algorithm is proposed with alternating paths to avoid routing failures due to battery exhaustion on the shortest path.

The proposed scheme has three operation modes:

1. normal operation of the network without any event,
2. reporting the phenomenon and group formation for reporting data in case of an event,
3. routing collected data to actuator nodes.

The organization of the thesis is as follows. In Chapter 2, background information about WSNs are given together with related work from literature. Since PEDD covers different aspects that are usually studied separately, a detailed literature survey is performed. Chapter 3 explains the proposed scheme in detail. Simulation parameters, scenarios and results are given in Chapter 4. Finally, we conclude the thesis and explore future study areas in Chapter 5.

2. BACKGROUND INFORMATION

In this section, we first discuss the general structure of a WSN and then provide detailed information and proposed solutions for sleep scheduling, clustering, and routing problems in the literature.

2.1. General Information

A WSN is composed of a large number of connected sensor nodes with sensing, processing, and communication capabilities [2]. Nodes are battery operated, tiny, and inexpensive devices with limited communication range. A node may be specialized in just sensing a single phenomenon or it may sense a couple of different aspects of the environment including but not limited to temperature, humidity, seismic activity, intrusion, etc. A WSN is deployed in an ad-hoc manner and sensors are usually placed randomly in ROI (Region of Interest). A WSN does not require a fixed communication infrastructure so it can easily be deployed.

In addition, ROI is usually a remote and/or hostile area so after initial deployment nodes should organize themselves as a network. After deployment, nodes monitor the data and report back to sink. Since ROI may be huge, there can be thousands of sensors. To ease up communication and coordination, actuator nodes with more processing capability and power may also be deployed.

WSNs are used in both civil and military applications. For civil applications, the usage may range from monitoring a patient's health to monitoring a habitat. On the other hand, for military applications, the aim may be border surveillance or target tracking.

As sensors have limited power supply, their lifetime is also limited. Since they are deployed in a remote area, accessing the ROI after deployment may not be an easy task. Moreover, since a WSN consists of a large number of sensors, recharging

or replacing the battery of each sensor is not a feasible solution. When designing a protocol, these inherent disadvantages should always be considered and the protocol should try to maximize network lifetime.

In a sensor unit, transceiver circuitry expends most of the energy compared to other units. Some sources of energy wastage due to communication can be listed as:

- idle listening where a node listens the channel for possible communication,
- packet overhearing where a node hears a packet while it is not the intended receiver,
- protocol overhead due to message exchanges required by the protocol employed,
- packet collisions when two transmitters transmit a packet at the same time without knowing each other's transmission and a receiver that hears both of them.

First two cases can be avoided by putting radio to sleep mode whenever possible. For the third case, protocol must be designed carefully in order not to induce too much overhead. Finally, for the last case, some clever medium access schemes should be employed to avoid collisions.

Sensors have a limited communication range, they may not be able to reach to the sink directly. Even if they could, transmitting a packet to a long distance costs more energy than transmitting it in hops. To increase network lifetime, one should use multihop communication whenever possible to reduce the energy expenditure. Another aspect of multihop communication is that it causes a couple of nodes to expend less energy rather than just one node to use too much battery power, providing a little degree of evenness in consumed energy.

Another difficulty of WSN comes from the fact that nodes have limited processing power and memory. There is no overall coordination of the network and global knowledge of the network at any given time cannot be obtained without incurring a huge communication cost. Even if this is tolerable, the state of the network may change until all the information is collected at the sink due to delay. That is to say, a protocol

designed for WSNs should be distributed and each node should take its own actions depending its current state, its neighbors, and environment to be monitored. Moreover, due to large number of nodes, the protocol should be scalable.

When the purpose of WSN is event detection, nodes do not need to report periodically to sink unless the event of interest has occurred. It is obvious that the traffic load is very low in this case. However, when the event occurs, especially the nodes close the event need to report periodically. The type of event is very important in the design of a protocol. Depending on the event, protocol may need to satisfy some strict requirements. For instance, if radioactive material leakage is to be detected from a reactor, even seconds may be important. On the other hand, if the event to be detected is forest fire where the response time is rather long, delay requirements may not be that strict. In this thesis, we focus on the second type of events where delay is important but not that crucial.

Duty cycling is a commonly used term in WSNs to refer to the case when nodes switch between *on* and *off* states to save energy. A DC (Duty Cycle) with a value 0.01 means that a node is in the *on* state for one unit of time and in the *off* state for 99 units of time. In the *on* state, all units of a node are on whereas, depending on the protocol, the *off* state may refer to the case where just the transceiver circuitry is off or all units are off. A low DC leads to a longer lifetime for the network but it may also cause unacceptable delays. Therefore, depending on the context both of these metrics should be balanced carefully.

2.2. Related Work

Sleep scheduling in WSNs can be defined as putting nodes into sleep mode whenever possible to save energy in an organized manner such that some latency requirements are met.

A new channel is introduced for pipelined tone wakeup in [3]. Wakeup radio is on for T_{dtone} and off for T_{sleep} where $T_{dtone} + T_{sleep} = T$. When a node has a

packet to send, it sends a tone to wakeup channel that lasts for T_p. Wakeup tone does not contain the identity of the receiver so any node with active wakeup channel and within transmission range switches to data channel. T_p is set to $T + T_{dtone}$ so that all neighbors become awake. Transmitter first notifies the receiver so other neighbors go back to sleep whereas receiver begins to wakeup its neighbors to minimize delay. Similar wakeup radio schemes are also used in [4], [5] and [6]. The problem with these schemes is that a sensor with an extra wakeup channel is not feasible to manufacture both industrially and cost wise. In addition, since wakeup radio is assumed to be low power, its transmission range would be shorter than the data channel so a transmitter may not be able to reach all its neighbors on the wakeup channel.

A sleep scheduling scheme for high density cluster-based WSNs is proposed in [7]. It is assumed that clusters are already formed, nodes have variable transmission ranges, and each member node knows its distance from its CH (Cluster Head). Three different sleeping mechanisms are discussed.

- Randomized scheduling: Each member node goes to sleep with the same probability, β_s.
- Distance-based scheduling: β_s is a function of the distance from CH. This scheme favors the nodes far from CH to sleep more since they use more energy to communicate with CH.
- Balanced-energy scheduling: This is a special case of distance-based scheduling where expected energy usage does not depend on the distance from CH by carefully selecting β_s function.

It should be noted that in this work, energy expenditure due to idle listening is not considered.

A delay efficient sleep scheduling scheme is discussed in [8]. They assumed perfect node synchronization and low traffic load such that no interference or collisions occur. The number of slots is fixed and slot length is defined as the time required to successfully receive a packet. They tried to minimize DD (Delay Diameter) that is defined as the

maximum delay among the shortest path for all pairs. A node is wake $1/k$ of the time slots. In their central algorithm, all nodes start with the same slot for wakeup and then each node picks up the slot that minimizes DD in turn. Algorithm is run until a finishing time. In their distributed algorithm, each node only knows the slot assignments of its neighbors and selects a slot that minimizes DD based on that knowledge. If node's current slot is updated, it informs its neighbors who in turn run the algorithm again. This algorithm is run at each node for a predefined number of iterations.

Exponentially distributed node sleep times and per hop delays are assumed in [9]. Each transmitting node appends a timestamp to its packet that shows how long it has been waiting. Per hop delay constraint $\{\tau, p_r\}$ is met when per hop delay is less than τ with probability p_r. After waking up nodes estimate the ongoing activity based on the timestamps of waiting packets and update their DC accordingly.

Another distributed algorithm is proposed in [10]. Slot length is again the time required for successful reception of a packet and all nodes are synchronized. Another assumption is that a node can only receive in its active time whereas it can transmit at any time. A stochastic distributed algorithm where each node only knows the schedules of its neighbors is used with different objectives that can be listed as:

- min-max: minimize the maximum delay to its neighbors,
- min-max-degree: minimize the delay to the neighbor with the highest degree (number of neighbors),
- min-avg: minimize the average delay to all its neighbors,
- min-avg-degree: minimize the weighted average delay where weights are the degrees of its neighbors.

A protocol for HWSNs is introduced in [11] and [12]. Geographic routing with FCSs (Forwarding Candidate Sets) is used. A node's FCS is the set of neighbors that are closer to sink than itself. A random asynchronous wakeup scheme is discussed in [12] where each node in FCS wakes up one in every slot, stay awake for a predetermined period of time, and go to sleep again. On the other hand, active time is a function

of node's energy in [11]. Nodes exchange their energy levels by using beacons and each node knows the energy levels of its neighbors. A node's active time is given by $max(E_x w/E_{avg}, w_{th})$ where E_x is the current energy level of the node, E_{avg} is the average energy level of its neighbors, w is the normal wakeup duration, and w_{th} is the minimum wakeup duration. Since actuator nodes have higher battery capacities compared to ordinary nodes, this scheme forces them to be active more and sensor nodes sleep more often.

A pipelined mechanism is offered in [13]. A static sleep schedule is used when there is no query. If sink broadcasts a query nodes adapt to dynamic schedule. Static sleep schedule forms a pipeline based on the node's hop count from sink. Suppose that nodes A, B, and C are one, two, and three hops away from sink, respectively. Then their sleep schedule looks like the one shown in Figure 2.1. But whenever sink broadcasts a query the sleep schedule becomes dynamic. If hop count of destination from sink is known, schedule is modified to form a reverse pipeline. Otherwise nodes closer to the sink remain active until they hear from their neighbor that is further from the sink. These two cases are shown in Figure 2.2. This approach brings energy savings and with its pipeline mechanism latency is at minimum. However; it cannot be used for other usage cases.

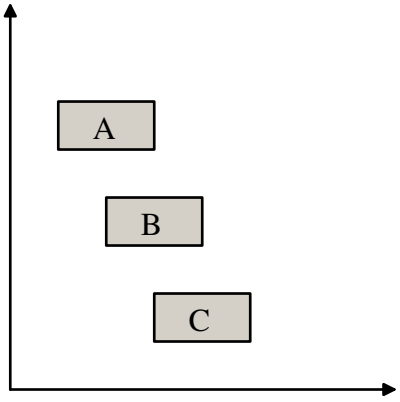
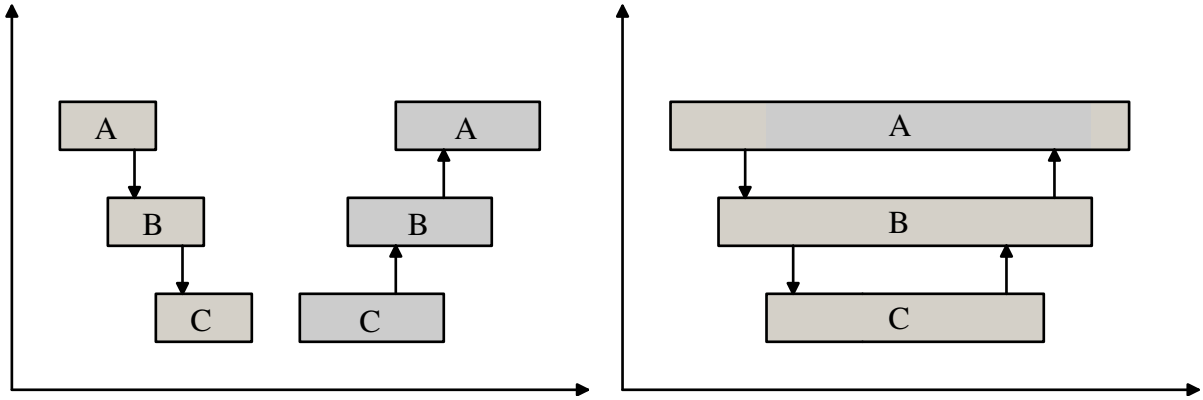


Figure 2.1. Static sleep schedule without a query

An adaptive sleeping discipline for diffusion stimulus is explained [14]. Diffusion stimulus is defined as a phenomenon originating at a source and constantly spreading outward like a leakage. Sensor nodes assumed to be equipped with GPS (Global



(a) Destination hop count known (b) Destination hop count unknown

Figure 2.2. Dynamic schedule during a query

Positioning System). Sensors can be in one of the three states, which are restricted, non-restricted, and covered. Covered state means that sensor is already inside the stimulus. Restricted state implies that although sensor is not inside the stimulus, it is close to the stimulus boundary and may become covered soon. Finally, non-restricted state means sensor is far enough from stimulus. Non-restricted nodes follow their DC as normal. Restricted ones modify it in such a way that they are awake when stimulus comes close to them. Covered nodes are always in the active state. There are four types of messages that are as follows:

- wakeup: When a non-restricted node wakes up, it broadcasts a wakeup message,
- present: A covered sensor who hears a wakeup message replies with a present message. A non-restricted node goes to restricted state upon hearing this message.
- detection: A node that detects the stimulus goes to covered state and broadcasts a detection message. Non-restricted nodes who hear this message go to restricted state.
- elapsed: Covered nodes that hear the detection message reply back with elapsed message that contains the elapsed time between their detection time and current time.

By making use of the elapsed messages and GPS coordinates, nodes estimate local velocity of the stimulus and arrival time. Based on these values, restricted nodes wake up more often, update their velocity and arrival time estimates.

A sleep scheduling scheme for rare event detection is proposed in [15]. Network is assumed to be consisting of two layers, primary nodes and secondary nodes. Primary nodes are selected in such a way that they maintain sensing coverage with minimum number of nodes. They are rotated periodically. On the other hand, secondary nodes have a very small DC. The proposed algorithm assumes synchronized nodes and concentrates only on how to rotate the primary nodes. The algorithm is as follows:

Step 1 : Each primary node selects a wake up time randomly,

Step 2 : Each primary node recalculates its wakeup time, exactly once in each iteration based on recent neighbor schedules. A node continues to iterate until it does not receive updates from neighbors and thus has not changed its wakeup time. When a node updates its wakeup time, it broadcasts this information to inform its neighbors.

The problem with this scheme is large overhead for communication during each iteration.

A sleeping discipline for data gathering is offered in [16]. The network is assumed to be like a tree where sink is at the root. Nodes start to send to their neighbors at the upper level when they wake up. DC of a node consists of time for reception, time for transmission, and time for sleep. Nodes transmit their messages just at the time when their parents wakeup for reception, which is illustrated in Figure 2.3. Additional mechanisms are employed to improve the algorithm. When a node has more data to send, is sets up a more data flag in MAC header to inform the receiver. Receiver stays awake for an additional slot, which improves latency. Sibling nodes at the same level may race for channel, a parent that receives a packet from one of its children stays awake for another time slot in case another children may send a packet to prevent starvation.

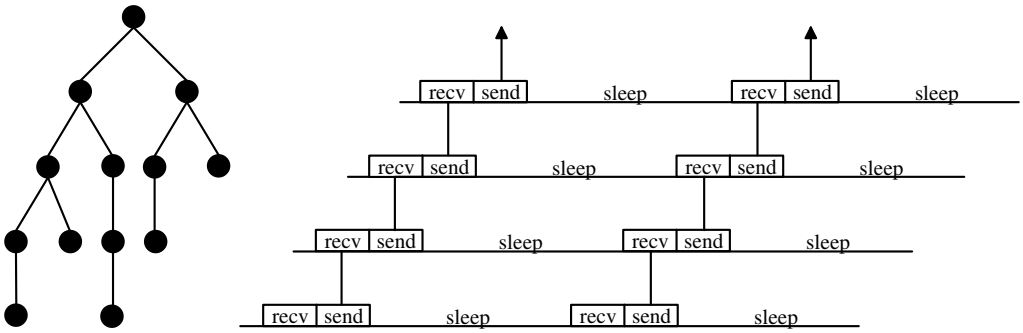


Figure 2.3. Pipelined wakeup scheme for data gathering

S-MAC protocol is introduced in [17]. Nodes exchange schedules and sleep and listen at the same time. A node receiving multiple different schedules adapt to both of them. This scheme provides virtual clustering among nodes. Slot length is fixed. IEEE 802.11 DCF (Distributed Coordination Function) is used and nodes overhearing RTS (Ready to Send) or CTS (Clear to Send) packets go to sleep mode during data packet transmission. Message passing ,where long messages are fragmented into small fragments and transmitted in bursts, is used. RTS and CTS messages are exchanged just for the first fragment, but all fragments require ACK (Acknowledgment) messages. Each fragment and ACK have duration field to prevent hidden terminal problem. The problem with S-MAC is that since nodes form virtual clusters, a node receiving a packet has to wait until the next slot to transmit it to its neighbor. At each hop, a latency equal to slot length is incurred.

T-MAC also uses IEEE 802.11 DCF for communication and schedule exchange mechanism for virtual clustering [18]. Unlike S-MAC active period of a node is adaptive. Active period of a node ends when no activation event occurs for a time period of TA. Activation events can be listed as:

- firing of a periodic timer,
- data reception,
- sensing of communication (collision),
- ending of a packet transmission,
- overhearing a data exchange by a neighbor.

To prevent asymmetric communication problem (also called the early sleeping problem) where a future receiver goes to sleep without knowing it is the receiver of packet in the near future two workouts are explained. First one makes use of a so called Future RTS packet. Node receiving the Future RTS packet do not go to sleep, which reduces latency. Second workout is taking priority on full buffers. A node with a full buffer prefers transmitting to receiving. To achieve this task, whenever it hears a RTS packet destined for itself, instead of replying with a CTS, it immediately sends its CTS.

Another MAC Protocol, TEA-MAC is introduced in [19]. TEA-MAC employs short but frequent sleeps with simplified IEEE 802.11 control frames. There are sleep counters and wakeup counters that operate antagonistically. That is to say, whenever sleep counter reaches zero, node wakes up and whenever wakeup counter reaches zero, node sleeps. Depending on the packet a sensor receives or transmits, its wakeup counter may be reset to a new value. For instance, when a node receives a packet, its wakeup counter is reset to the value of the packet duration plus processing delay and sleep counter is set to zero. It is argued that when the value shared by both counters is small, nodes wakeup frequently, which provides higher throughput. Since wakeup time is small, idle listening is reduced. Moreover, since nodes in the same locality share the same counter values, they sleep and wake at the same time, which achieves virtual clustering. However, the cost between switching between sleep and active states is not considered in this work and can become significant if such state change occurs frequently.

Schedule based access schemes are proposed in [20] and [21]. TRAMA (Traffic Adaptive Medium Access) protocol assumes time slotted channel and offers a distributed election mechanism based on schedules [20]. It consist of the following three components:

- Neighborhood Protocol : Nodes obtain two hop neighborhood information.
- Schedule Exchange Protocol : Schedule information is established based on the traffic of the receiver.

- Adaptive Election Algorithm : A hash function is used to select the transmitter for the current slot.

Moreover, some mechanisms for channel reuse when the selected transmitter has no data to send are employed to select a new transmitter.

FLAMA (Flow Aware Medium Access) is very similar to TRAMA but this time instead of traffic, flows are used to select the transmitter for a particular slot [21]. FLAMA is designed for query based WSNs whereas TRAMA is designed for a general case.

Clustering protocols group nodes into clusters and selects one of them as the cluster head. Member nodes usually just communicate with CH (cluster head). CH is responsible for aggregating gathered data and sending it to the sink. With clustering hierarchy, some benefits for WSNs can be obtained that can be listed as follows:

- TDMA scheme is used for member nodes to access the CH. With its inherent duty cycling, TDMA provides energy savings.
- CHs are responsible for coordination and management of member nodes, which reduces the burden of the sink and also the burden of the network as a whole.
- It is assumed that different clusters use different codes for communication and TDMA scheme is used within a cluster. Hence, interference is reduced.

