Game Theoretical Approach for Utility-Based Distributed Balanced Data Routing in Wireless Sensor Networks

by

Afshin Behzadan

M.Sc., Amirkabir University of Technology Tehran, Iran, 2008

A Thesis

Presented to the School of Graduate Studies at

Ryerson University

in partial fulfilment of the

requirements for the degree of

Master of Applied Science

in the Program of Computer Networks

Department of Electrical and Computer Engineering

Toronto, Ontario, Canada, August 2010

Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Author's Signature: [illegible]

I further authorize Ryerson University to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Author's Signature:

[redacted]

Abstract

Game Theoretical Approach for Utility-Based Distributed Balanced Data Routing in Wireless Sensor Networks

©Afshin Behzadan, 2010

Master of Applied Science
Computer Networks Program
Department of Electrical and Computer Engineering
Ryerson University

In this thesis, two distributed algorithms for the construction of load balanced routing trees in wireless sensor networks are proposed. In such networks load balanced data routing and aggregation can considerably decrease uneven energy consumption among sensor nodes and prolong network lifetime. The proposed algorithms achieve load balancing by adjusting the number of children among parents as much as possible. The solution is based on game theoretical approach, where child adjustment is considered as a game between parents and child nodes, in which parents are cooperative and children are selfish players. The gained utility by each node is determined through utility functions defined per role. Utility functions determine the behavior of nodes in each role. At the game termination, each individual node gains the maximum benefit based on its utility function, and the network reaches the global goal of forming the balanced tree. The proposed methods are called Utility Driven Balanced Communication (UDBC) algorithm which is designed for homogenous environment, where all nodes are assumed to produce equal amount of information, and Heterogenous Balanced Data Routing (HBDR) algorithm which is proposed for heterogenous environment, where different applications use different aggregation functions, and nodes can be vary in terms of the amount of produced information, energy levels, data transmission rate and available bandwidth for transmission. The advantage of this work over similar work in the literature is the construction of more balanced trees which results in prolonging network lifetime, with the capability of adaption according to specific application needs for sensitivity to delay and reliability of data delivery.

Acknowledgement

I would like to express my gratitude to my supervisor, Dr. Alagan Anpalagan for his continuous encouragement, tremendous guidance and kind supports throughout this research. It was a great and unforgettable opportunity for me to work with him.

I also would like to thank Dr. Bobby Ma for his advice and supports in various steps of my work. His great and invaluable knowledge and experience was a great help and privilege for me.

I would like to acknowledge the Computer Networks Department and the School of Graduate Studies at Ryerson University for their support in terms of financial aid, and work experience as a graduate assistant.

I would also like to thank my defense committee for taking the time and effort to review my work and provide me with their insightful comments.

In addition I would like to thank the members of RRM+RAN Research Group. Working in such a friendly and productive environment was one of the basis of my progress in this work.

I can never find the words to thank my beloved father and mother, without them I could never reach my current stage in life. Special thanks to my lovely brother and sister, Amir Hossein and Nazanin, my playmates in childhood and the best friends and supporters now. I never felt alone with their kind support and encourages.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Sensor Networks and Query Processing

Wireless sensor networks (WSNs) contain battery powered nodes. Each node consists of one or more sensors for sensing the surrounding environment. Sensors are small and usually inexpensive, and have limited processing resources. Sensors sense and gather information from the environment, based on the decision guidelines provided by users in forms of queries. Acquired data are transmitted, via embedded radio in nodes, among nodes in a multi-hop way and finally reach the root, i.e., the base station [1].

Usually, a WSN has no infrastructure and has an ad-hoc topology. This topology is motivated by the fact that sensor networks are typically deployed in hard-to-access environment. Obstructions in the environment can limit the wireless communication between nodes, affecting the network connectivity. WSNs have many applications, such as military target tracking and surveillance [2,3], natural disaster relief [4], biomedical health monitoring [5,6], and hazardous environment exploration and sensing [7].