Besides these advantages of clustering, the main disadvantage is that CHs consume more energy than member nodes since they are active all the time. To prevent CHs to die quickly, they need to rotated periodically, which induces significant protocol overhead.

LEACH (Low Energy Adaptive Clustering Hierarchy) is introduced in [22]. In LEACH, single hop architecture is assumed, where CHs can reach the sink in one hop. All non-CH nodes transmit their data to CH, which aggregates the data and transmits it to sink. Operation of LEACH is divided into rounds. Each round begins with a setup

phase in which, the clusters are organized, followed by a steady-state phase where data is transferred from member nodes to CH and from there to sink.

In the setup phase, each node elects itself to be a CH with probability $P_i(t)$, which is chosen such that the total number of clusters is equal to a predetermined value, k. That is

$$\sum_{i=1}^{N} P_i(t) = k$$

where N is the total number of nodes. To ensure all nodes become CH the same number of times requires each node to be a CH once in N/k rounds on the average. Let, $C_i(t)$ be the indicator function determining whether or not node i has been a CH in most recent ($r \ mod(N/k)$) rounds ($C_i(t) = 0$ if node i has become a CH, 1 otherwise), then each node chooses to become a CH at round r with probability:

$$P_i(t) = \begin{cases} \dfrac{k}{N - k * (r \ mod(\frac{N}{k}))} & \text{if } C_i(t) = 1 \\ 0 & \text{if } C_i(t) = 0. \end{cases}$$

After CH selection step, elected CHs broadcast an advertisement message. Non-CH nodes receiving these messages select which cluster to join and transmit a join-request message. Then, CHs set up a TDMA schedule and transmits the schedule to the members of the cluster. After schedule is known to all nodes in the cluster, steady state transmission begins. During steady state transmission phase, each node uses power control and adjusts its power just enough to reach the CH. Moreover, radio of each node except CH is turned off until its allocated transmission time.

A refined version of LEACH is presented in [23]. The aim of refinement is to achieve better evenness of energy load distribution over the whole network. To accomplish this goal, remaining energy levels of candidates and their cost for transmission is considered.

The main idea of the algorithm is to avoid selecting nodes with lower residual energy and higher energy dissipation as cluster heads. Modified threshold function becomes:

$$P_i(t) = \frac{k}{N - k * \left(r \; mod \left(\frac{N}{k}\right)\right)} \frac{E_{r\_residual} - E_{r\_dissipate}}{E_{r\_average} - E_{r\_dissipate}}$$

where $E_{r\_residual}$ is the node's initial residual energy in last round r, $E_{r\_average}$ is the average initial energy of all nodes in the cluster in round r and $E_{r\_dissipate}$ is the total energy dissipated by node for data transmission in last round r.

Another modification of LEACH is introduced in [24]. The main difference is that cluster sizes are more predictable than LEACH since in this protocol each elected CH broadcast the advertisement message within a predefined range. This predefined range may be calculated by some theoretical analysis for dense networks.

An energy-efficient version of LEACH for event-driven applications is presented in [25]. This time member nodes use E-TDMA (Energy-Efficient TDMA) to communicate with the CH. This time when a member node wakes up, it does not send any data to CH, if it has no significant data to send. This scheme provides major energy savings and thus, an increased lifetime.

Shortcomings of LEACH are discussed and an ACW (Adaptive Contention Window) based algorithm is proposed in [26]. Disadvantages of LEACH can be listed as:

- unsuccessful probability of head election,
- probability of inaccurate number of elected heads,
- probability of sufficient separation distance.

In the proposed algorithm, each node selects a random backoff value from the contention window (between 0 and CW) and the one with the minimum backoff value becomes the CH in its region.

CW value of each node is modified based on one of the three schemes below:

- Long-term fairness based: Initial value of CW is the number of nodes in the network and it is not modified. In the long run, each node becomes CH approximately the same number of times.
- Medium-term fairness based: In this scheme, CW value for each node is modified at each round based on the formula:

$$CW_{i,r} = \frac{\sum_{j=1}^{N} E_{j,r}}{E_{i,r}}$$

  where $CW_{i,r}$ denotes the CW value and $E_{i,r}$ denotes current remaining energy for sensor i in round r. Though it is impossible to know each node's energy, the overall energy of the network (nominator) is estimated by using the energy level of the cluster.
- Short-term fairness based: In this scheme, all non-CH nodes decrease their CW value by one in each round whereas CH nodes update their CW value as infinity so they do not become CHs for the next N rounds.

An iterative clustering protocol, HEED is proposed in [27]. HEED is designed to meet the following requirements:

- whole process is distributed,
- clustering terminates within a fixed number of iterations,
- at the end of a clustering period, each node is either a CH or a member of exactly one cluster,
- clustering is efficient (low processing complexity and message exchange),
- CHs are well-distributed over the network and have high average residual energy compared to other nodes.

Before the first iteration, each node sets a probability of becoming a CH that is given by:

$$CH_{prob} = C_{prob} \frac{E_{residual}}{E_{max}}$$

where C_{prob} is the desired percentage of CHs among all nodes, $E_{residual}$ is the remaining energy of the node, and E_{max} is the energy of a fully charged battery. During any iteration i, $i \leq N_{iter}$, where N_{iter} is the maximum number of iterations (predefined), every non-CH node elects to become a CH with probability CH_{prob}. After step i, the set of tentative CHs -S_{ch}- is set to $\{$ CHs after step $i-1 \bigcup$ new heads selected at step i $\}$. A node, s_i, selects its CH to be the node with the lowest cost in S_{ch} (S_{ch} may include s_i). Every node than doubles its CH_{prob} and goes to the next step.

If a node elects to become a CH, it broadcasts an announcement message containing its ID and status. In this message status is set to tentative CH if its CH_{prob} is less than 1, or final CH if its CH_{prob} has reached 1. A node is said to be covered if it hears an announcement message, otherwise it is uncovered. An uncovered node, announces itself as a CH with state as final CH. A tentative CH can become a regular node at a later iteration if it finds a lower cost CH.

Though HEED performs better than LEACH in terms of network lifetime, its iterational nature brings considerable delays and overhead to form clusters.

A modification of HEED is proposed in [28]. In the first phase, each node checks if its cost is the least among its neighbors. If it has the lowest cost, it sets itself as core head, otherwise it sets the least cost neighbor as core.

After core election, CH selection excludes non-core nodes. Every node has an initial probability CH_{prob}. During each iteration, each uncovered node becomes a tentative CH with a probability of CH_{prob}, and resolves conflicts with neighbor tentative CHs by using residual energy level as criteria. All nodes double their CH_{prob} for each iteration. The election process repeats until the values for CH_{prob} of all nodes reach 1.

In the final round, final CHs are considered as CHs and tentative CHs as non-CHs. Uncovered nodes run the algorithm to elect extra CHs.

A sink coordinated algorithm is presented in [29]. It is assumed that sink knows location of all nodes in the network. During each setup phase, sink gathers information from all sensors about their current energy. Then, sink computes the average node energy of the network and chooses a subset -S- of nodes whose energy values are above average as tentative CHs.

The steps of the algorithm are as follows:

1. From S, select two nodes s_1 and s_2 that have maximum separation distance,
2. Group remaining nodes with either s_1 or s_2, whichever is closest,
3. Balance the two groups so that they have approximately same number of nodes,
4. Split S into two sets S_1 and S_2.

The algorithm is applied recursively until the desired number of clusters is obtained. After clusters are formed, sink chooses the lowest-energy routing path and forwards this information to sensor nodes together with the details on clusters and CHs. The network uses multihop architecture.

A clustering protocol for HWSNs is offered in [30]. The algorithm uses the initial and residual energy of the nodes to select CHs. To avoid global knowledge of the network by each node, the algorithm estimates the average energy of the network by using the initial energy of the network and energy spent for each round. The general structure of the algorithm is very similar to LEACH but this time probability of becoming a CH differs for ordinary and advanced nodes based on their initial energy levels.

An unequal clustering mechanism for multihop communication is presented in [31]. The logic behind the idea is that nodes closer to the sink become hot spots and die quickly. Each node has a competition range for becoming a CH that decreases as

a node's distance to the sink decreases. The result is clusters closer to the sink have smaller cluster sizes.

In the network deployment stage, sink broadcasts a *hello* message at a certain power level. By hearing this message, each node computes its approximate distance to the base station based on received signal strength. For clustering phase, first several tentative CHs are selected. Every node becomes a tentative CH with same probability T that is predefined. Other nodes keep sleeping. For a tentative CH s_i, its competition range is defined as:

$$s_i.R_{comp} = \left(1 - c\frac{d_{max} - d(s_i, BS)}{d_{max} - d_{min}}\right) R^0_{comp}$$

where d_{max} and d_{min} denote the maximum and minimum distances between sink and nodes in the network, $d(s_i, BS)$ is the distance between s_i and sink, c is a constant, and R^0_{comp} is the maximum competition range.

For selecting CHs, each node broadcasts a *compete head* message that contains its power level and competition radius with power level just enough to achieve a radius of R^0_{comp}. Each tentative CH forms a list of adjacent CHs, S_{ch}, that are closer than its competition radius. After receiving all compete head messages, s_i becomes a CH if it has the most residual energy among all nodes in S_{ch}. In this case it broadcasts a *final head* message, otherwise it broadcasts a quit election message. Selected CHs broadcast advertisement messages. And other nodes join the closest clusters based on received signal strength and broadcast a join cluster message.

A similar unequal clustering mechanism for single hop WSNs ,where each CH reaches sink directly, is discussed in [32]. This time clusters far from the sink should have smaller sizes, since they consume more energy reaching to sink, than clusters nearby the sink. The algorithm is similar to the one in [31], but this time after CHs are selected, each node decides its cluster based on a cost function that is a weighted sum of the distance of CH to the sink and distance of node to the CH.

After observing that some of the coverage ranges can be fully covered by other sensors in the area, it is proposed that coverage based scheduling methods may not work well for heterogeneous environments where monitored measure changes quickly in some sub-regions but slowly on others [33]. Moreover, it may be the case that readings of two neighboring sensors may be significantly different due to a boundary (e.g., one is under the sun and the other is in the shadow).

The proposed approach is to cluster the sensors based on accumulated data. Then only some of the sensors from a cluster are active in a round and others are asleep. Of course, active sensors in a cluster change in turn. The logic behind this approach is that since these sensors have similar readings now, they will also have similar readings in the future. With this approach, the authors tried to achieve energy saving and quality guarantee. Even if a reading of a sensor changes, it is still detected with a minor delay.

The whole protocol is built on top of the sink. Sensors only have a simple scheduler module that simply extracts the schedule obtained from the sink and make the sensor active or asleep based on the schedule. On the other hand, the sink has the following modules:

- Data storage module: It stores all sampling data received form sensors.
- Dissimilarity measure module: It calculates the pair wise dissimilarity measure of time series for each sensor. Note that dissimilarity measure is application specific.
- Clustering module: With given dissimilarity and a maximal dissimilarity threshold, this module divides the sensors into clusters.
- Sensor node working schedule generator: It generates a working schedule for each sensor node based on the clusters.

In the implementation, single-hop architecture is assumed, that is, all sensors can directly reach the sink. Another assumption is that, sensor nodes are time synchronized. Moreover, they make only one sensor active in each cluster at any time.

Clustering problem has also attracted interest for mobile adhoc networks and metaheuristics are used for the problem. A simulated-annealing based technique is proposed in [34]. A weighted metric that uses the weighted sum of a node's

- sum of distances of members of the cluster,
- average speed of the nodes in the cluster,
- accumulative time of that node being the cluster head,
- degree (number of neighbors) difference.

After introducing the metric, simulated annealing is applied to find a solution to the problem. The main drawback of the approach is that a neighborhood scheme is not specified, in each iteration a random solution is produced and tested for acceptance criteria. The problem with the algorithm is that, it is central, not distributed. The same metric is also used in [35] where genetic algorithm is used for optimization.

Two budget based algorithms ,where nodes have local growth budgets to allocate for neighbors, are offered in [36]. The first algorithm is called *Rapid* and produces of clusters of bounded size whereas the other one called *Persistent* always tries to produce a cluster with specified size.

For *Rapid* algorithm, the initiator is assigned a budget of B, of which it accounts for itself and evenly distributes $B - 1$ among its neighbors by sending a message to each of them. An arbitrary set is chosen if there are more neighbors than the budget. The neighbors receive the message account for themselves and distribute the remaining among all their neighbors except the parent. The messages propagate until the budget is exhausted. Each node that receives the message sends an acknowledgment to its parent when either the budget is exhausted or it has received acknowledgments from all its children. The algorithm terminates when the initiator receives acknowledgments from all neighbors that it sent message.

Persistent algorithm is similar to *Rapid* algorithm. But in this case, each node that receives the message does not send acknowledgments to its parent immediately

or receiving acknowledgments from its children. It computes the size of the subtree and compares it to the budget allocated to it. It distributes the shortfall among its neighbors that either was not explored previously or met all previously allocated budgets. When either the budget is met or when further growth is not possible, it returns an acknowledgment to its parent. The algorithm terminates when the initiator determines that the bound has met or when no further growth is possible (there are not enough nodes around to join the cluster).

The problem with both algorithms is that initiators need to be spaced apart both in time and space to avoid collusions. An exponentially distributed timer is used to minimize collisions.

CuMPE protocol consisting of routing part and MAC part is introduced in [37]. Routing part uses the selective flooding to propagate messages and to find upstream node with lowest cost, and a greedy algorithm to choose the minimum cost path. MAC part finds the critical path in the tree and schedules the nodes to send messages without extending the tree. CuMPE has four steps, which are CH election, route setup, TDMA scheduling, and data transfer. For CH election, existing algorithms such as LEACH are used.

In route setup, protocol finds the path to CH with minimum cost. Instead of flooding the same information through network, protocol only propagates new information when a node finds a lower cost path. In this phase, elected CHs first broadcast advertisement messages. Any node receiving the advertisement knows the ID of the CH and signal strength of the packet that is used for calculating the cost to CH. The node then broadcasts a message to tell other nodes about the cost information. Any node receiving the message knows the ID of the sender, cost from sender to CH and cost to sender. Then it calculates cost to the CH by adding the two cost values, it compares this value with its current cost. If new cost is higher, it is discarded, otherwise sender node becomes the new upstream node and new information about cost is also broadcasted.

After clusters and routes are set up, TDMA schedule is used to ensure collision free transmission. The protocol first selects the node with greatest number of downstream and blocking nodes and schedules them as early as possible in order to minimize overall transmission delay. A node X is said to block node Y if and only if transmission range of X covers the upstream node of Y. When X sends data, it also reaches the upstream node of Y and if Y also transmits at the same time, their packets collide at the upstream node of Y. To build the TDMA schedule, each sensor sends its downstream node list and downstream-blocking node list. Then sink, creates the schedule and broadcasts it to all nodes. To construct the two lists, each node sends a message containing a type field that is set to upstream and an *ID* field that is set to the *ID* of its upstream node. A node receiving the message puts that node to its downstream node list if the *ID* field matches its own *ID* or downstream-blocking node list otherwise.

Balancing the energy consumption throughout the network ,which also leads to increasing coverage time of the network, is also an important criterion. It does not necessarily mean minimizing consumed total energy. To tackle this problem, a signomial optimization problem is formulated and solved using generalized geometric programming for two different approaches in [38]. The first approach is routing-aware optimal cluster planning, in which clustering is performed for shortest hop count inter cluster routing. The second one is clustering aware optimal random delay, in which routing is done for load-balance cluster topologies. In this approach, a CH probabilistically decides either to relay the traffic to the next hop CH or deliver it directly to the sink.

Another method for balancing the energy consumption is considered as balancing the size of the clusters [39]. In the initial phase of the protocol, nodes adjust their power in such a way that its number of neighboring nodes is about a predefined threshold. That is to say, nodes that have more neighbors than threshold shrink their transmission power with certain power level in each step until the desired number of neighbors is achieved. The procedure operates the opposite way around for nodes that have fewer neighbors than the threshold. The aim of this setup phase is to balance the energy expenditure of each node.

For cluster formation, an existing protocol like LEACH can be used. After partitioning of network into clusters, all nodes within a cluster send their energy level to the CH. Then, CH selects the node with the maximum power as the new CH and announces this change. Although this scheme provides balanced sized clusters, it induces excessive overhead.

Cluster radius configuration is also studied in [40]. In this work, CHs are assumed superior to ordinary sensors with high energy capacity and more processing capability. Moreover, it is also assumed that sink knows the position of all CHs. The protocol is able to regulate the cluster radius for energy savings.

In initial phase, CHs deployed at random construct a triangle to determine a CRDP (Cluster Radius Decision Point). The distance between CRDP and each point is used to estimate the radius of the cluster. Delaunay triangulation is used for constructing an approximate equilateral triangle consisting of three CHs. The construction of an equilateral triangle leads to balanced energy consumption of CHs.

In the second phase cluster radius is controlled. The radius of each cluster constituting the triangle is controlled by using CRDP as a pivot point. Two types of algorithms are used: NLP (Nonlinear Programming) based approach and VC (Vector Computation) based approach.

Some design guidelines for cluster based WSNs are proposed in [41]. In general, the analysis focuses on communication within a cluster to decide when single hop communication should be used to reach to CH and when multiple-hop should be used for the same purpose. A hybrid approach is also proposed, in which nodes alternate between single hop and multi hop communication. The problem with single hop is that nodes far from CH consume more energy. On the other hand, with multi-hop communication, nodes nearby the CH become hot spots and die quickly.

All of these works about clustering assume constant monitoring over the ROI. An event-based clustering mechanism for HWSNs is discussed in [1]. Actuator nodes

provide communication between CH and sink. In this work, a sensor can be in one of the four states, which are given in Table 2.1. Transceiver state T, R means node can both transmit and receive, whereas R means it can only receive incoming packets.

Table 2.1. Sensor states for the event-driven clustering scheme [1]

State	**Processor**	**Memory**	**Sensor, ADC**	**Transceiver**
S_0	Active	Active	On	T, R
S_1	Active	Active	On	R
S_2	Active	Active	On	Off
S_3	Active	Sleeping	On	Off

In the initialization phase, every node is in S_1 state. Sink broadcasts a control packet that includes a threshold value (such as critical temperature) leading to the state change of sensors. A sensor receiving the value stores it and goes to S_3 state. If an event occurs, all sensors that sense the event and whose data exceeds the threshold goes to state S_0.

Once an event occurs, each sensor node near the event broadcasts its energy to a close gateway node. Gateway nodes exchange sensor energy information with each other. One of them sorts the values, selects k nodes with highest energy as CHs, and broadcasts the information. Then each CH broadcasts an advertisement and non-CH nodes choose the nearest one as their CH based on received signal strength.

In steady phase, each CH sets up a TDMA schedule to avoid packet collusions and transmits the schedule to member nodes. Then CH collects the data, aggregates it, and transmits the data to gateway node. Gateway node transmits the collected data to sink in single hop.

Routing in WSN is the process of collecting gathered data from sensor nodes at sink. In a single-hop architecture where nodes communicate directly with sink, no routing structure is needed. Although it is simple, this scheme does not benefit from the energy savings obtained by use of a multihop architecture. Moreover, since sensors have limited communication range, communicating with the sink directly may not be

possible at all times. Therefore, to design a scalable protocol, multihop routing should be employed. For an ordinary WSN, nodes do not change location so proactive routing protocols cause too much overhead. On the other hand, on-demand routing protocols incur a large delay.

In [40], the network is modeled as a graph and the objective is to find a balanced spanning tree originating at the sink with minimum number of edges and same number of child nodes. The logic behind this idea is that this formation minimizes the total energy consumption and balances the energy expenditure of all nodes. It is assumed that sink knows locations of all nodes priori by some means like GPS. The steps of the algorithm are as follows:

Step 1 : Path construction is started from sink.

Step 2 : In each iteration, minimum weighted edge as much as the decided child number of nodes from a vertex in the tree to a vertex not in the tree is selected and added to the tree. Number of child nodes is calculated as $\frac{N\pi R^2}{A}$ where N is the total number of sensors, R is the transmission range of a sensor, and A is the area of the field. The weight of an edge is calculated as the energy expenditure for both transmitting and data aggregation.

Step 3 : When neighbor node is found over the limit of the search, the node is labeled as a leaf.

Step 4 : This procedure is repeated until all nodes are added to the tree.

Step 5 :After a specified number of rounds, sink recomputes the routing information excluding the dead nodes and resetting the number of child nodes.

Graph abstraction is also used in [42]. Edges exist between all pair of nodes with a corresponding weight, which is the transmission cost between each pair. Let L denote the maximum possible lifetime of any node and C denote maximum transfer cost between any two nodes. A scaling factor ξ is defined as $\xi = \frac{L}{C}$.

The lifetime of a path is bounded by the lifetime of all nodes along the path, thus lifetime of a path can be defined as the minimum lifetime of all nodes along the path.

The cost of a path is the sum of all costs calculated between two consecutive nodes along the path from source to destination. The path selecting parameter β is defined as:

$$\beta_i = \frac{Lifetime\ of\ path_i}{\xi(Cost\ of\ path_i)}.$$

The algorithm selects the path with largest β value, which in turn implies the path with higher lifetime and lower cost. With this selection, the algorithm tries to find a balance between the two competing factors.

A reliable routing scheme is offered in [43] that maintains connectivity after node failures. The approach is to create and maintain a spanning tree that connects all nodes to the sink. The tree is dynamically maintained. Reconfiguration procedure -in case of node failures- should be energy efficient. Reconfiguration neighborhood is limited that bounds the energy required. This limitation also implies that some node failures may cause nodes to be isolated from the tree although a tree can still be constructed.

To construct the spanning tree, flooding approach is used. Sink broadcasts a *parent* message with its ID. Nodes without parents, that receive message set their parent and grandparent information and broadcast their own *parent* messages. Neighboring nodes that already have parent, set their children and sibling information based on *parent* messages they receive.

When a faulty node is found, the path must be reconfigured. There are three flow directions: HIGH, LOW and UNKNOWN. When a node finds a neighbor that guarantees a cycle free path, it sets neighbor relationship to HIGH (higher in the tree). A neighbor that cannot be used to send to sink is set to LOW. Otherwise, it is UNKNOWN. Steps of the reconfiguration phase are as follows:

Step 1 : Whenever a node needs a new parent because of failure of its current parent, it broadcasts an *I need parent* message with its ID, ID of its parent, and grandparent. Any node that is not child or sibling of that node, receiving the

message checks if it can provide HIGH direction based on its local information and information in the message. If so, it acknowledges the message. After receiving all acknowledgments, initiator node selects the one of them as its parent and broadcasts the selection. By means of broadcast, children of initiator also update their grandparent information.

Step 2 : If the initiator cannot find a cycle free path, it broadcasts a *Cannot find cycle free* message. Then each sibling of initiator tries to find its own cycle free path by broadcasting *I need parent* message. If a sibling finds a cycle free path, it acknowledges the initiator and thus initiator finds its new cycle free path.

Step 3 : If initiator has still no parent, it sends *I cannot be your parent* message to its children. Then all children of initiator start to find their own cycle free path by broadcasting *I need parent* messages. If any of the children finds a new path, then it acknowledges this by a broadcast message and initiator has a new path from that child.

Step 4 : If initiator still has no parent, it knows that neither its siblings nor its children can provide a HIGH flow direction. Then, the node tries to obtain more local information from its neighbors. It broadcasts an *I need information* message and gets acknowledgments from its neighbors that contain their neighbors. With this information, initiator tries to capture local connectivity. LOW flow direction neighbors are discarded. It selects a node with UNKNOWN direction and sends a path find message to that node. Path find message propagates upwards in the tree and each node checks for a possible configuration. If a new configuration exists, the initiator is acknowledged. If a path cannot be found, initiator tries other UNKNOWN direction neighbors. If there are no such nodes, then initiator stops the search.

A central algorithm that takes network topology as a graph input is introduced in [44]. Each node is assigned a degree of importance. Degree of importance is calculated as the number of disconnected cluster that results as that node dies. The algorithm then tries to use important nodes as little as possible since their death results in disconnected components.

A Laplacian matrix is used, which is formed as:

$$L(u,v) = \begin{cases} d_v & \text{if } u = v \\ -1 & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

where d_v is the degree of node v. After importance of each node is assigned, routing is done in such a way that minimizes the total importance degree of the nodes along the path.

A sink driven routing strategy that facilitates a node to know the information of one hop destination node that is nearest to the sink is presented in [31]. Algorithm steps are given below:

Step 1 : Every node broadcasts *hello* messages to each other in order to construct neighbor lists.
Step 2 : Sink starts the route by sending out a *route request* packet.
Step 3 : The node receiving the *route request* packet, record sink as their next hop node and transmit it to their neighbors.
Step 4 : The nodes receiving the message, record sender's ID as their next hop node and relay the packet with their own ID.
Step 5 : Go to step 4

In [45], directional flooding is used for initial route setup. Sink broadcasts a query message into network with TTL equal to zero. Whenever a node receives the message, it increments TTL. If the TTL value is lower than the node's distance in number of hops to the sink (which is infinity initially), the node modifies its distance to the sink.

After this setup phase, when a node decides to send a packet to the sink, it checks whether its upstream neighbor exists. If that is the case, it sends its packet to that node. Otherwise it wakes up all neighbors, broadcasts an *election* message, and starts a timer. Any node receiving the *election* message compares the TTL field with its

hop number to the sink. If the node is one hop closer to the sink, it marks itself as a candidate, calculates service time that is a function of its remaining energy, and replies with an *election_reply* message. When the timer expires, the initiator node checks the messages received and selects the node with the highest service time as its leader. Then the node sends data out to the leaders. Other upstream nodes are shutdown to save energy and avoid overhearing.

A probabilistic routing scheme is offered in [46]. Node degree and link loss are the main parameter affecting the selection of next hop node. Node i sends its packet to upstream neighbor j with probability P_{ij}, which is given by:

$$P_{ij} = \frac{(Outdegree\ of\ j)^{\alpha}(1-p_{ij})^{\beta}}{\sum_{\forall k \in K}(Outdegree\ of\ k)^{\alpha}(1-p_{ik})^{\beta}}$$

where K is the set of upstream neighbors of i, p_{ij} is the probability of link loss between i and j, and α and β are adjustment parameters.

A routing scheme for clustered WSNs is discussed in [47]. In this architecture, each node has a primary membership to one cluster and a secondary membership to another cluster. The CH selection algorithm is similar to HEED, but this time a node selecting the lowest cost CH as its (primary) CH also selects the second lowest cost CH as its secondary CH.

Secondary CHs are used for route discovery. During route setup phase, each CH broadcasts a route request. All member nodes receive this request, forward it to their secondary CH, and the procedure continues in the same manner as that CH also broadcasts a route request message to its members. Although the idea may seem charming at first, it is not much different than flooding the whole network.

Another routing mechanism for clustered WSNs deployed for a water shed monitoring project is proposed in [48]. The protocol is designed for a large number of stationary sensors that are required to be as simple as possible. CHs are static and

have more advanced communication equipment and larger power supply than ordinary sensors. For the application, it was necessary for all sensors transmit data so data bundling and aggregation is used for energy conservation. Routing paths for each cluster are discovered independently with a beacon based approach. Periodically, each CH sends a beacon. Every node that receives the beacon evaluates the information and possibly updates its routing information. If an update has been done, receiving node also forwards the beacon to other nodes for each node to discover a maybe better path to CH. Beacon header contains a link rank part that indicates the cumulative link rank up to the receiving node and updated at each node. It may be based on hop count, remaining battery life or an aggregation of the two. Then each node selects the path with the minimum link rank towards the CH. The protocol also runs on CHs in order to find the path from each CH to sink.

3. POWER EFFICIENT DISASTER DETECTION

The main contribution of this work is to design a scheme for disaster detection. Basic goal of PEDD is allowing nodes to sleep whenever possible to increase the lifetime of the network while achieving tolerable delay. To operate a long-lived network, message exchanges need to be minimal since transceiver is the most energy consuming unit of a sensor. Moreover, nodes should be put to sleep whenever possible where the energy expenditure is minimum. These are the main aspects for the proposed scheme.

In this chapter, we first discuss the implemented network topology together with some assumptions and constraints. Initialization of the network is discussed subsequently. Then, we explain the operation of the network without any disaster. Finally, operation of the network under a disaster is discussed.

3.1. Network Topology

We constrain that two nodes within communication range of each other cannot hear each other's transmission if they are separated by the disaster. In other words, disaster distorts the channel considerably. As an illustration, a couple of cases are shown in Figure 3.1.

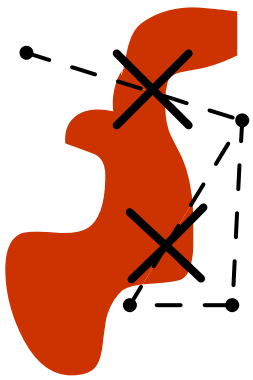


Figure 3.1. Communication in case of a disaster

We assume that ROI is a huge area. In this case, addition of actuator nodes to the network is a good idea. With their powerful batteries, they can reach farther distances

than an ordinary sensor, which decreases latency and saves energy for the sensor nodes. We assume that an actuator can reach the sink in one hop. This assumption can be relaxed by deploying a simple routing algorithm between the actuators.

We assume that the network has a grid structure where each grid point has an actuator node on each corner. As an example a 2x3 grid sized network is given in Figure 3.2 where the big dots represent the actuators and small dots represent the sensor nodes. We call a grid point as a grid area. Considering the few number of actuators and assuming a large distance between them, their regular deployment is a reasonable assumption. The reader should note that the sink is located outside ROI and is not depicted in the figure. The logic behind using four actuator nodes per region is that a sensor node may not be able to reach to its nearest actuator because of a blocking disaster. With extra actor nodes, it is more likely that a sensor node can reach at least one actuator.

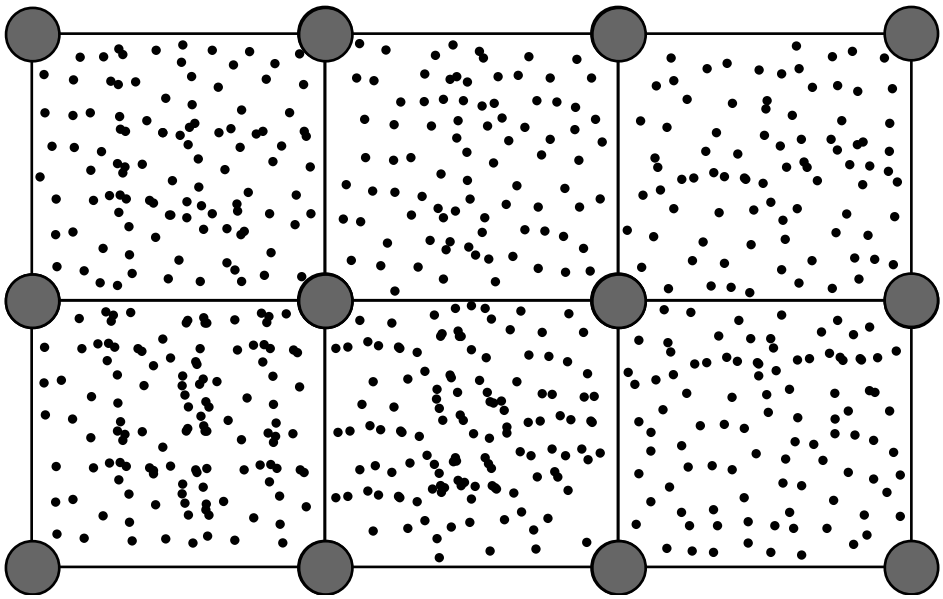


Figure 3.2. A two by three network

For sensor nodes, we assume random deployment where sensors are uniformly scattered over the field. By random deployment, not all grid areas have exactly the same number of sensors.

From this point on, we are focused on just a single grid area. We assume that actuator nodes are well coordinated and a message delivered to an actuator node is guaranteed to be delivered to sink. That is to say, coordination and communication of actuator nodes with each other is beyond the scope of this work. When talking about actuator nodes we refer to them as NWAN (North-West Actuator Node), NEAN (North-East Actuator Node), SWAN (South-West Actuator Node), SEAN (South-East Actuator Node).

Most of the sensors manufactured today have power control capability which allows a node to adjust its power level. In the literature, a sensor's transmission strength is usually assumed to be a continuous variable. In this work, we assume a discrete model where a node's transmission power level can take one of the values built in to the sensor. This is a realistic assumption since sensor transceiver units are not complex enough to allow a continuous power level selection. Furthermore, this assumption holds with the real implementation of the sensor nodes. Besides power control, each node has a predefined normal transmission range, tx_{normal}, and a maximum transmission range, tx_{max}. tx_{normal} is predefined and may be built into the node whereas tx_{max} depends completely on the hardware.

3.2. Network Initialization

In the network initialization phase, nodes form neighbor lists. Each node keeps five lists for its neighbors. *allNeighbors* list contains all neighbors of a node, whereas *NEneighbors*, *NWneighbors*, *SEneighbors*, *SWneighbors* contain the neighbors of the node that are closer to NEAN, NWAN, SEAN, and SWAN, respectively. Moreover, each node keeps the number of hops to each of its neighbor actuator in the variables $hopCountNE$, $hopCountNW$, $hopCountSE$, $hopCountSW$, which are all zero initially.

After deployment, all nodes are in the active state with all units on. Then, SWAN broadcasts a *flood* message that contains a hop count field. Initially this field is set to zero. The packet's SWAN flag is set to 1 to indicate it originates from SWAN. In

addition, t_{thr} value that causes the nodes to go into disaster state is included in this message.

A node receiving the packet performs the steps given by Algorithm 1 in Appendix A.1. It first adds the transmitter to its *allNeighbors* list if it is not already in there. Then, receiver checks if its $hopCountSE$ needs an update. If so, it is updated and the transmitter node is added to the $SWneighbors$ list. In addition, the receiver node waits for some random time and transmits its own *flood* message with its own hop count to SWAN with tx_{normal}. The waiting time is selected randomly between 0 and $floodCW_{max}$ to prevent collisions. The selection of $floodCW_{max}$ is important because a large value may cause unnecessary energy wastage whereas a small value may increase collisions.

The algorithm is same for *flood* packets originating from other actuators. The only difference is that receiver uses its hop count to that particular actuator.

When the flood packet reaches to the opposite actuator (NEAN for SWAN), the next actuator can begin sending its own *flood* packet after waiting for a period of time. This waiting time, $timeout_{AN}$, ensures there are no ongoing transmissions when the next actuator begins its transmission. The reader should note that only in this stage, nodes use a four way handshaking mechanism since the traffic load is high.

3.3. Normal Operation

When the final actuator's *flood* packet reaches the opposite actuator, it waits for a $timeout_{AN}$ period of time and broadcasts a *startDC* packet with power level enough to be heard by all nodes in the network.

Normal operation of PEDD uses a slotted channel. There are two types of slots:

- Normal slots: In these slots only a subset of nodes in the network are active; others are in sleep state. By sleep state, we mean all units of the sensor are

off except a timer. Our approach differs from the sleep states usually found in the literature where a sleeping node's transceiver unit is off but its sensing and processing units are active.

- Random access (RA) slots: All nodes in the network are active in these slots with all their circuitry on. These slots are used for delivering important packets such as *alarm* packets to actuators in a quick way.

Duration of normal slots and RA slots are equal to each other and just as long to facilitate transmitting a packet and receiving an acknowledgment. RA slots are less frequent than normal slots. Namely, one RA slot succeeds each $RA\_frequency$ number of normal slots. Therefore, for K_n normal slots, number of slots in a frame, K, is given by Equation 3.1. The structure of a frame is shown in Figure 3.3.

$$K = K_n + \lceil \frac{K_n}{RA\_frequency} \rceil \tag{3.1}$$

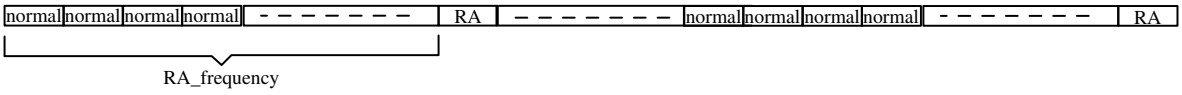


Figure 3.3. Frame structure

Each node wakes up for only one normal slot and for all RA slots in a frame. A node can receive a packet only during its selected normal slot or at RA slots but it can transmit at any particular slot. It is assumed that all nodes are synchronized. This can be done with periodic heart beats from actuators.

When a sensor receives a *startDC* packet, it selects its normal slot, *normalSlot*, which is given as:

$$\begin{cases} \frac{(sensorID \ mod \ K_n) - 1}{RA\_frequency} + (sensorID \ mod \ K_n) & \text{if } sensorID \ mod \ K_n \neq 0 \\ \frac{K_n}{RA\_frequency} + ((sensorID - 1) \ mod \ K_n) & \text{if } sensorID \ mod \ K_n = 0 \end{cases}$$

where $sensorID$ is the unique ID of a node. It is assumed that all nodes have integer unique IDs. If that is not the case, MAC addresses can also be used. Since nodes are distributed uniformly random over ROI, selected slots of nodes are also uniformly random.

After selecting its normal slot, a sensor goes to sleep mode to wakeup at either next RA slot or its normal slot, whichever is closest. When a node wakes up, it stays awake for at least a *timeout* amount of time. If it does not hear anything on the channel and its sensed value is below t_{thr} it goes to sleep after a period of *timeout*.

During normal operation, sensor nodes periodically send their reported data to the closest actuator once in every T_{report} amount of time using *alive* messages. The first sending time is selected randomly between 0 and T_{report} to distribute traffic load. This mechanism provides actuators a picture of the network and they may guess which nodes may be dead already. To increase lifetime, T_{report} should be as large as possible. When a node's time for transmitting *alive* message comes, it first finds the closest actuator node by looking at the hop counts to each actuator. Then, it checks neighbor list corresponding to that actuator node and finds the node whose selected normal slot is closest. *alive* message is scheduled to be transmitted during that slot. If transmission is successful, the receiver performs the same steps. If not, transmitter finds another node from the list whose selected normal slot is closest and schedules transmission at that slot. Note that, transmissions are performed with a power level to achieve a transmission range of tx_{normal}.