Data sensing and transmitting consume a considerable portion of nodes' energy which is limited and vital [8]. The main goal of *query processing* is to answer queries posed by users, while decreasing the energy consumption and prolonging network lifetime are also considered [9]. Queries are declared by users to the network using a SQL-like language, and the network returns the required data by using the query processing software [10]. However, a single query processing assumes one active query in the network. In reality, different users may connect to the base station through the Internet and query various data. In many cases, although different users may have different requests, their requests are somehow similar. Thus, assigning a network to a single query and running queries sequentially not only lead to undesirable delays in responding to user requests, but also results in a vast energy wasting by doing redundant operations corresponding to different queries. Therefore, *multi-query processing* has been considered in which multiple queries run in the network simultaneously [11, 12].

Severe energy constraints and consequently communication and processing limitations of sensor nodes make centralized methods not suitable for these networks, as their execution involve noticeable processing overhead and need a global view of the network. Generating this global view especially when the network is large will lead to a quick energy drain of nodes. Thus, distributed and lightweight methods which are executable by low-powered sensor nodes and that need only local information to run properly, are required [13].

Multi-hop data routing, driven by the above mentioned limitations, is typically used in a tree topology for data delivery, where intermediate nodes receive data from their beneath nodes, apply in-network aggregation on them and their local data, and send the aggregated data to the next hop towards the base station [9, 10, 14]. This topology is formed by issuing a flood from the base station towards the network through which nodes are assigned to different levels. Due to flooding and the location of nodes, each node might receive multiple flooding messages from some other nodes which can be its candidate parents. Thus, each node can have some candidate parents. However, at the end of flooding process, each node selects a parent in one level lower than its level, to which it will send the acquired data [15]. Some children might have the opportunity to select their parents from a set of possible parents. Blind parent selection leads to an uneven distribution of children among those parents. In this case, it is highly probable that some parents have more children while the others have less or even no children. A higher number of children implies a higher amount of traffic at a certain node, since it receives, processes and forwards data from its children. Higher amount of traffic leads to faster energy depletion and the nodes suffering this problem have a shorter lifetime compared to nodes dealing with a less amount of traffic [16]. Thus, uneven distribution of children among parents implies uneven drain of energy and lifetimes for nodes. As the nature of sensor networks is ad-hoc and randomly distributed, the connectivity of the network highly depends on the lifetime of nodes.

Wireless sensor networks are usually modeled via a graph $G = (V, E)$ with set of nodes (G) and their adjacency (E) [17]. As mentioned before, each node has a level equal to its distance to the base station in terms of number of hops. Data routing tree is a tree topology $T = (V_T, E_T)$, $V_T \subseteq V$ and $E_T \subseteq E$ built on a graph G. Using this tree, the acquired data from the network are streamed to the base station. The routing tree is formed when each node selects a closer node to the base station as its parent to send its data. Different policies can be considered for parent selection [18] such as random selection, selection based on distance, or selection based on the quality of the link. However, as the network becomes more dense, consideration of load on nodes looks more important, since nodes with higher load have a faster energy depletion and failure [19]. Here, load is defined as the required energy for receiving, processing and transmitting of data.

One of the advantages of sensor nodes is their processing ability [10]. Although this ability is limited, it gives the opportunity of moving some of the data processing operations into the network. As the energy resources are restricted in sensor nodes, the main goal of in-network data processing is to decrease the amount of communication overhead among nodes. In-network aggregation [20] is the most common case of such processing, where each node aggregates its acquired data with the received data from other nodes, and sends the aggregated data to its parent in the routing tree. Aggregation functions can be divided into two main groups: perfect and imperfect [21]. For perfect aggregations, the number of output data items is constant and independent of the number of input data items. Examples of these kinds of aggregations are maximum and average functions. The former causes one data item to be sent out from each node, and the latter leads to two output data items from each node, i.e. the sum and the number of values. Perfect aggregations can be completely performed in the network. On the other hand, the number of output data items for imperfect aggregation is variable and depends on the number of input data items. Median function is an example of such kind of aggregations. For the computation of a median function, all data should be available. This requirement forces nodes to send all raw data to the base station, where the result can be determined, after these data has been gathered.