With this mechanism, nodes do not need to know each other's selected normal slots. Since they know all their neighbors' IDs, they can calculate them. Hence, it is not necessary for nodes to exchange schedules. A per hop delay of $frameLength/2$ is incurred on the average. Since *alive* messages do not contain time sensitive data, this does not constitute a problem.

When sending a packet, a node first listens to the channel for a $waitingTime$ of time, which is given as:

$$waitingTime = backoffPeriod + CW * random(0, 1) \tag{3.2}$$

where $backoffPeriod$ is the fixed backoff time and CW is the contention window size.

The reader should note that, IEEE 802.11 DCF is not employed to minimize energy usage since traffic load is very low during normal operation, especially when T_{report} is large.

3.4. Operation during a disaster

Before discussing the details of operation under the disaster, first we explain the disaster model. We model the disaster as two nested ellipses. The inner ellipse is called the *core* and the outer is one called the *shell*. An example disaster is given in Figure 3.4.

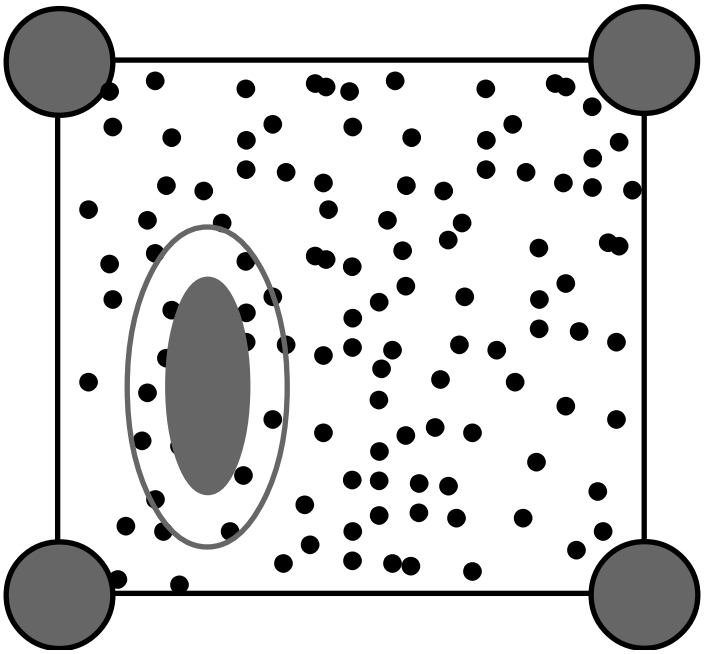


Figure 3.4. Disaster model

The nodes inside the *core* are assumed to be completely destroyed and do not function anymore. The ones between the *core* and the *shell* sense values above t_{thr}. Every point inside the *core* has its phenomenon value over t_{max}, the maximum value a sensor can operate. We assume that the value of the phenomenon gets the value t_{thr} just on the *shell*, t_{max} just on the border of the *core*, and is linearly distributed between two borders. That is, a point has a phenomenon value of

$$t_{thr} + \frac{d_{ns}(t_{max} - t_{thr})}{d_{sc}}$$

where d_{ns} is the distance between the node and the point closest to the node on the *shell*, and d_{sc} is the distance between the *shell* and the *core* on the line that connects the closest point on the *shell* and the closest point on the *core* to the node. This line passes through that particular sensor node. The mathematics of finding these points is given in Appendix B.

3.4.1. Alarm Mechanism

When a node's sensed value exceeds t_{thr}, it schedules an *alarm* packet to be transmitted during the nearest RA slot and goes to sleep. The pseudo code of the algorithm is given by Algorithm 2 in Appendix A.2. When it wakes up during the RA slot, it again senses the channel for a duration of $waitingTime$, given by Equation 3.2. If there is no activity on the channel, it transmits its packet. On the other hand, if it receives an *alarm* packet, it cancels its packet, forms a new *alarm* packet, and schedules its transmission for the nearest RA slot. The algorithm is the same for nodes inside and outside the disaster region.

Alarm packets have four flags that show whether the packet is intended for a particular actuator. For instance, $intendedSE$ flag indicates whether the packet is intended for SEAN, or not. In addition, the packet also has four fields that contain transmitter's hop count to a particular actuator. As an example, $hopCountNW$ shows the transmitter's hop count to NWAN. The algorithm for forming the new *alarm* packet is given by Algorithm 3 in Appendix A.3. Basically, it checks the set flags of the

received *alarm* packet and compares the receivers hop count to that actuator with the received packet's hop count to the same actuator. If the difference is greater than one, *newAlarm*'s corresponding flag is set and corresponding hop count field is assigned. If all flags of the *newAlarm* is set to $FALSE$, the algorithm returns $NULL$, otherwise it returns the *newAlarm*. *Alarm* packets are transmitted at maximum power level to a transmission radius of tx_{max}. The logic behind seeking a difference greater than one when comparing received *alarm* packet's hop count and receiver's hop count to the same actuator is that we assume $tx_{max} > 2tx_{normal}$, and we want the packet to propagate as much as possible. Therefore, an *alarm* packet propagates at least two hops for each RA slot.

Although an *alarm* packet may be sent to the closest actuator, the path may be blocked due to disaster. We take the safer approach and send it to all four actuators. By using flags and hop counts, only one *alarm* packet is sufficient instead of four packets destined to each actuator.

3.4.2. Grouping Mechanism

Nodes that sense, that value of the phenomenon (either during their normal slot or RA slot) is above t_{thr} try to send the *alarm* message. These nodes abandon their normal DC and start grouping process during the normal slot following the closest RA slot.

We do not employ a probabilistic approach like LEACH [22] for grouping, because probabilistic approaches require number of desired heads a priori. Since the location of the disaster is not known at deployment time, the number of heads cannot be known a priori. In case it is known, another problem with probabilistic approaches is that they do not guarantee coverage. That is to say, some nodes may be left without heads. As an example let's consider the case where all heads are located to the left of the *core* as given in Figure 3.4. In this case, nodes to the right should execute the algorithm again. Moreover, there are more group heads on the left than necessary, with few members, diminishing the benefit of grouping. Although, this problem could

be avoided by increasing the probability of becoming the head, it would still result in groups with few members.

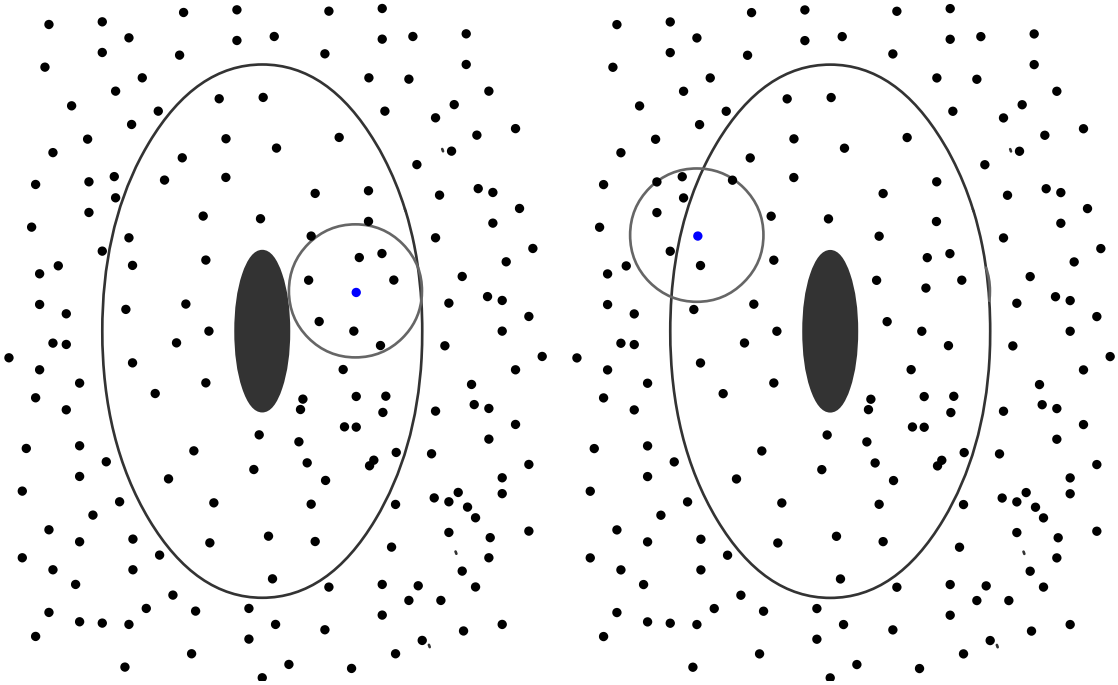
On the other hand, an iterative approach like HEED [27] has the disadvantage of protocol overhead since nodes need to exchange messages for each iteration. This also increases the time required for group formation, which in turn increases response time.

Considering these factors, we use a contention window based approach, where the size of the contention window is modified. The node with the lowest selected value starts sending its group head advertisement first and becomes the group head in its locality.

The properties of a group head can be listed as follows:

- Group heads should be selected from nodes that have higher residual energy compared to other nodes in their neighborhood.
- By considering the model we use for the disaster, we want the group head to be close to the middle of the *core* and the *shell*. The logic behind this is that the group head should cover more area in the disaster region, have more members, and thus, be utilized efficiently. Another approach is selecting nodes that are close to the *shell*. However, since nodes outside the *shell* do not send any data, these group heads are ineffective. The same argument also applies to the case when group heads are selected close to the *core*. The cases where group heads are selected close to the middle versus close to the *shell* are shown in Figure 3.5.

By considering the two factors above, we devise three functions. The first one takes the sensor reading, t_{sensor}, into account, and the second one takes battery status of the sensor, $p_{current}$. Our final function is the multiplication of the two.



(a) Group head in the middle (b) Group head near *shell*

Figure 3.5. Possible group head locations

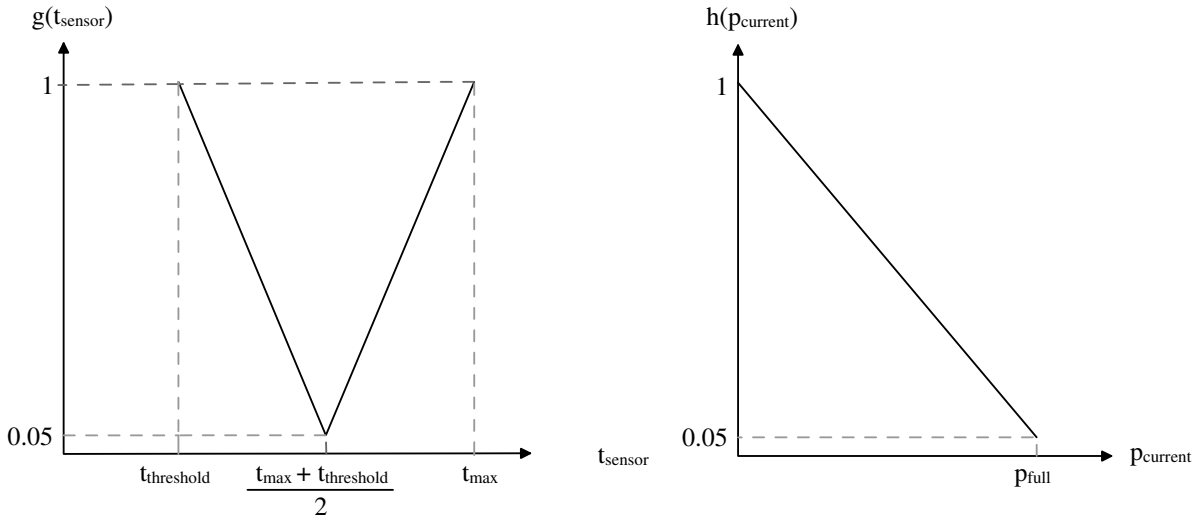
$$g(t_{sensor}) = \begin{cases} 1 - \dfrac{0.95(t_{sensor} - t_{thr})}{0.5(t_{max} - t_{thr})} & \text{if } t_{thr} \leq t_{sensor} < \frac{(t_{max}+t_{thr})}{2} \\ 1 - \dfrac{0.95(t_{max} - t_{sensor})}{0.5(t_{max} - t_{thr})} & \text{if } \frac{(t_{max}+t_{thr})}{2} \leq t_{sensor} < t_{max} \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

$$h(p_{current}) = 1 - \frac{0.95 p_{current}}{p_{full}} \quad (3.4)$$

$$f(t_{sensor}, p_{current}) = g(t_{sensor}) h(p_{current}) \quad (3.5)$$

$g(t_{sensor})$, given in Equation 3.3 is a triangular function and favors nodes that have sensor readings close to $\frac{t_{max}+t_{thr}}{2}$. That is, nodes close to the middle of two ellipses are more likely to become group heads. On the other hand, $h(p_{current})$ in Equation 3.4 is a monotonically decreasing linear function of $p_{current}$ and favors nodes that have more energy.

These two functions are depicted in Figure 3.6 and the sketch of $f(t_{sensor}, p_{current})$ is given in Figure 3.7. Both functions take values between 0.05 and 1. A node with a reading value greater than or equal to t_{thr} selects a random number between 0 and $f(t_{sensor}, p_{current}) * CW$. We do not want neither of the functions to get the value of 0, since this destroys the affect of the other, and cause a node with a reading value of $\frac{t_{max}+t_{thr}}{2}$ to become group head, no matter what its battery level is.



(a) Graph of $g(t_{sensor})$

(b) Graph of $h(p_{current})$

Figure 3.6. Sketch of $g(t_{sensor})$ and $h(p_{current})$

With this algorithm, all nodes try to send their advertisements. The nodes that have more energy and close to the middle are more likely to win the channel. A node hearing an advertisement cancels its own advertisement. If a node hears multiple advertisements, it selects the transmitter based on received signal strength. In other words, it selects the closer transmitter.

Nodes that hear the advertisements send join requests to the selected group head. The group head than forms a TDMA schedule and transmits it to member nodes. Each node has a reserved slot in the schedule. An additional slot is used by the group head. In this slot, the group head may send control packets. Nodes wait for at least *timeout* amount of time during this slot, and if there is no incoming transmission from the group head, they go to sleep. TDMA schedule is used for determining at which normal

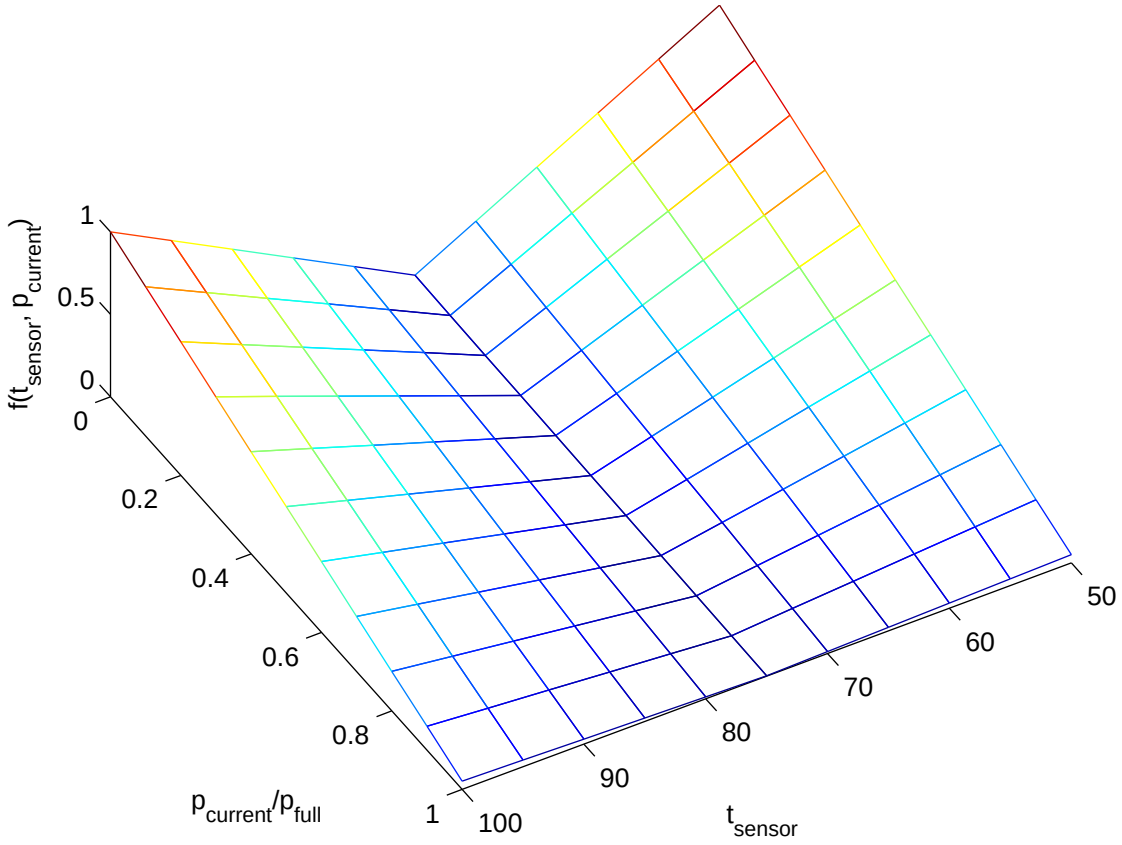


Figure 3.7. Graph of $f(t_{sensor}, p_{current})$

slot a node wakes up. Both group head and member nodes also wake up at RA slots. Member nodes abort their current duty cycle and send their collected data to the group head in their reserved slot specified by the TDMA schedule. The group head collects the data and aggregates it. During this state, member nodes adjust their transmission power to the minimum level required to reach the group head.

3.4.2.1. Re-grouping. Most approaches in the literature use periodic clustering; nodes periodically run an algorithm to form new groups. The reasoning behind this approach is that the cluster heads should be rotated since they consume too much energy. For applications that require constant monitoring, this is a reasonable approach. On the contrary, for long lived event driven applications, overhead of rotating the cluster heads may be unbearable and nodes are not able to report data during setup phase for the new cluster. Moreover, for most cases re-grouping may not be necessary since group heads can handle the burden during disaster most of the time and new nodes need to be deployed around the disaster area anyway. Considering these factors, we avoid constructing groups as long as there is no disaster. When the disaster strikes, we form groups among sensors around the disaster for data aggregation.

In our scheme, when a group head's energy falls below $p_{critical}$ per cent of p_{full}, it starts the re-grouping procedure. The group head first sends a *poll* message to its members in its reserved slot. In the next cycle, each member node sends its residual battery level to the group head. The group head, which already knows the readings of the member nodes, decides on the next group head based on $f(t_{sensor}, p_{current})$, broadcasts this new group head, and passes along path information to the new group head. The distance between a member node and the new group head may be more than tx_{normal}. However, this does not constitute a problem since member nodes adjust their transmission ranges accordingly.

3.4.3. Routing

After groups are formed, group heads collect data from member nodes and aggregate it. Group heads need to forward this aggregated data to the actuators. Problems with disaster still exist in this phase. That is some neighbors and some actuators may be impossible to reach. An easy solution may be sending data just like *alarm* packets during RA slots. Although, this scheme is easy to apply, it has some drawbacks. Firstly, unlike *alarm* packets that are sent just once after initial sensing of the disaster, aggregated data is sent periodically after each TDMA cycle. In addition, the group head does not know which actuator is reachable. So, it needs to be sent to all four actuators. This creates an unnecessary burden on the network. Secondly, *alarm* packets are sent with maximum power to achieve a transmission range of tx_{max} to inform about the disaster as soon as possible. On the other hand, sending aggregated data with maximum power level is too power consuming since it needs to be sent periodically. Finally, using just RA slots increases latency.