While in-network aggregations decrease processing and communicating overheads on nodes, load balancing techniques try to distribute them evenly among nodes [19]. Load balancing tends to distribute excessive loads on nodes among other possible nodes in a fair manner to prolong their lifetime [19] and maintain network connectivity as a global objective. Most of the proposed works in the context of load balancing are accompanied by aggregation techniques. Existing works in the literature which use aggregation along with load balancing, can be divided into two groups. The first group considers perfect aggregation [21 23]. The main feature of this consideration is the equal load sent out from each node. In the second group, different loads from nodes are assumed [21]. This assumption makes the load balancing problem more complicated when dealing with the second group compared to the first group, but it is more practical. In this case, in addition to perfect aggregations, imperfect aggregations can be included, where the load sent out from each node can be different due to the fact that complete in-network aggregations cannot be performed. A perfect solution for heterogenous case should also work for homogenous case as load homogeneity is a specific case of load heterogeneity when the output load of nodes are equal.

Node heterogeneity is not limited to various amount of produced data. Nodes may also differ in the amount of power supply, total amount of data which they can transmit in a time slot, or the total bandwidth that they can allocate for data transmission to their children.

Besides the energy issue, time and bandwidth are two important factors in the quality of data delivery. Thus, the importance of load balancing can mainly be discussed from three aspects: nodes energy consumption, delay and link quality in terms of available bandwidth, all of which are critical for applications in sensor networks, and it would be more realistic if they are considered in the proposed solutions.

1.2 Game Theory

Game theory is a discipline aiming to model situations in which decision makers have to make specific actions that have mutual, possibly conflicting, consequences [24]. It has been used primarily in economics, in order to model the competition between companies. In the field of wireless networks, game theory is often used as a tool for making cooperation between nodes, terminals or various authorities. Game theory is applied to help solving problems in routing, resource allocation and power management [25, 26].

Game theory is related to the actions of decision makers who know that their actions affect each other. A game includes a set of players $i = 1, 2, ..., N$, each of which selects a strategy $s_i \in S_i$, where S_i is a strategy space including all possible strategies for i. Player i has the objective of maximizing its utility u_i. A game can be modeled by a set of players, a set of available resources in the game, all possible choices for each player, which are made possible from the set of resources, and the payoffs assigned to each player after choosing a specific resource.

Two types of games can be considered: in *non-cooperative* games, each player selects strategies without coordination with others. On the other hand, in a *cooperative* game, the players have the choice and try to cooperate with each other, so that they can gain a maximum benefit which is higher than what they could have obtained by playing the game without cooperation [27]. The objective is to allocate the resources so that the total utility is maximized. In wireless networks, the formation of coalitions involves the sharing of certain resources. However, when the costs of such resource sharing is more than the benefits gained by the nodes, they are less likely to participate.

1.2.1 Nash equilibrium

The equilibrium strategies are chosen by the players in order to maximize their individual payoffs. In game theory, the Nash equilibrium is a solution concept of a game involving two or more players, in which no player has anything to gain by changing only his own strategy unilaterally [28]. If each player has chosen a strategy and no player can benefit by changing his strategy while the other players keep theirs unchanged, then the current set of strategy choices and the corresponding payoffs constitute a Nash equilibrium.

1.2.2 Repeated games

In strategic games, the players make their decisions simultaneously at the beginning of the game. On the contrary, the model of an extensive game defines the possible orders of the events. In repeated games, players can make decisions during the game and react to other players decisions. In this case, a game is played many times and the players can check the outcome of the previous game before playing in the next repetition.