We employ a scheme that uses both RA slots and normal slots that allows duty cycling. We first build paths from the group head to the actuators. These paths show which actuators are reachable and some possible paths to those actuators. To achieve this, the group head sends a *routeRequest* packet. Similar to *alarm* packet, this packet has four flags that show the intended actuators and four fields that indicate hop count of group head from those actuators. Initially, all flags are set. Packet also contains the IDs of the nodes along the path that just contains the ID of the group head initially, and a bitmap that indicates whether the corresponding node in the list is a group head, or not. The group head sends this packet during its first RA slot after becoming the group head with a power level to achieve a transmission radius of $2tx_{normal}$. We use a range of $2tx_{normal}$ because the group head may be very close to the *core* and only nodes that are in the disaster area may be reachable. In this case, we do not want non-group head nodes to participate in the routing process. An example case is shown in Figure 3.8. In this example, group heads just above and below the *core* are very far from the *shell* and not able to reach nodes outside the disaster area.

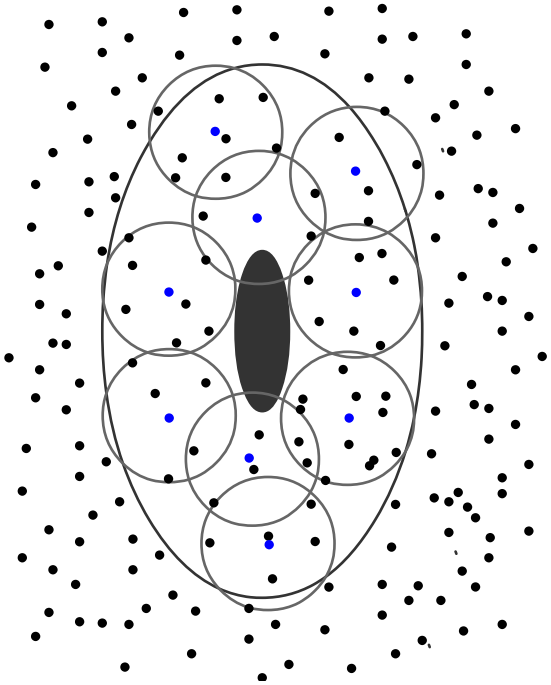


Figure 3.8. An example case with two group heads unable to reach nodes outside the disaster area

By using a range of $2tx_{normal}$, it is guaranteed that another group head is able to hear this message. A node hearing this message discards it if it is not a group head and its sensor reading is above t_{thr}. Other nodes compare the hop count fields in the packet with their hop counts to actuators and update the packet's flags accordingly.

A node outside the disaster area receiving the *routeRequest* packet modifies flags of the packet, appends its ID to the packet, and schedules it for transmission during the next RA slot. Unlike group heads, ordinary nodes transmit at a range of tx_{normal}.

Eventually, *routeRequest* packet reaches to an actuator. Only one dedicated actuator is responsible for dealing with path selection. An actuator receiving the *routeRequest* informs the dedicated actuator for the reception of the new path. Dedicated actuator first checks whether this is the first *routeRequest* packet that is received from that group head. If that is the case, it starts a timer that fires just after *pathTimeout* amount of time. If this is not the case, than that path is added to the list of paths for that actuator node.

When the timer fires, the dedicated actuator node selects the shortest *maxPathCount* number of paths from the list of available paths. This selected path information is broadcasted during the nearest RA slot to all nodes in the network in a *routeReturn* packet.

A node receiving the *routeReturn* packet checks if its ID is contained in any of the paths. If its ID does not exist in the list, it continues its normal operation. Otherwise, it calculates the normal slot during which, it is awake for data reception. The following normal slot is used for data forwarding. Moreover, the node also knows its previous and next hop neighbors along the path. From that moment on, the sensor node is in the *on* state during these two slots, its regular wakeup slot, and RA slots.

As an example, let us assume that a *routeReturn* packet that contains the following path information {13, 7, 9, 8, 22, 30}, {13, 15, 41, 44, 88}, {13, 26, 32, 10} and node 13 is the only group head node that is contained in the packet. Node 13 is the initiator that started the path formation process and all the other nodes are outside the disaster area. Assuming that this packet is received during slot n (that needs to be an RA slot), upon receiving the packet, node 20 does not modify its schedule whereas, node 7 is awake during slots $\{n+1,\ n+3*RA\_frequency+1,\ n+6*RA\_frequency+1,\ \dots\}$ in addition to its regular slot and RA slots (Since its previous hop is the group head that sends the aggregated data during RA slot). On the other hand, node 41 is awake during slots $\{n+RA\_frequency+2,\ n+RA\_frequency+3,\ n+4*RA\_frequency+2,\ n+4*RA\_frequency+3,\ n+7*RA\_frequency+2,\ n+7*RA\_frequency+3,\ \dots\}$. Node 41 receives data from node 15 during the first slots of the given pairs and forwards it to node 44 during the next one.

When there is another group head contained in the packet, initiator group head sends its aggregated data to that group head during an RA slot. Receiving group head transmits received data during the next RA slot to the next hop.

Although, a bit complex, this scheme provides duty cycling for the nodes on the path. Furthermore, in each slot data moves one hop closer to the sink. If only RA slots

are used for routing, a delay of $(RS\_frequency + 1) * slotLength$ would be incurred for each hop.

3.4.4. Returning to Normal Operation

We employ a simple mechanism for nodes to return back to normal operation when the disaster in ROI ceases to exist. When a group head's collected data is below t_{thr} for all members for $2*pathCount$ TDMA cycles, it sends an *endGroup* message to its members during its reserved slot. Nodes hearing this message break their associations with their group head and return to their normal operation.

For the nodes on the routing path from the group head, they return to their normal operation when the aggregated data they get from the group head is below t_{thr} twice.

New nodes need to be deployed inside the *core* to achieve satisfactory coverage, since all nodes inside are destroyed. Moreover, after this redeployment, network initialization algorithm needs to be executed again in order to form neighbor lists of recently deployed nodes.

3.4.5. Packet Sending

We choose not to use RTS/CTS scheme since it incurs a large overhead and we do not expect a high traffic load anyway. In our scheme, when a node broadcasts a packet whose destination is not known like *alarm* or *routeRequest*, it does not require an ACK packet, assuming that a node receives it correctly. On the other hand, packets that are transmitted to a specific neighbor like *alive* packet or *report* packet, require an ACK. If transmitter node does not get an ACK back, it schedules that packet for transmission until its retransmission count reaches a predefined maximum number of retransmissions. After that, the packet is dropped.

4. SIMULATION RESULTS

In this chapter, we first give the simulation scenario and parameters. We present and discuss simulation results, subsequently.

4.1. Simulation Scenario and Parameters

For simulation, we assume deployed nodes are MICAz nodes from Crossbow [49]. Their specifications are given in Table 4.1. Although MICAz nodes come with three battery options (250 mA-hr, 100 mA-hr, 3000 mA-hr), we use 2.5 mA-hr battery in our simulations to decrease simulation time. Radio current for transmission is not given in the table since nodes adjust their transmission power based on the required output level.

Table 4.1. Sensor parameters

Full Battery Level for a Sensor Node	2.5 mA-hr
Processor current at full operation	8mA
Processor current at sleep	8μA
Radio current in receive mode	8mA
Radio current in sleep mode	2μA
Sensor board current at full operation	5mA
Sensor board current at sleep	5μA

Other parameters that are related to the network topology and sensor nodes are given in Table 4.2. Parameters of PEDD are given in Table 4.3

In each of our runs, the same topology that is generated randomly, is used to be consistent. We simulate PEDD for two different scenarios. In the first scenario, no disaster occurs during the lifetime of the network. In the second scenario, disaster starts in the field just after the initialization phase is completed. We compare the results of our algorithm with T-MAC [18] protocol.

Table 4.2. Topology and other sensor parameters

Field Size	1000 m x 1000 m
Number of sensor nodes	2000
t_{thr}	50 ℃
t_{max}	100 ℃
tx_{normal}	45 m
tx_{max}	100 m
Bandwidth	16 Kbps

Table 4.3. PEDD parameters

K_n	100
$RA\_frequency$	10
K	110
$SlotLength$	500 msec
$timeout$	100 msec
$backoffPeriod$	30 msec
CW	50 msec
Maximum Retries for a Packet	3
$maxPathCount$	5
$timeout_{AN}$	3000 msec
$floodCW_{max}$	100 msec
T_{report}	21600 sec
Packet Size	500 bytes
ACK Size	20 bytes
Packet processing time	20 msec
$p_{critical}$	4

T-MAC [18] protocol uses a four way handshaking system for communication. It also uses a timeout scheme. Nodes exchange schedules and perform virtual clustering. That is neighbor nodes sleep and wake up at the same time in a frame. Active period of a node is adaptive. When a node senses communication in the channel, it stays awake since it may be the future receiver of the packet. Otherwise it goes to sleep after TA amount of time.

In order to be fair, when comparing PEDD with T-MAC, we assume that virtual clustering is already performed and nodes are synchronized. Without this assumption, T-MAC requires all nodes to exchange schedules periodically, incurring a large overhead. All nodes wake up at the same time, which is the start of a new frame. In addition, to make T-MAC more effective, we assume data aggregation through the network. For instance, if a node receives a packet and it already has a packet to send in its buffer, it aggregates their data. In our model, aggregation always results in a single packet. By doing this, we favor T-MAC by the use of in-network processing.

Similar to our algorithm, T-MAC also requires neighbor information at each node. We use the make strategy we use in our scheme to form neighbor lists available to T-MAC. Each node keeps the list of its neighbors that are closer to a particular actuator. The initialization process can be seen as a setup phase that is the same for both algorithms. We exclude energy consumption in this stage since it is the same for both algorithms and we want to see the big picture more clearly.

T-MAC uses randomized shortest path routing scheme; that is, a node selects the receiver from the lists of neighbors that are closer to the destination in a random fashion. For our topology, nodes try to send their data to the closest actuator and select the next hop neighbor randomly from the list of neighbors closer to the specified actuator. T-MAC uses a fixed contention interval and waiting time is selected randomly from the interval. Parameters for the T-MAC algorithm are given in Table 4.4.

Table 4.4. Parameters for T-MAC

TA	100 msec
Contention Interval	50 msec
Frame Length	2000 msec
Maximum Retries for a Packet	3
Packet Size	500 bytes
ACK Size	20 bytes
Packet processing time	20 msec
RTS Size	20 bytes
CTS Size	20 bytes
T_{report}	21600 sec

4.2. Results

As stated before, both algorithms are run for the same topology that is generated randomly. Each case is run for ten times with different seeds for the random number generator and their average is depicted in the figures. Both algorithms are analyzed in terms of network lifetime, number of dead nodes at any time, packet latencies, and average battery consumption of nodes. Node deaths percentages are given in figures, where five per cent node death means five per cent of all nodes in the network are dead either because of battery exhaustion or disaster. The reader should be reminded that the simulations are run with an initial battery level of one per cent for the sake of shorter simulation time.

4.2.1. Lifetime without fire

In this section we give the results of PEDD and T-MAC for no disaster case. In this case, after network initialization and neighbor list formation, the only packets that are exchanged with neighbors are *alive* packets that are generated very infrequently.

The average network lifetimes for PEDD and T-MAC are shown in Figure 4.1 where FND denotes first node death. It is observed that since PEDD is designed

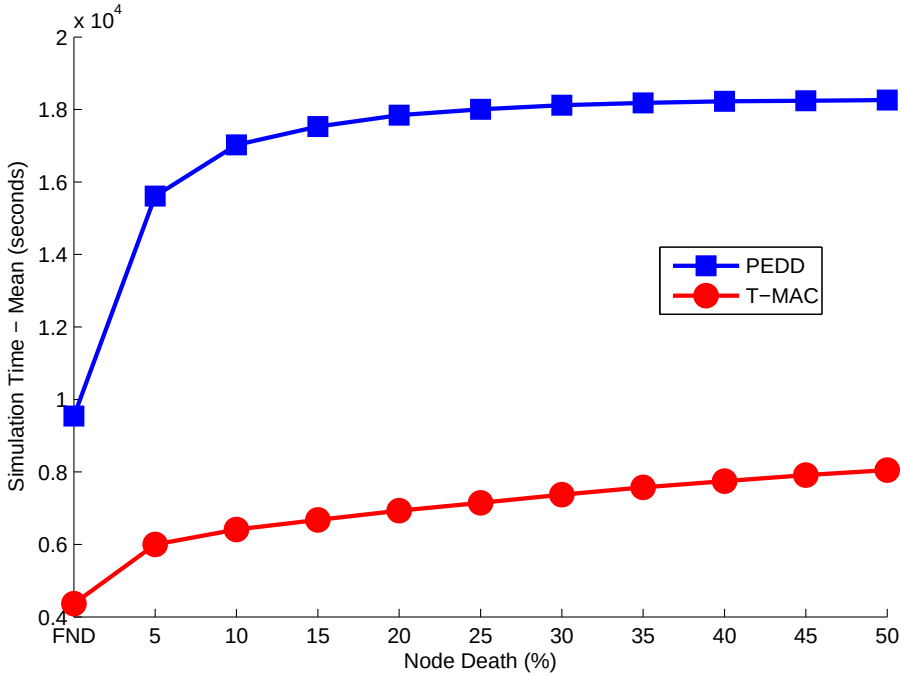


Figure 4.1. Mean network lifetime without fire

for rare event detection, it is superior to T-MAC when there is no event. Moreover, PEDD balances energy consumption during regular operation. This can be seen from the figure where all node death times are close to each other after five per cent node death. On the other hand, T-MAC follows approximately a linear model, since it does not balance energy consumption throughout the network. In addition, our algorithm is scalable in terms of energy expenditure. For an application that requires even longer network lifetime, K_n or $RA_{frequency}$ may be increased to achieve that goal.

Standard deviation for network lifetime for both algorithms is given in Figure 4.2. The effect of balancing energy expenditure can be seen in this figure. T-MAC has higher standard deviation that causes less predictable network lifetime. Predictability is an important criteria in disaster detection, since deploying new sensors early is more costly and more harmful to the environment due to sensor batteries. On the other hand, performing redeployment late may make the ROI more vulnerable. Lower standard deviation makes it easy for the designer to decide on the redeployment time.

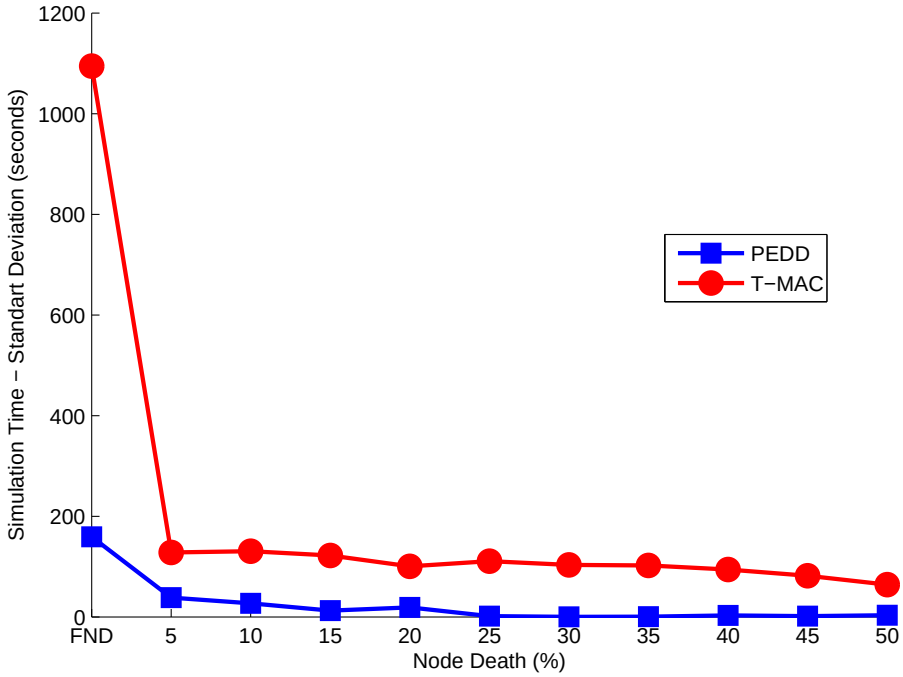


Figure 4.2. Standard deviation of network lifetime without fire

Average number of dead sensors for both algorithms is shown in Figure 4.3. As can be seen from the figure, most sensors suddenly die in T-MAC, whereas sensor death in PEDD is distributed over time. PEDD owes this feature to its ability to balance energy expenditure more efficiently. Nodes survive for similar durations, and then, large number of nodes die in a relatively short period of time that favors our argument.

Average of total energy consumption for a node over ten runs is given in Figure 4.4. It is apparent that both curves are linear, and T-MAC has a steeper curve indicating it results in higher battery consumption.

Statistics about *alive* packets are given in Table 4.5. T-MAC outperforms PEDD in terms of latency of *alive* packets, which is expected. Since alive packets do not contain data that is of great importance, their large latency will not constitute a problem. That is why during the design of our scheme, we trade off their latency with conserving energy. We believe that benefit of conserving energy outweighs delays in the detection of some individual sensors.

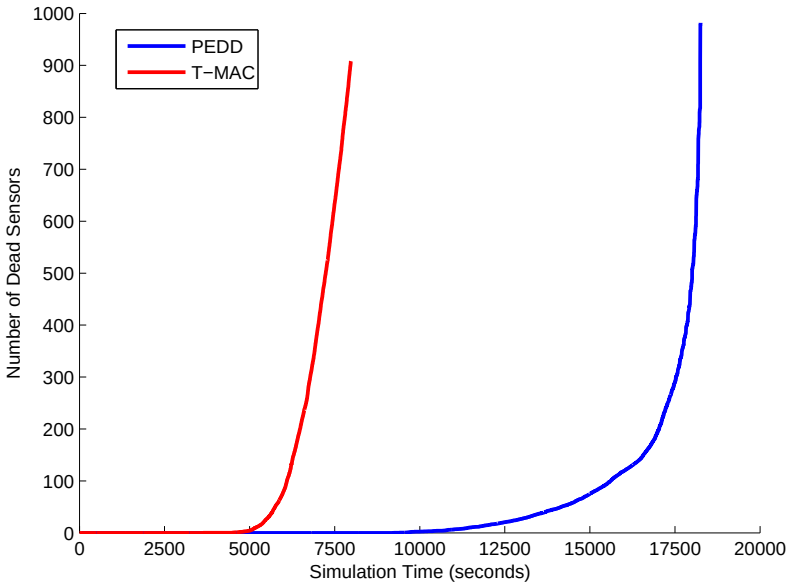


Figure 4.3. Average number of dead sensors without fire

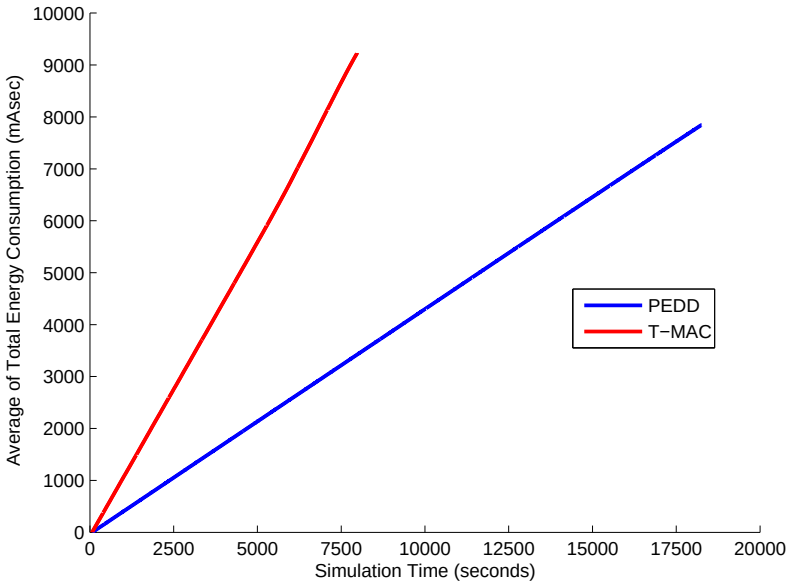


Figure 4.4. Average of total energy consumption for a node without fire

Table 4.5. Latencies of *alive* packets without fire in seconds

	PEDD	**T-MAC**
Mean	362.01	13.44
Standard Deviation	340.76	60.16
Minimum	0.78	0.31
Maximum	3312.83	1988.73

4.2.2. Lifetime with fire

After analyzing network lifetime without fire, we move on to the case where a fire occurs. We start the fire just after initialization phase and it exists until the end of simulation. We simulate three different cases.

4.2.2.1. Case 1. In this first case, we start a fire in the middle of the field. The parameters of the fire are given in Table 4.6

Table 4.6. Parameters for fire for case 1

x Coordinate for Both Ellipses	500
y Coordinate for Both Ellipses	500
Radius at x Axis for *core*	50
Radius at y Axis for *core*	40
Radius at x Axis for *shell*	100
Radius at y Axis for *shell*	80

Average network lifetimes for both algorithms in this case are given in Figure 4.5. The death time of the first node is not given in the graphs since it occurs just at the start of the fire. PEDD again proves to be superior to T-MAC as can be seen from the figure. Reasons for this can be stated as follows:

- Although, we simulate T-MAC operation using data aggregation, in-network aggregation is not as efficient as aggregation performed by group heads.
- The routing mechanism used by T-MAC consumes too much energy since T-MAC always tries to send the data to the same actuator by using similar paths. On the other hand, PEDD allows nodes along the path benefit from duty cycling.
- Four-way handshaking in T-MAC brings an overhead.

The reader should also note that in a case where the path to the nearest actuator is blocked due to the fire, reporting node fails to send its data to the actuator in T-MAC. Moreover, since it may not know that the path to nearest actuator is blocked, it

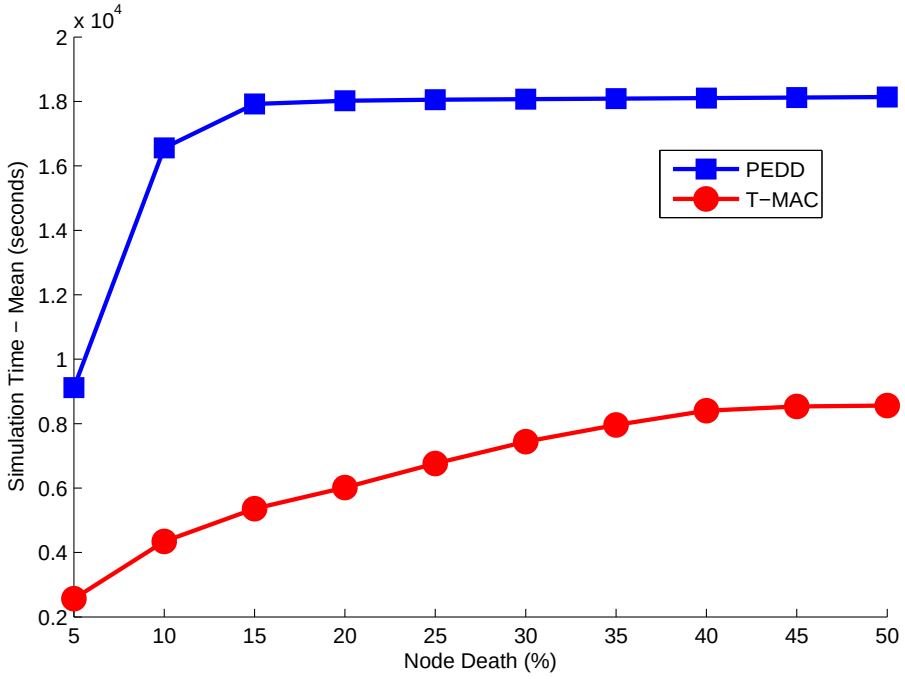


Figure 4.5. Mean network lifetime with fire for case 1

keeps transmitting its data that results in useless energy expenditure and traffic load to the network. On the other hand, PEDD tries to find paths not just to the closest actuator but to all actuators so that the alarm and information packets are received even in the case of a "blocking" fire.

Standard deviation of network lifetime for both algorithms for the case with fire is given in Figure 4.6. Even though, our algorithm's standard deviation is large at first, it diminishes rapidly. The initial large value can be attributed to the fact that five per cent and 10 per cent node death times are much larger than those of T-MAC, thus the variation is also larger.

Average node deaths are given in Figure 4.7. Again, our algorithm distributes energy expenditure and most sensors die at the very end of the lifetime of the network.

Average of total energy expenditure for a node for all runs is shown in Figure 4.8. Both algorithms produce linear curves, especially after the start of the fire. Again, PEDD maintains a smaller slope, implying its efficiency in saving energy.

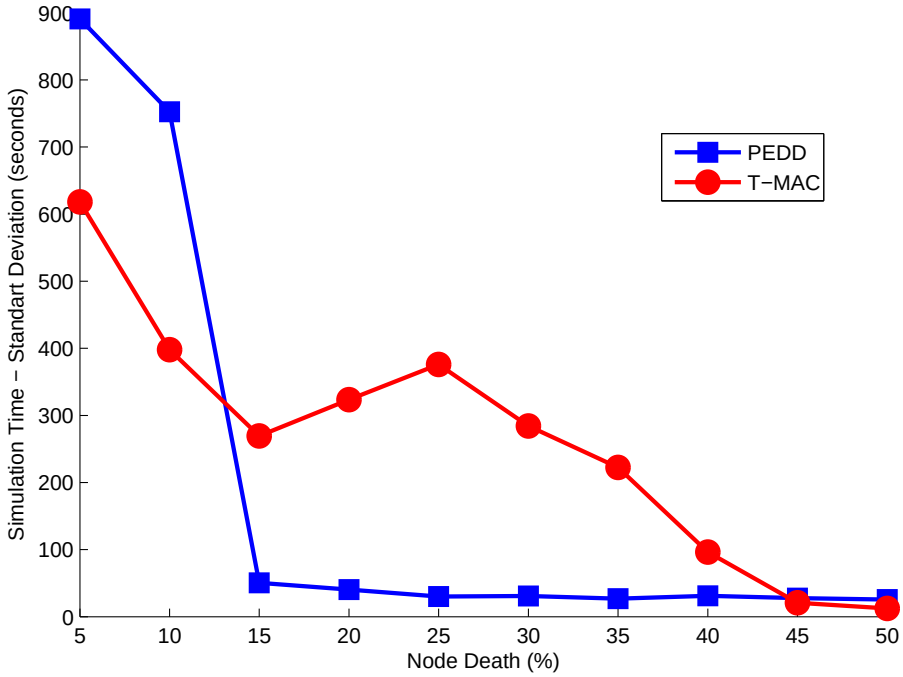


Figure 4.6. Standard deviation of network lifetime with fire for case 1

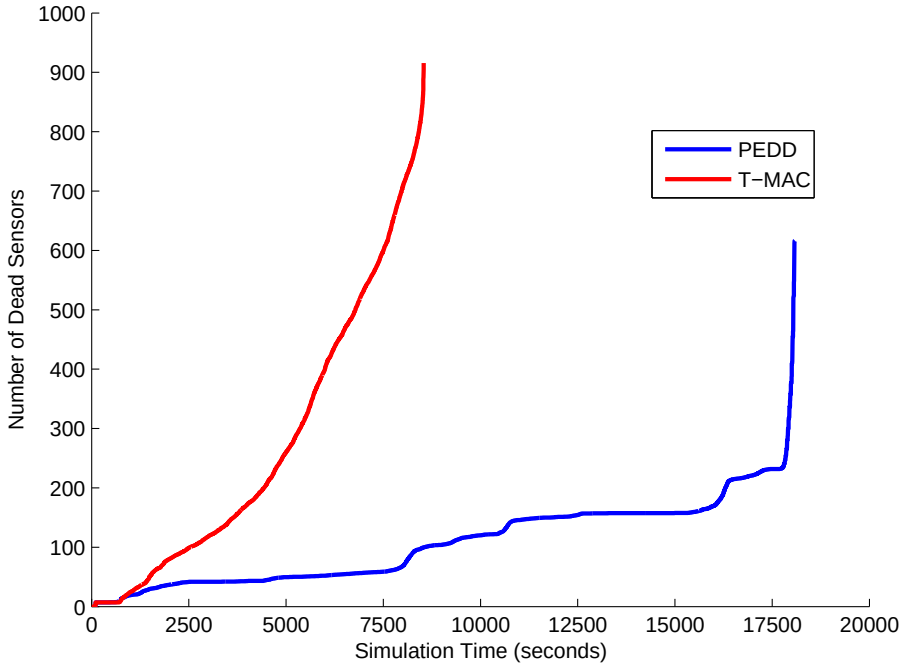


Figure 4.7. Average number of dead sensors with fire for case 1

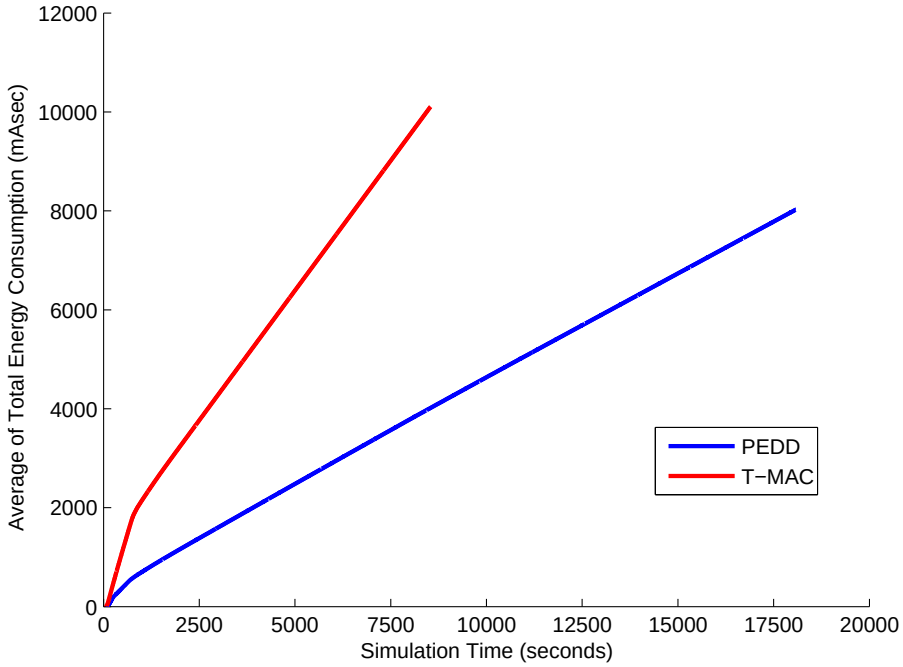


Figure 4.8. Average of total energy consumption for a node with fire for case 1

We do not incorporate alarm mechanism for T-MAC; an actuator concludes there is fire whenever it receives a *report* message. Response times for informing the actuators about the fire are given in Table 4.7. It is apparent that our algorithm has a large response time for informing the actuators about the fire. This can be attributed to just using RA slots for sending the *alarm* packets. Although, it can be decreased by decreasing the period of RA slots or using not only RA slots but also normal slots, we do not favor this approach to keep PEDD simpler. Moreover, for a slowly spreading disaster like fire, a mean response time of 55 seconds is not a large value. For more mission critical applications,response time may be decreased by trading off with energy saving.

Table 4.7. Response times in seconds for case 1

	PEDD	**T-MAC**
Mean	55.18	16.79
Standard Deviation	7.67	0.17
Minimum	45.26	16.69
Maximum	65.56	17.15

To see the efficiency of grouping, we collect statistics about group heads. The statistics about the number of group heads is available in Table 4.8. It can be observed that the number of group heads does not vary too much, showing the robustness of the scheme.

Table 4.8. Statistics about number of group heads for case 1

Mean Number of Group Heads	7
Standard Deviation of Number of Group Heads	0.66
Minimum of Number of Group Heads	6
Maximum of Number of Group Heads	8

Statistics about battery level of selected group heads are given in Table 4.9. Battery levels given in this table are percentages of the full battery level.[1] Since we start the fire just after initialization, group head energies are close to full battery. Very low standard deviation stems from the fact that out algorithm balances energy expenditure and early start of fire does not allow very much differentiation in battery levels.

Table 4.9. Statistics about battery level of group heads just after becoming group heads (percentage of full battery) for case 1

Mean Battery Level	84.9
Standard Deviation of Battery Level	0.032
Minimum Battery Level	84.8
Maximum Battery Level	85.1

Group head statistics related to the sensed value and number of nodes in a group are given in Tables 4.10 and 4.11, respectively. Mean value of sensed value is close to $\frac{t_{max}+t_{thr}}{2}$, which is 75 in our case. However, after observing the tables, one may see that there are group heads that are very close to the *core* and also group heads with no other sensors in their group. After carefully analyzing the data, we observe that these nodes are the same nodes that are beyond the reach of the other group heads. Although, their calculated $f(t_{sensor}, p_{current})$ value is very high, they eventually become

[1]By full battery, we mean one per cent of the total capacity of the battery. It is explained in Section 4.1 that we start with one per cent of the battery capacity for the sake of shorter simulations.

group heads since they are not covered by any group heads. We consider these nodes as outliers.

Table 4.10. Statistics about sensed value of group heads in ℃ for case 1

Mean Sensed Value	80.52
Standard Deviation of Sensed Value	10.18
Minimum Sensed Value	54.27
Maximum Sensed Value	98.78

Table 4.11. Statistics about number of nodes in a group for case 1

Mean Number of Nodes	5.41
Standard Deviation of Number of Nodes	2.90
Minimum Number of Nodes	1
Maximum Number of Nodes	12

Statistics for route setup delay for PEDD are given in Table 4.12. Route setup takes a rather long time but the benefit of having routes can be seen in Table 4.13. Although route setup procedure causes a long delay initially, it significantly reduces latencies of *report* packets. Moreover, this setup delay occurs just after the grouping process once, whereas *report* packets are sent periodically during the fire, which justifies the route setup delay. Finally, path formation allows nodes along the path to perform duty cycling, which results in energy savings.

Table 4.12. Route setup delay for PEDD in seconds for case 1

Mean	86.67
Standard Deviation	25.39
Minimum	49.00
Maximum	142.51

4.2.2.2. Case 2. This time we double the parameters of the fire. By doubling, we mean radius of both *core* and *shell* are doubled on both x and y axes. The parameters of fire for this case can be seen in Table 4.14.

Table 4.13. Latencies of *report* packets with fire in seconds for case 1

	PEDD	**T-MAC**
Mean	4.63	23.42
Standard Deviation	0.34	18.61
Minimum	3.78	6.99
Maximum	5.33	524.75

Table 4.14. Parameters for fire for case 2

x Coordinate for Both Ellipses	500
y Coordinate for Both Ellipses	500
Radius at x Axis for *core*	100
Radius at y Axis for *core*	80
Radius at x Axis for *shell*	200
Radius at y Axis for *shell*	160

Average network lifetimes for both algorithms is shown in Figure 4.9. Since the parameters for fire are doubled, five per cent node death occurs just at the start of the fire. The pattern in the figure is very similar to the one for the first case. T-MAC resembles a linear pattern whereas PEDD balances energy consumption effectively and has a higher network lifetime. This can be observed from the figure where node death times are very close to each other after 20 per cent node death.

Standard deviation of network lifetime is shown in Figure 4.10. Standard deviation of PEDD is higher for 15, 20, and 25 per cent node death times but it diminishes and stabilizes rapidly. Since network lifetime of PEDD is larger compared to T-MAC, higher standard deviation values may occur.

Average number of dead sensors can be seen in Figure 4.11. T-MAC resembles a linear pattern. On the other hand, PEDD seems to stabilize number of dead sensors for quite a while. The rapid increase at the end for PEDD shows that nodes die in bursts which is another indication of balancing energy consumption.