A game theoretic approach [29] looks suitable for this purpose, since it considers independent players with local knowledge. This feature supports the nature of sensor environments, in that nodes have only local information due to their limitations. In proposing such an approach, nodes are considered as players of the game trying to reach a higher benefit according to their individual utility functions. On the top level, a global utility function also should be defined in a way that when each player tries to gain more individual utility, it implicitly helps in maximizing the global benefit defined by the global utility function. As each game should have a termination at the end, Nash equilibrium can be used to show that the game will stop at some point. However, Nash equilibrium proves the termination of the game, and not necessarily the optimum outcome resulted by playing the game.

1.3 Motivation and Contribution

This work is motivated by the works proposed in [21,22]. In those works, a decentralized utility-based algorithm for construction of load balanced data gathering tree, where nodes are considered homogeneous, is presented. Although that heuristic is impressive, as mentioned in [22], the algorithm produces optimum balanced trees in around 90% of scenarios. Also, an algorithm in [21] is proposed specifically when in-network aggregation is not possible and each node have to send information from all its beneath nodes without performing any data merging. The solution is based on the cumulative cost from the base station to each node. The cost of each link is defined by the remained energy of nodes. A node chooses a parent to join which has the least cumulative cost among other available parents for that node. The implementation is possible by distributed distance vector algorithm. However, this solution has a problem, since the best parents are selected based on their remaining energy, and with a distance vector algorithm some nodes may attach to a node at a same time because of its best condition. In this case, the fast energy depletion of the attached parent causes more frequent update for re-construction of tree to keep the network connectivity. In this work, two distributed methods for utility-based balanced data routing in sensor networks, referred to as Utility-Driven Balanced Communication (UDBC) and Heterogenous Balanced Data Routing (HBDR), are proposed. Specifically, we investigate the construction of load balanced routing trees, where nodes in each level have a balanced amount of load in terms of energy consumption compared to other nodes in that level, as much as possible.

Utility driven notation is derived from using a game theoretic approach used for the construction of the tree. The solutions are based on the gained utility by each node. Utilities are determined by utility functions which steer node behaviors. These functions are defined based on different roles and in a way that while each node tries to reach its own maximum individual benefit, it implicitly helps the whole network to achieve the global benefit.

UDBC algorithm is a new distributed algorithm is proposed, which is able to produce more balanced trees in different scenarios. Distributed term implies that the algorithm uses just local data of nodes for execution. By adopting the game theoretic approach distrusted condition is met, where each node as an independent player has only access to its local data and data of its adjacent nodes. Also, comparing to the similar work [21,22], construction of tree accomplishes faster, i.e., in an order less than maximum number of candidate children in the adjacency of all nodes in the network. The low order of time complexity is the main reason for low communication overhead imposed by tree construction. Overall, by adding some small informative data items to certain messages and using a new distributed algorithm, UDBC method achieves a better performance in terms of producing more balanced trees, and less time and communication overheads.

The HBDR algorithm benefits from a game theoretic approach to solve the problem of load balancing in heterogenous environment. It considers heterogeneity in terms of traffic amount from nodes, energy levels, bandwidth, and data transmission rate. It also uses a model to work with different kind of aggregation functions, covering perfect and imperfect aggregation. Though the mentioned features are the advantages of HBDR algorithm over UDBC algorithm, the UDBC algorithm has a simpler implementation and requires less computational resources to run.

The main contributions of the proposed work are:

- We address the load balancing problem and formulate it using two utility based approaches.

- We propose two new game theoretical approaches, which satisfy the distributive nature of sensor environments, for balanced data routing in sensor networks. The first algorithm is called UDBC and is a light weight algorithm which produces more balanced trees faster with lower communication overhead for perfect aggregation. The second algorithm, referred to as HBDR, not only takes prolonging network lifetime into account, but also considers delay and bandwidth as quality criteria for balanced routing tree construction.

- We evaluated the performance of the proposed algorithms in terms of adjustment of energy consumption among nodes, time and communication overheads required to achieve load balancing, and adaption to specific requirements of different applications in terms of different aggregation function, data delivery delay and reliability.

- We prove and show that the two proposed algorithms improves balancing of load among nodes and improve the network lifetime, while they meet the mentioned above criteria.

The rest of the thesis is organized as follows: In chapter 2 a review of related existing work in the literature is provided. Problem definition and formulation are presented in chapter 3. Detailed explanations of algorithms are also provided in this chapter. Chapter 4 analytically evaluates the performance of the algorithm and includes simulation results. Finally, chapter 5 summarizes the work and points to some future directions of the research.

Chapter 2

Literature Review

The proposed algorithms in this thesis lie in different arenas, which require us to review the relevant work in the literature. Generally, the load balancing problem can be considered from the routing and query processing point of view, since one of the main purposes of load balancing in wireless sensor network is the data delivery in an energy efficient way. The main fields for this purpose are routing and query processing fields. Routing aims to route the data in the network with the least cost, mostly based on the energy and least number of hops. On the other side, query processing tries to decrease the amount of data which travels through the network. Both of the mentioned fields try to decrease the energy consumption of individual nodes, and has many related concepts associated with each other.

Load balancing also tries to decrease the energy consumption not for individual nodes, but for the whole network. Approaches which rely on load balancing adjust the load. The load usually is defined as the energy consumption due to data communication, among nodes. However, the inseparable parts of load balancing are routing and query processing, since it benefits from the concepts and techniques, but in a balanced way. Thus, three fields should be reviewed and investigated for proposing a solution for the load balancing problem.

When the concept of "utility" is considered for nodes in the sensor networks, utility based approaches will be a relevant field. Also, sensor nodes can be considered as independent agents as they have local information. A utility based approach which models the nodes as independent agents having their own local knowledge, implies using a game theoretic approach with nodes as players. Game theory approaches have recently received attention in different fields of sensor networks such as routing, load balancing and security, as the nature of these networks encourages this tendency. As our work considers a utility-based approach based on game theory concepts a section of this chapter is dedicated to reviewing the use of game theory in sensor networks.

The the chapter is organized as follows. In Section 2.1, the existing concepts and techniques that mostly focus on energy saving are reviewed. As these concepts are aimed at energy conservation, routing and query processing related works in the literature are included in this section. Section 2.2, specifically is devoted to works in load balancing field. Finally, Game theory concepts and their applications in wireless sensor networks is reviewed and discussed in Section 2.3.

2.1 Query Processing in Sensor Networks

Many works have been proposed in the area of query processing and energy conservation techniques in wireless sensor networks. The authors in [14,30] present some fundamentals for processing of queries and discuss some optimization techniques for data gathering. TinyDB [10] and Cougar [31] are two query processors, considered as main references in this area. TAG [20] introduces in-network aggregation for energy consumption decrease.

The following works propose various techniques to increase energy efficiency. The authors in [9] moot some fundamentals for query processing systems and mention some optimization techniques for query plan generation. The proposed work in [32] partitions the network among multiple routing trees to run a single query, to increase performance and fault tolerance. In [33], the authors propose some multi-query processing fundamentals and use a framework to support it. In their work base station (BS) batches and aggregates queries together and injects a new query resulted from them to the network. In the run time if any new query arrives, it will be aggregated by the previous queries, but the new resulted query will be sent as the second query to the network. This is a transient state in which two combined queries run in the network. The transient state continues until the first query finishes, and then only the second query continues to run. However, the proposed solution aggregates all the received queries together and, in cases that queries are not compatible with each other their aggregation will not be beneficial.