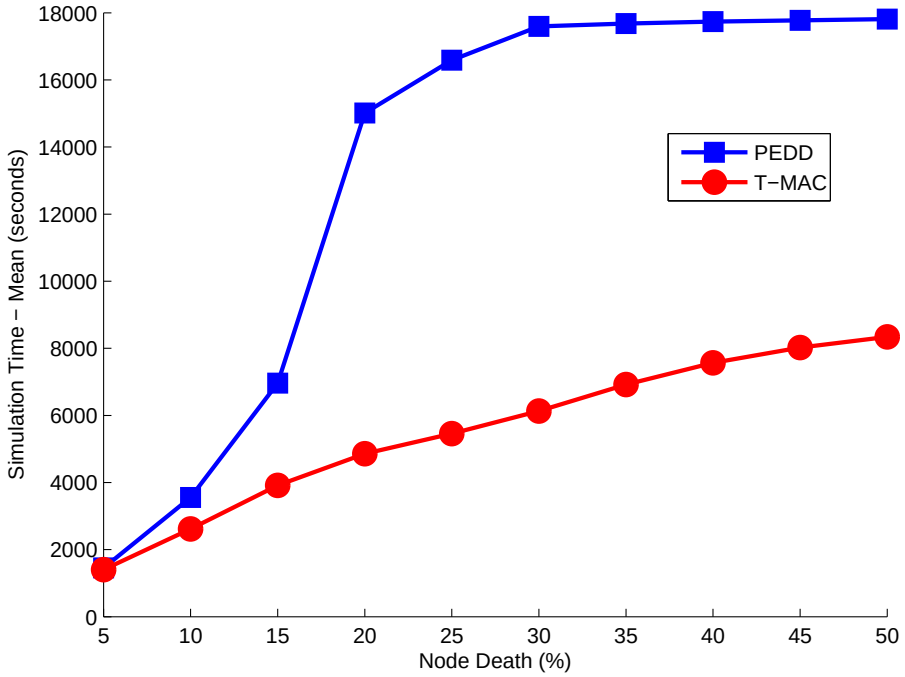


Figure 4.9. Mean network lifetime with fire for case 2

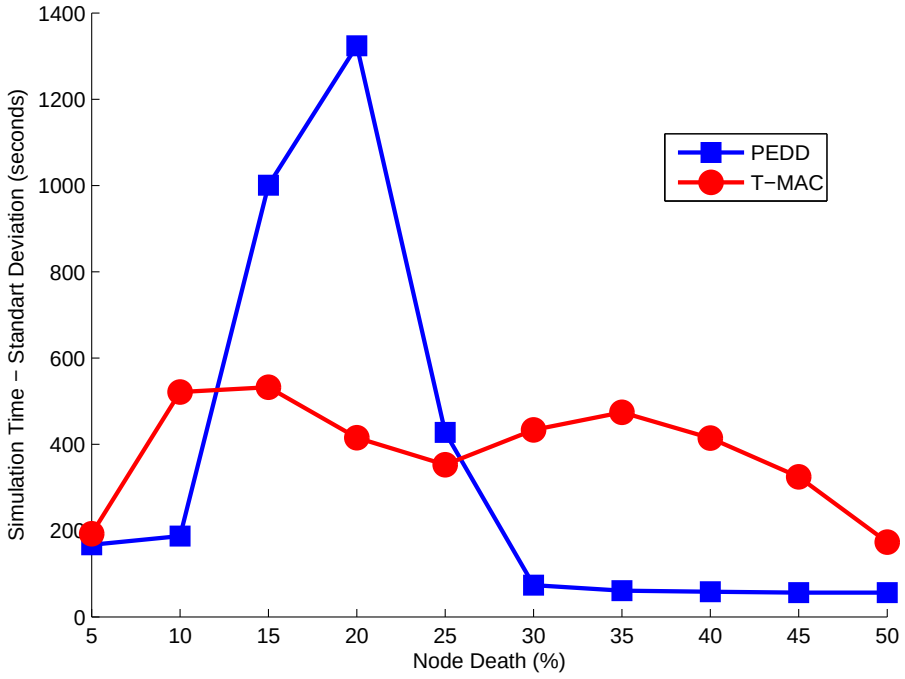


Figure 4.10. Standard deviation of network lifetime with fire for case 2

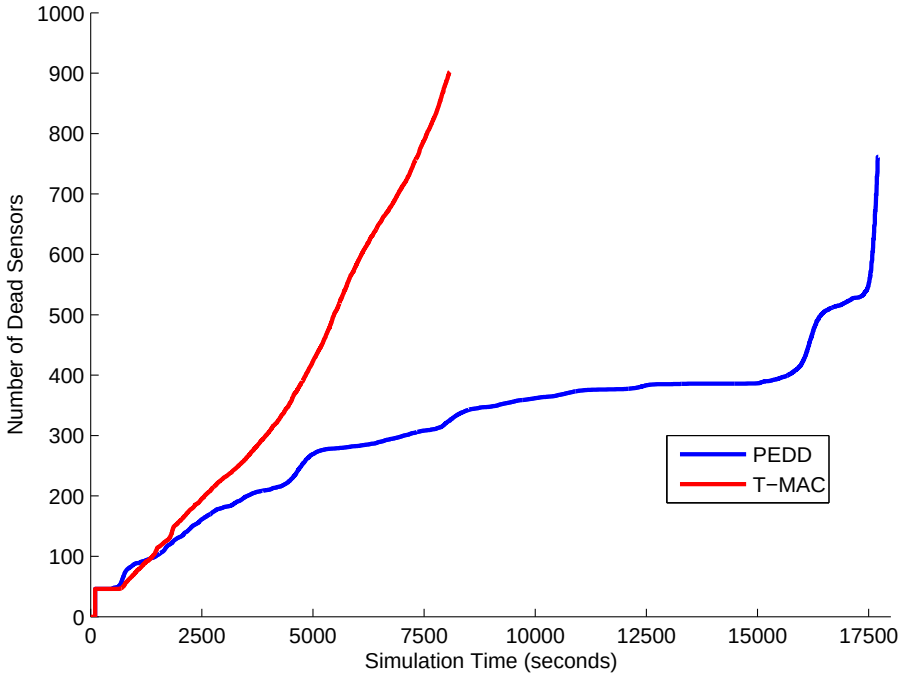


Figure 4.11. Average number of dead sensors with fire for case 2

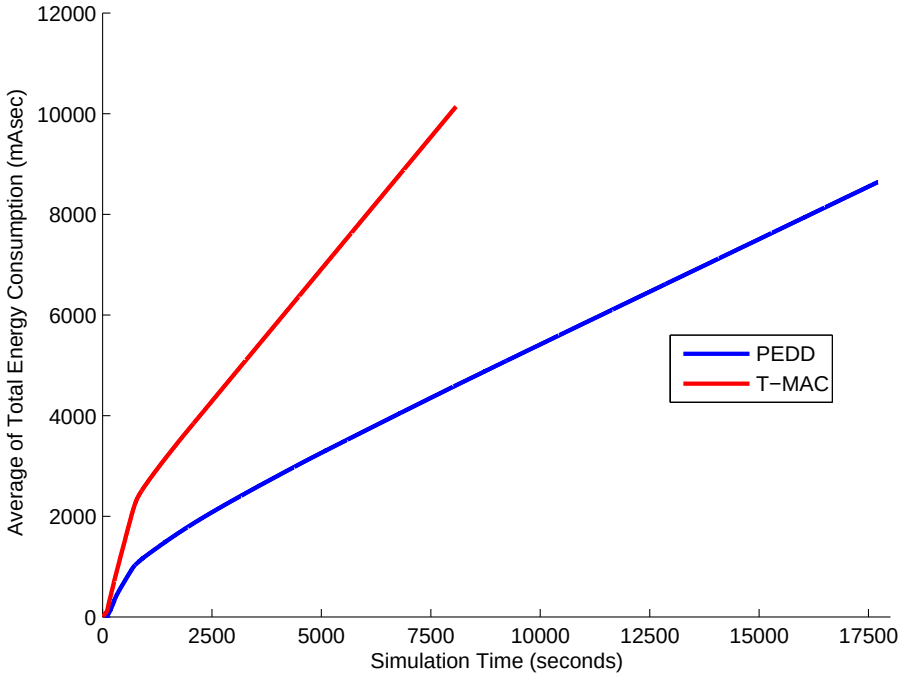


Figure 4.12. Average of total energy consumption for a node with fire for case 2

Average battery consumption for this case is given in Figure 4.12. Both algorithms follow a linear pattern with PEDD having a smaller slope indicating its success in saving energy.

Statistic related to response times for case 2 are given in Table 4.15. Again T-MAC has a better response time. Since we increase the area of the fire, nodes between the *shell* and *core* are closer to the actuator so a decrease in response times is expected. As we can see, PEDD has benefited more from this situation. If we compare this with the response times for the first case, we observe that the margin between the two algorithms gets smaller. If we look at the size of the groups, since group head advertisements are broadcasted to a range of tx_{normal} and sensor nodes are distributed randomly over the area, size of the groups is predictable. Again, there are groups with just a group head due to outliers.

Table 4.15. Response times in seconds for case 2

	PEDD	**T-MAC**
Mean	33.20	13.22
Standard Deviation	3.21	1.35
Minimum	28.99	10.26
Maximum	37.59	14.76

The average number of groups can be seen in Table 4.16. It does not show too much variation and we can say that the average number of group heads is predictable. Statistics about battery levels and sensor readings for group heads are given in Tables 4.17 and 4.18, respectively. Statistics about the size of the groups are given in Table 4.19. As we can see, battery levels are selected from a very narrow interval and group heads are selected from nodes that have their battery almost full. Furthermore, as the disaster area is expanded, the average sensor reading of group heads is very close to $\frac{t_{max}+t_{thr}}{2}$. However, there are group heads very close to the *shell* and the *core* as we observe sensor readings very close to t_{thr} and t_{max}. As stated before these nodes are outliers.

Table 4.16. Statistics about number of group heads for case 2

Mean Number of Group Heads	24.8
Standard Deviation of Number of Group Heads	1.87
Minimum of Number of Group Heads	23
Maximum of Number of Group Heads	28

Table 4.17. Statistics about battery level of group heads just after becoming group heads (percentage of full battery) for case 2

Mean Battery Level	84.9
Standard Deviation of Battery Level	0.003
Minimum Battery Level	84.8
Maximum Battery Level	85.0

Table 4.18. Statistics about sensed value of group heads in ℃ for case 2

Mean Sensed Value	74.48
Standard Deviation of Sensed Value	14.86
Minimum Sensed Value	50.23
Maximum Sensed Value	99.74

Table 4.19. Statistics about number of nodes in a group for case 2

Mean Number of Nodes	6.41
Standard Deviation of Number of Nodes	6.25
Minimum Number of Nodes	1
Maximum Number of Nodes	21

Route setup delay for PEDD is given in Table 4.20. As the disaster area is larger, reporting nodes are closer to the actuators compared to the first case, so some improvement is obtained for route setup delay. If we look at the latencies for *report* packets for this case in Table 4.21, we can see that PEDD is still significantly superior to T-MAC thanks to its pipeline based approach. Moreover, standard deviation is also low compared to T-MAC that causes more predictable packet arrival times.

Table 4.20. Route setup delay for PEDD in seconds for case 2

Mean	75.01
Standard Deviation	17.15
Minimum	43.51
Maximum	98.54

Table 4.21. Latencies of *report* packets with fire in seconds for case 2

	PEDD	**T-MAC**
Mean	4.72	22.93
Standard Deviation	0.39	18.72
Minimum	3.28	5.79
Maximum	5.22	41.72

4.2.2.3. Case 3. In this final case, we designate the fire such that some of the nodes cannot reach their nearest actuator since the fire is blocking the path. Parameters of fire for this case are given in Table 4.22. The size of the fire is the same as in the first case. However, it is very close to SWAN that is located at $(0, 0)$. That is, some nodes on the right of the fire are not able to reach their nearest actuator.

Average network lifetime for this case is depicted in Figure 4.13. Since nodes always select the closest actuator for sending their data in T-MAC, some nodes are blocked by the fire. These nodes spend useless energy. Moreover, these nodes also cause nodes listening the channel to waste energy. This effect can be seen in the figure, where the lifetime of T-MAC is very low. On the other hand, PEDD achieves a similar lifetime to the one in the first case.

Table 4.22. Parameters for fire for case 3

x Coordinate for Both Ellipses	100
y Coordinate for Both Ellipses	80
Radius at x Axis for *core*	50
Radius at y Axis for *core*	40
Radius at x Axis for *shell*	100
Radius at y Axis for *shell*	80

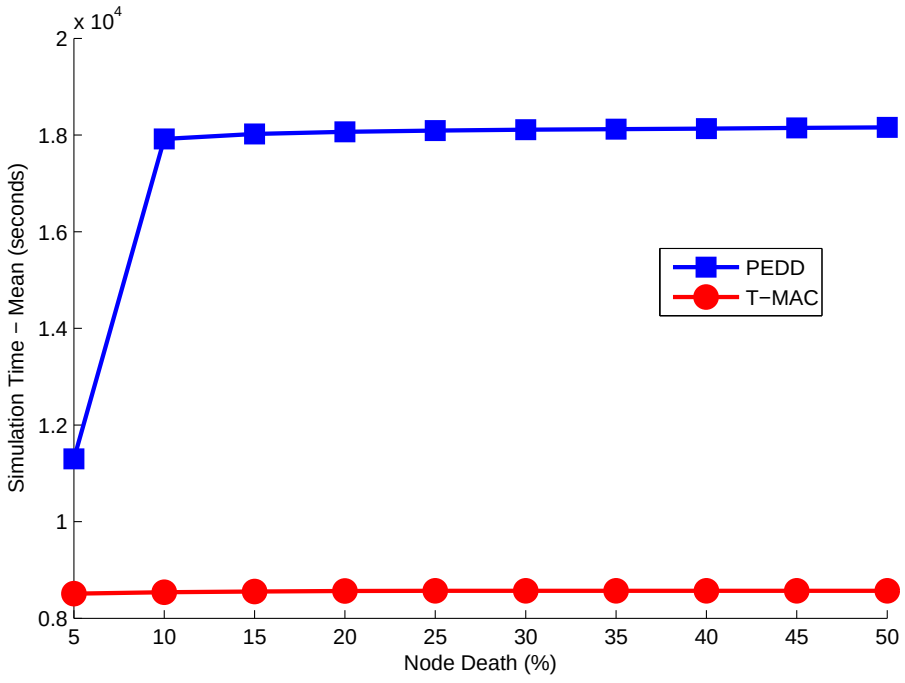


Figure 4.13. Mean network lifetime with fire for case 3

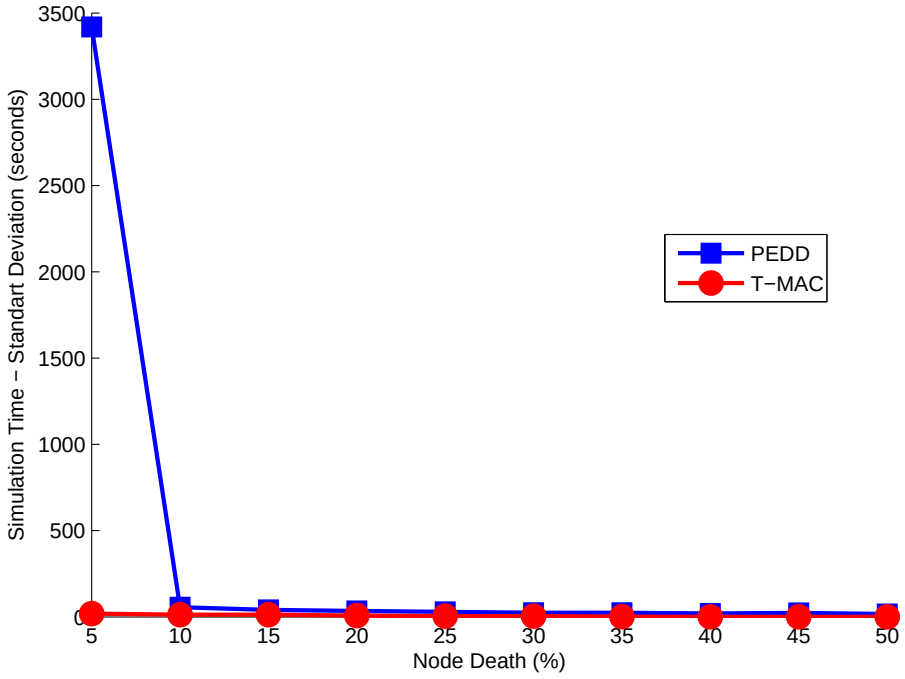


Figure 4.14. Standard deviation of network lifetime with fire for case 3

The standard deviation of network lifetime is shown in Figure 4.14. Although, PEDD has a larger standard deviation at first, it diminishes rapidly just after five per cent node death. In this case, both algorithms have very low standard deviations.

The average number of dead sensors and average of total energy consumption are depicted in Figures 4.15 and 4.16, respectively. Although number of dead sensors for PEDD is slightly higher than T-MAC, useless energy consumption causes nodes to die very quickly in T-MAC. PEDD continues to operate for a longer duration and again most nodes die at the end of the network lifetime. Average of total energy consumption follows a linear pattern for both algorithms with PEDD having a smaller slope compared to T-MAC.

Response times for both algorithms is given in Table 4.23. Since the disaster is very close to SWAN, response times are low for both. Although SWAN is just one hop away for some of the nodes in the disaster area, the larger values for PEDD stems from the fact that PEDD only uses RA slots for *alarm* packets. That is to say, the delay for PEDD includes the time between the sensing of the fire and the nearest RA slot.

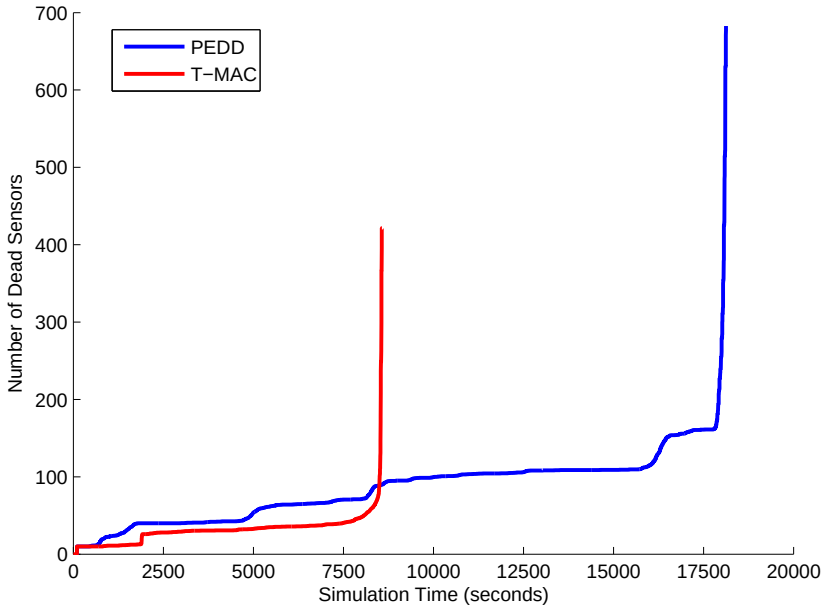


Figure 4.15. Average number of dead sensors with fire for case 3

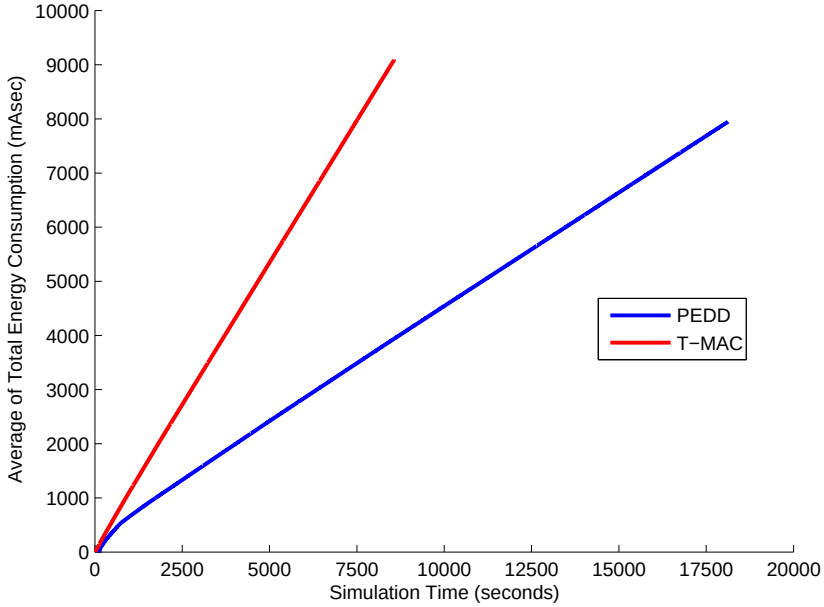


Figure 4.16. Average of total energy consumption for a node with fire for case 3

Table 4.23. Response times in seconds for case 3

	PEDD	**T-MAC**
Mean	2.51	0.78
Standard Deviation	0.27	0.16
Minimum	2.28	0.69
Maximum	2.80	1.13

The average number of groups can be seen in Table 4.24. As in this case, the disaster area is equal to the one in the first case, we expect similar values. Indeed, the values are very close, which shows the robustness of PEDD.

Table 4.24. Statistics about number of group heads for case 3

Mean Number of Group Heads	7.22
Standard Deviation of Number of Group Heads	0.67
Minimum of Number of Group Heads	6
Maximum of Number of Group Heads	8

Battery levels shown in Table 4.25 is similar to the previous cases. Sensed values of group heads is again close to $\frac{t_{max}+t_{thr}}{2}$ as can be seen from Table 4.26. Statistics about number of nodes in groups can be seen in Table 4.24 which shows a resemblance to the first case.