The proposed works in [34-36]are also similar to [33] as they introduce some multi-query processing concepts. In [11] the idea of dividing the network into multiple zones and processing of results inside each zone is investigated. Results from each zone are collected and aggregated by nodes inside the zone, and finally just one aggregated data will be sent out. This way, the amount of outgoing traffic from each zone decreases, which leads to a considerable energy conservation. The authors in [37] propose some routing techniques for optimum result transition to BS in multi-query environments. The authors in [38] use some pre-computed partial results, called materialized in-network views, to decrease energy consumption and share network among multiple queries. In their work, some partial aggregation results are computed before running of queries and they are kept in nodes. When the queries run, if a node has the desired value, it returns its partial aggregation value and does not send the query further to its beneath nodes. This way a decrease in communication and computation overhead can be gained. However, computation of materialized views at first can impose a considerable energy overhead, and also in time dependent applications, after a while views are spoiled and cannot be used anymore. The proposed work in [39] supports concurrent run of queries; however, in that work each node has to aggregate received queries and run aggregated query which can impose considerable computation overhead on it.

2.2 Load Balancing in Sensor Networks

Several authors studied balanced data gathering in sensor networks. Harvey et al. [40] propose an optimum centralized method for making an optimal semi-matching which minimizes the variance of the load among nodes. Considering a bipartite graph, a semi-matching is a balanced assignment of vertices in one part to vertices in another part. The graph can be represented as a set of tasks in one part and a set of machines in another part, each is able to process a subset of tasks. Then, an edge exists between a node from tasks part to another node in machines part, if and only if that machine is able to perform the task. Loads on machines are determined by the number of tasks they run. Thus, the goal is to assign tasks to machines in a fair way, so at the end, machines run as equal number of tasks as possible. Their method produces optimum load balanced assignments in all cases. To avoid unbalanced assignments, formation of a kind of path, called *alternating path* in the graph is prevented, as it is a sign of unbalancing.

Due to the limitations of sensor nodes and their autonomous mode, most of the proposed work in the literature tend to be distributed. Sandagopan et al. in [22] and [21] propose a decentralized utility based method for making the balanced data gathering tree. They use a game theoretic approach, where nodes are the selfish players of a tree construction game. Their method is suitable where in-network aggregation can be applied as they consider nodes with similar amount of data. In-network aggregation gives the opportunity to interpret the whole game as a number of games per level of tree. Thus in each level, the game is the balanced joining of children to possible parents in one level higher. Parent selection is done based on bids which are sent by parents to children. Each parent sends the guaranteed bandwidth to its children. Initially, parents determine their bids by c^{-1}, where c is the number of adjacent nodes from the next level or in other words the number of possible children for them. Since children may receive various bids from their candidate parents in their adjacency, they select the sender of best offer, i.e., the highest received bid as their parent. The game terminates when all parents are saturated meaning that the number of children multiplying their guaranteed bids becomes one. If a parent is not saturated, and there is no joining children for it, this means that its possible children currently have a better bid from another parent. Thus, it will increase its bid to $(c-1)^{-1}$ and continue this decrement in each round to either force its children to switch to it or reach a saturation condition. However, this technique may cause a number of switchings by children among parents when different parents increase their bids resulting in considerable communication overhead.

In [21] the authors also discussed the construction of a spanning tree where no in-network aggregation is applied. They assume each sensor node samples and transmits one unit of data per epoch. With this situation, each leaf node transmits one unit of data, while every intermediate node transmit its local data plus data from nodes in the subtree for which it is the root. Assuming each data unit transmission consumes one unit of energy, each leaf node dissipates one unit of data and each intermediate node dissipates energy proportional to size of its subtree. The goal is to construct a spanning tree such that all the sensor nodes are fairly utilized on the average. If $S(T,i)$ shows the size of subtree rooted at node i, and e^i shows its current energy, then the fractional remaining energy of node i is defined as $\frac{e^i - S(T,i)}{e^i}$. To ensure fair utilization of energy resources the goal is the construction of a spanning tree which maximizes the summation of the fractional remaining energy of all nodes. If the network is modeled by graph $G=(V,E)$, the strategy space of S^i_k of each node i in iteration k is