Table 4.25. Statistics about battery level of group heads just after becoming group heads (percentage of full battery) for case 3

Mean Battery Level	84.9
Standard Deviation of Battery Level	0.002
Minimum Battery Level	84.8
Maximum Battery Level	85.0

Table 4.26. Statistics about sensed value of group heads in ℃ for case 3

Mean Sensed Value	78.49
Standard Deviation of Sensed Value	11.34
Minimum Sensed Value	53.77
Maximum Sensed Value	97.40

Route setup delay statistics for PEDD are given in Table4.28 and latencies of *report* packets for both algorithms are given in Table 4.29. Since the disaster is very close to SWAN, some group heads do not need to find routes as they are one hop away from SWAN. The large values for route setup delays stem from the fact that some nodes cannot reach the nearest actuator so they need to find paths to other actuators. As

Table 4.27. Statistics about number of nodes in a group for case 3

Mean Number of Nodes	4.71
Standard Deviation of Number of Nodes	3.39
Minimum Number of Nodes	1
Maximum Number of Nodes	8

the disaster is very far from other actuators, it takes long time to find paths. The same discussion also applies to latencies of *report* packets. Blocked group heads send their data to further actuators and thus, the latency values are high. Even though, T-MAC has smaller latency, some nodes cannot send their packets to the nearest actuator since they are blocked. Thus, some of the disaster area remains unmonitored. Furthermore, they cause useless energy expenditure throughout the network, causing serious effect on network lifetime.

Table 4.28. Route setup delay for PEDD in seconds for case 3

Mean	85.42
Standard Deviation	49.26
Minimum	5.02
Maximum	175.51

Table 4.29. Latencies of *report* packets with fire in seconds for case 3

	PEDD	**T-MAC**
Mean	6.86	2.12
Standard Deviation	7.68	6.64
Minimum	0.28	0.68
Maximum	41.32	237.15

5. CONCLUSION AND FUTURE WORK

In this thesis, a new scheme is proposed for detecting rarely occurring events (such as forest fires) using wireless sensor and actuator networks. The proposed scheme includes network deployment and network initialization in addition to efficient operation before and during disaster. Groups are immediately formed and routes are established when the disaster strikes. The main aim of the scheme is to minimize energy expenditure and overhead to increase lifetime. Increased lifetime offers lower cost and less harm to the environment due to fewer disposal of sensor batteries to the nature.

The algorithm is compared with T-MAC [18], which is a well known and well studied sleep scheduling algorithm in the literature. Comparison is done in terms of network lifetime, energy consumption, packet latency and number of dead nodes for cases with and without fire. T-MAC is complemented where necessary to provide fair comparison. The proposed scheme outperforms T-MAC in network lifetime. Moreover, with our protocol, network lifetime is more predictable which is an important factor for redeployment phase. The properties of the selected group heads are also studied.

The drawback of the proposed scheme is the longer delay in the alarm delay and initial path setup time (up to one and two minutes, respectively). Since the considered scenario is rarely occurring events (like forest fires) where these delays are partly tolerable, delay is tolerated for longer network lifetime (lifetime is approximately doubled). In case shorter delays are required, scheduling parameters may be easily adjusted.

As future work, we plan to work on cases where actuator nodes are randomly distributed throughout the network instead of forming a grid, expanding or shrinking the disaster area, and operation under multiple disasters.

APPENDIX A: PSEUDO CODE FOR ALGORITHMS

A.1. Neighbor Discovery Algorithm

Algorithm 1 Neighbor discovery algorithm

1: **if** $floodPacket.senderID \notin sensor.allNeighbors$ **then**
2: $sensor.allNeighbors \leftarrow sensor.allNeighbors \cup floodPacket.senderID$
3: **end if**
4: **if** $sensor.hopCountSW > packet.hopCount + 1 \wedge sensor.hopCount == 0$ **then**
5: $sensor.hopCountSW \leftarrow packet.hopCount + 1$
6: $sensor.SWneighbors \leftarrow sensor.SWneighbors \cup floodPacket.senderID$
7: $newFloodPacket.hopCount \leftarrow sensor.hopCountSW$
8: Wait for some time
9: $sensor.sendPacket(newFloodPacket)$
10: **end if**

A.2. Alarm Sending Algorithm

Algorithm 2 Alarm Sending Algorithm

```
1: Wait for waitingTime
2: if no activity on the channel then
3:     send alarm packet
4: else
5:     if receivedPacket.type == alarm packet then
6:         cancel my alarm packet
7:         newAlarmPacket ← formNewAlarm(receivedPacket)
8:         if newAlarmPacket == NULL then
9:             exit
10:        else
11:            schedule newAlarmPacket for next RA slot
12:        end if
13:    end if
14: end if
```

A.3. Form New Alarm Algorithm

Algorithm 3 Form New Alarm Algorithm (Part 1)

```
1: if packet.intendedSE is TRUE then
2:     if sensor.hopCountSE < packet.hopCountSE − 1 then
3:         newAlarm.intendedSE ← TRUE
4:     else
5:         newAlarm.intendedSE ← FALSE
6:     end if
7: else
8:     newAlarm.intendedSW ← FALSE
9: end if
10: if packet.intendedSW is TRUE then
11:     if sensor.hopCountSW < packet.hopCountSW − 1 then
12:         newAlarm.intendedSW ← TRUE
13:     else
14:         newAlarm.intendedSW ← FALSE
15:     end if
16: else
17:     newAlarm.intendedSW ← FALSE
18: end if
19: if packet.intendedNE is TRUE then
20:     if sensor.hopCountSE < packet.hopCountNE − 1 then
21:         newAlarm.intendedNE ← TRUE
22:     else
23:         newAlarm.intendedNE ← FALSE
24:     end if
```

Algorithm 4 Form New Alarm Algorithm (Part 2)

25: **else**
26: $newAlarm.intendedNE \leftarrow FALSE$
27: **end if**
28: **if** $packet.intendedNW$ is $TRUE$ **then**
29: **if** $sensor.hopCountNW < packet.hopCountNW - 1$ **then**
30: $newAlarm.intendedNW \leftarrow TRUE$
31: **else**
32: $newAlarm.intendedNW \leftarrow FALSE$
33: **end if**
34: **else**
35: $newAlarm.intendedNW \leftarrow FALSE$
36: **end if**
37: **if** $newAlarm.intendedSE{==}FALSE \wedge newAlarm.intendedSW{==}FALSE \wedge newAlarm.intendedNE{==}FALSE \wedge newAlarm.intendedNW{==}FALSE$ **then**
38: return $NULL$
39: **else**
40: return $newAlarm$
41: **end if**

APPENDIX B: FINDING CLOSEST POINT ON AN ELLIPSE TO A GIVEN POINT

We study the case when the given point lies outside the ellipse. The same analysis also applies to the case when given point lies inside the ellipse. The reasoning is explained below. If the center point of the ellipse is (x_{center}, y_{center}), the ellipse is defined by the equation:

$$\frac{(x - x_{center})^2}{a^2} + \frac{(y - y_{center})^2}{b^2} = 1. \tag{B.1}$$

a and b in Equation B.1 denote the semimajor and semiminor axes of the ellipse, respectively. The length of the x axis is $2a$ and the length of the y axis is $2b$.

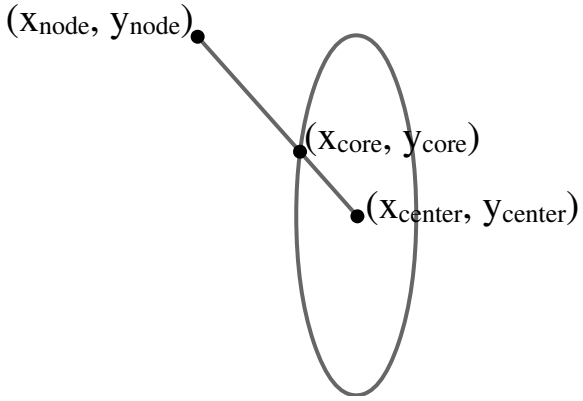


Figure B.1. A simple scenario for finding the coordinates of a point on the ellipse

A simple scenario is given in Figure B.1. (x_{node}, y_{node}) denote the coordinates of the node and (x_{core}, y_{core}) is the point we are looking for. (x_{core}, y_{core}) satisfies the ellipse equation. Moreover, since it is on the line connecting (x_{node}, y_{node}) and (x_{center}, y_{center}), it also satisfies the equation of that line. We represent the line with the equation $y = mx + c$ where m and c are given by Equation B.2 and Equation B.3, respectively. The reader should note that, the only point where we use the coordinates of the node is, for finding the coefficients of the line. Since line equation does not change when the node lies inside or outside the ellipse, we can easily say that the analysis applies to both cases.

$$m = \frac{y_{node} - y_{center}}{x_{node} - x_{center}} \tag{B.2}$$

$$c = y_{center} - m x_{center} \tag{B.3}$$

If we plug (x_{core}, y_{core}) and c into the line equation, we obtain $y_{core} = m x_{core} + y_{center} - m x_{center}$. In addition, (x_{core}, y_{core}) is also on the ellipse so it satisfies the ellipse equation given by Equation B.1. So, if we plug (x_{core}, y_{core}) into the ellipse equation and substitute y_{core} and perform the following steps, we obtain x_{core}. By plugging x_{core} into the line equation, we can easily get y_{core}.

$$\begin{aligned}
\frac{(x_{core} - x_{center})^2}{a^2} + \frac{(y_{core} - y_{center})^2}{b^2} &= 1 \\
\frac{(x_{core} - x_{center})^2}{a^2} + \frac{(m x_{core} + y_{center} - m x_{center} - y_{center})^2}{b^2} &= 1 \\
\frac{(x_{core} - x_{center})^2}{a^2} + \frac{m^2 (x_{core} - x_{center})^2}{b^2} &= 1 \\
(a^2 m^2 + b^2)(x_{core} - x_{center})^2 &= a^2 b^2 \\
(x_{core} - x_{center})^2 &= \frac{a^2 b^2}{a^2 m^2 + b^2}
\end{aligned}$$

$$x_{core} = \begin{cases} x_{center} + \dfrac{ab}{\sqrt{a^2 m^2 + b^2}} & \text{if } x_{node} > x_{center} \\ x_{center} - \dfrac{ab}{\sqrt{a^2 m^2 + b^2}} & \text{otherwise} \end{cases}$$

REFERENCES

1. Zeng-wei, Z., W. Zhao-hui, and L. Huai-zhong, "An event-driven clustering routing algorithm for wireless sensor networks", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 1802–1806, 2004.

2. Akyildiz, I., W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks", *IEEE Communications Magazine*, Vol. 40, No. 8, pp. 102–114, 2002.

3. Yang, X. and N. Vaidya, "A wakeup scheme for sensor networks: achieving balance between energy saving and end-to-end delay", *Real-Time and Embedded Technology and Applications Symposium*, pp. 19–26, 2004.

4. Guo, L. C. Z. C. and J. M. Rabaey, "Low Power Distributed mac for ad hoc sensor radio networks", *IEEE GlobeCom*, 2001.

5. Schurgers, C., V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Topology management for sensor networks: exploiting latency and density", *MobiHoc*, 2002.

6. Shih, E., P. Bahl, and M. J. Sinclair, "Wake on wireless: an event driven energy saving strategy for battery operated devices", *ACM Mobicom*, 2002.

7. Deng, J., Y. S. Han, W. Heinzelman, and P. Varshney, "Balanced-energy sleep scheduling scheme for high density cluster-based sensor networks", *4th Workshop on Applications and Services in Wireless Networks*, pp. 99–108, 2004.

8. Lu, G., N. Sadagopan, B. Krishnamachari, and A. Goel, "Delay efficient sleep scheduling in wireless sensor networks", *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 4, pp. 2470–2481, 2005.

9. van Greunen, J., D. Petrovic, A. Bonivento, J. Rabaey, K. Ramchandran, and A. Vincentelli, "Adaptive sleep discipline for energy conservation and robustness in dense sensor networks", *IEEE International Conference on Communications*, Vol. 6, pp. 3657–3662, 2004.

10. Chachra, S. and M. Marefat, "Distributed algorithms for sleep scheduling in wireless sensor networks", *IEEE International Conference on Robotics and Automation*, pp. 3101–3107, 2006.

11. Paruchuri, V., A. Durresi, and L. Barolli, "Energy aware routing protocol for heterogeneous wireless sensor networks", *Sixteenth International Workshop on Database and Expert Systems Applications*, pp. 133–137, 2005.

12. Paruchuri, V., S. Basavaraju, A. Durresi, R. Kannan, and S. Iyengar, "Random asynchronous wakeup protocol for sensor networks", *First International Conference on Broadband Networks*, pp. 710–717, 2004.

13. Vasanthi, N. and S. Annadurai, "Energy Efficient Sleep Schedule for Achieving Minimum Latency in Query based Sensor Networks", *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, Vol. 2, pp. 214–219, 2006.

14. Ngan, H., Y. Zhu, L. Ni, and R. Xiao, "Stimulus-based adaptive sleeping for wireless sensor networks", *International Conference on Parallel Processing*, pp. 381–388, 2005.

15. Cao, Q., T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection", *Fourth International Symposium on Information Processing in Sensor Networks*, pp. 20–27, 2005.

16. Lu, G., B. Krishnamachari, and C. Raghavendra, "An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks", *18th International Parallel and Distributed Processing Symposium*, 2004.

17. Ye, W., J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks", *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3, pp. 1567–1576, 2002.

18. van Dam, T. and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks", *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 171–180, 2003.

19. Nguyen, C. and A. Kumar, "An energy-aware medium-access-control protocol with frequent sleeps for wireless sensor networks", *10th IEEE Symposium on Computers and Communications*, pp. 386–391, 2005.

20. Rajendran, V., K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks", *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 181–192, 2003.

21. Rajendran, V., J. Garcia-Luna-Aveces, and K. Obraczka, "Energy-efficient, application-aware medium access for sensor networks", *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, 2005.

22. Heinzelman, W., A. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks", *IEEE Transactions on Wireless Communications*, Vol. 1, No. 4, pp. 660–670, 2002.

23. Liang, Y. and H. Yu, "Energy Adaptive Cluster-Head Selection for Wireless Sensor Networks", *Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2005.

24. Musunuri, R. and J. Cobb, "Hierarchical-battery aware routing in wireless sensor networks", *IEEE 62nd Vehicular Technology Conference*, Vol. 4, pp. 2311–2315, 2005.

25. Lazarou, G. Y., J. Li, and J. Picone, "A cluster-based power-efficient MAC scheme for event-driven sensing applications", *Ad Hoc Networks*, Vol. 5, No. 7, pp. 1017–1030, 2007.

26. Wang, L.-C., C.-W. Wang, and C.-M. Liu, "Adaptive contention window-based cluster head election mechanisms for wireless sensor networks", *IEEE 62nd Vehicular Technology Conference*, Vol. 3, pp. 1819–1823, 2005.

27. Younis, O. and S. Fahmy, "HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks", *IEEE Transactions on Mobile Computing*, Vol. 3, No. 4, pp. 366–379, 2004.

28. Huang, H. and J. Wu, "A probabilistic clustering algorithm in wireless sensor networks", *IEEE 62nd Vehicular Technology Conference*, Vol. 3, pp. 1796–1798, 2005.

29. Muruganathan, S., D. Ma, R. Bhasin, and A. Fapojuwo, "A centralized energy-efficient routing protocol for wireless sensor networks", *IEEE Communications Magazine*, Vol. 43, No. 3, pp. 8–13, 2005.

30. Qing, L., Q. Zhu, and M. Wang, "Design of a distributed energy-efficient clustering algorithm for heterogeneous wireless sensor networks", *Computer Communications*, Vol. 29, No. 12, pp. 2230–2237, 2006.

31. Li, C., M. Ye, G. Chen, and J. Wu, "An energy-efficient unequal clustering mechanism for wireless sensor networks", *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, 2005.

32. Ye, M., C. Li, G. Chen, and J. Wu, "EECS: an energy efficient clustering scheme in wireless sensor networks", *24th IEEE International Performance, Computing, and Communications Conference*, pp. 535–540, 2005.

33. Liu, C., K. Wu, and J. Pei, "A dynamic clustering and scheduling approach to energy saving in data collection from wireless sensor networks", *Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pp. 374–385, 2005.

34. Turgut, D., B. Turgut, R. Elmasri, and T. Le, "Optimizing clustering algorithm in mobile ad hoc networks using simulated annealing", *IEEE Wireless Communications and Networking*, Vol. 3, pp. 1492–1497, 2003.

35. Turgut, D., S. Das, R. Elmasri, and B. Turgut, "Optimizing clustering algorithm in mobile ad hoc networks using genetic algorithmic approach", *IEEE Global Telecommunications Conference*, Vol. 1, pp. 62–66, 2002.

36. Krishnan, R. and D. Starobinski, "Efficient clustering algorithms for self-organizing wireless sensor networks", *Ad Hoc Networks*, Vol. 4, No. 1, pp. 36–59, 2006.

37. Ho, S. and X. Su, "CuMPE: Cluster-Management and Power-Efficient protocol for wireless sensor networks", *3rd International Conference on Information Technology: Research and Education*, pp. 60–67, 2005.

38. Shu, T., M. Krunz, and S. Vrudhula, "Power balanced coverage-time optimization for clustered wireless sensor networks", *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, 2005.

39. Liu, J.-S. and C.-H. R. Lin, "Energy-efficiency clustering protocol in wireless sensor networks", *Ad Hoc Networks*, Vol. 3, No. 3, pp. 371–388, 2005.

40. sook Kim, H. and K. jun Han, "A power efficient routing protocol based on balanced tree in wireless sensor networks", *First International Conference on Distributed Frameworks for Multimedia Applications*, pp. 138–143, 2005.

41. Mhatre, V. and C. Rosenberg, "Design guidelines for wireless sensor networks: communication, clustering and aggregation", *Ad Hoc Networks*, Vol. 2, No. 1, pp. 45–63, 2004.

42. Hossain, M., O. Chae, M. Mamun-Or-Rashid, and C. S. Hong, "Cost-effective maximum lifetime routing protocol for wireless sensor networks", *Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/ E-Learning on Telecommunications Workshop*, pp. 314–319, 2005.

43. Jung, E. and D. Walker, "Reliable energy efficient routing in wireless sensor networks", *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, 2005.

44. Pandana, C. and K. Liu, "Maximum connectivity and maximum lifetime energy-aware routing for wireless sensor networks", *IEEE Global Telecommunications Conference*, Vol. 2, 2005.

45. Cheng, H. and X. Jia, "An energy efficient routing algorithm for wireless sensor networks", *International Conference on Wireless Communications, Networking and Mobile Computing*, Vol. 2, pp. 905–910, 2005.

46. Wu, J., L. jia Chen, P. liu Yan, J. guo Zhou, and H. Jiang, "A new reliable routing method based on probabilistic forwarding in wireless sensor network", *The Fifth International Conference on Computer and Information Technology*, pp. 524–528, 2005.

47. Nadimpalli, B., P. Mulukutla, R. Garimella, and M. Srinivas, "Energy-aware routing in sensor networks using dual membership clusters and data highways", *IEEE Region 10 Conference*, Vol. C, pp. 184–187, 2004.

48. Hempel, M., H. Sharif, and P. Raviraj, "HEAR-SN: A New Hierarchical Energy-Aware Routing Protocol for Sensor Networks", *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005.

49. Xbow, *MPR-MIB Users Manual*, USA, 2006.