$$S^i_k = \{j | (i,j) \in E\} \tag{2.1}$$

Then, the utility function of each node is defined as follows:

$$u^i_k = \mathrm{Min}_{j \in S^i_k} \left\{ \frac{1}{e^i} + Q^j_k \right\} = \mathrm{Min}_{j \in S^i_k} Q^j_k \tag{2.2}$$

where Q^j_k is the length of the shortest path from sensor node j to the BS at iteration k. $\forall k : Q_{BS} = 0$. Thus, at each iteration, a sensor node i chooses a node j that offers it the shortest path to BS, by using $\frac{1}{e^z}$ as the metric of edge length for any edge $(z, w) \in E$, where $z, w \in V$. This mechanism can be implemented by a distributed distance vector algorithm. Thus, the energy balanced tree construction terminates when the distance vector algorithm terminates.

Authors in [41] address a cluster-based load balancing multi-path routing (CLBM) method for sensor networks. CLBM classified nodes into multi-path routing nodes and cluster routing nodes. The multi-path routing nodes are close to the base station and each time randomly select next-hop nodes based on their next-hop routing tables. The cluster routing nodes are clustered and form a routing tree in a cluster. Cluster heads are selected based on their remaining energy and distance between them, and event center area. Authors in [42] propose a hybrid inter-cluster routing strategy for energy efficient and balanced data gathering called EEDP. In EEDP, each cluster head switches among direct and multi-hop aggregated data forwarding towards the base station. Their hybrid strategy achieves a fair distribution of communication overhead among cluster heads in different areas of a network and increase network lifetime. A load balancing algorithm is presented in [43] for which the current local traffic condition in the network is also taken into account resulting in a quicker data delivery with less collision probability.

As mentioned earlier, different characteristics can be considered for classification of the literature in the context of load balancing. First, the proposed works are either centralized and decentralized (or distributed). The latter is preferred for sensor environment since it is scalable and provides light weight solutions. Next, existing works can be investigated from the heterogeneity aspect. Heterogeneity includes some criteria such as different power, communication range, the amount of traffic from each node, transmission range and etc. Considering the current literature, most of works can be divided into two major groups. The first group considers homogenous nodes, and the second one considers two different types of nodes. One type includes more powerful nodes having more abilities which are deployed in a fewer number. The other group includes a higher number of nodes with less power and capabilities. While consideration of homogenous nodes suggests flat network structure, heterogeneity in the mentioned form suggests a hierarchy for the network structure formed by clustering in which and powerful nodes act as cluster heads and the rest constitutes nodes inside clusters. In the following, based on this brief discussion, some related works are briefly discussed.

Ma et al. [44] considered the problem of load balancing in the hybrid sensor networks. In this hybrid sensor network nodes are divided into two groups of static nodes and cluster heads nodes. The network mostly consists of static nodes which sense data from the environment. Cluster heads are less in number, but are more powerful, mobile and organize clusters including static nodes. Beside scheduling and controlling decisions, cluster heads act as a data aggregation point for clusters. In this scheme, static nodes deliver data in a multi-hop way to cluster heads, but can hear queries directly from cluster heads because of wider transmission range of cluster heads. As cluster heads and static nodes are different in terms of their power and capabilities, the proposed scheme considers heterogeneous networks. Considering mobile cluster heads, the problem is to position them for load balancing in the network. By proposing a heuristic for this purpose, they show that moving the cluster head to a better location can prolong network life time up to 35%.

Chu et al. [45] proposed a centralized approach for dynamic monitoring of nodes' energy to determine if changing the parent of a node is necessary. The process includes two procedures in the gateway. In the first procedure, priority of each node is determined. It starts from the node with heaviest nodes and repeats for every node. The number of parents is used for breaking ties, meaning that the node with the least parents is the winner. The energy of selected node's parent is compared to the energy of other nodes from lower layer in its communication range. The link among node and its parent is remains intact unless the energy of parent is less than the other nodes from the lower layer in its communication range. In that case, the second procedure starts and a new parent with the least load is selected. The least number of neighbors is used for breaking ties in this procedure.

Huang et al. [46] proposed a routing protocol to build a Minimum Balanced Tree (MBT) which is load balanced and dynamically adjustable. Their protocol is distributed and considers homogenous nodes. For the purpose of tree construction, each node periodically checks whether the status of its neighbors suggests any shorter path in terms of hop counts to the sink node. If such a path can be found it will reselect its parent and reconstruct its sub-tree. Minimum neighbors criteria are used for breaking ties in this case. The authors showed that the difference of degree among parent nodes which would be the candidate parents, is always less than or equal to one. Tree adjustment process occurs in the case of a node failure or existence of new node.

Based on ant colony algorithm [47], Hu et al. [48] introduced a distributed algorithm for constructing the data routing tree with nodes' residual energy consideration. Nourizadeh et al. [49] presented a distributed cluster based adaptive routing protocol which dynamically adapts to node's failure and mobility. One of the features of the mentioned protocol is load balancing by using fuzzy logic. Load is the sum of quality of links among a node and its children. Link quality is defined as the reliability between a node and its parent. Daabaj et al. [50] proposed a load balancing routing algorithm in which parent selection is based on the residual power of the intermediate node and the channel state, and the hop count as a third tier-break factor. Tree construction is performed in three stages: route setup, data transmission, and route maintenance.

Above brief review provides some instances of how current works focus on the load balancing problem. Most of the present works assume homogenous network, which is not practical. In reality, deployed nodes vary in terms of residual energy, transmission range and bandwidth. Even if the initial deployment uses similar nodes, nodes are not the same after awhile as each has different amount of processing and communication overhead based on its location in the network. Thus, node heterogeneity should be considered in the real implementation of load balancing algorithms for sensor networks.

2.3 Application of Game Theory in Sensor Networks

When a rational interaction is considered between nodes, the forwarding nodes can be encouraged to cooperate in data routing. Then, based on a price-based scheme nodes try to benefit other nodes to receive more benefits from them in a corporation scheme. In this section, the discussion of game theory in wireless sensor networks is provided by surveying recent work on routing among nodes, which relies on price-based or utility-based approaches and use concepts of the game theory.

The authors in [51] use game theory to analyze the outcome of a game, in which the deployed sensors belong to different authorities and can receive incentives for cooperative forwarding for routing, data storage and aggregation. When sensors request a service from another sensor belonging to a different authority, the other sensor may choose to cooperate or decline based on its resources. In this game, where none of nodes from different domains decline to cooperate with nodes from another domain, the Nash equilibria results in non-cooperation of nodes belonging to different sponsors. To avoid this situation, the authors propose the use of tokens as incentives to encourage cooperation between sensors which belong to different sponsors. Two organizations A and B deploy sensors $\{s_{A1}, s_{A2}, ..., s_{Ak}\}$ and $\{s_{B1}, s_{B2}, ..., s_{Bk}\}$, on a rectangular grid consisting of $2K$ nodes. Each sponsor pays the nodes of the other sponsor at the end of a time period T. When a sensor node a belonging to one sponsor requests a cooperation from a node b of another sponsor, it sends a request with the token. If node b cooperates, it receives the token; otherwise, node a keeps the token. The utility of a node is a function of the number of its received and provided cooperations and the total number of requests it made. The utility for a sponsor is the sum of the utilities of all its sensors plus the monetary transfer received by it at the end of T. Using this setting where the sponsors can program their nodes for cooperation, the authors state and prove the existence of various Nash equilibria for various conditions of acceptance or rejection of contract.

The proposed work in [52] can be also considered in the field of approaches that use the game theory in sensor networks, as it models the load balancing problem by using techniques from game theory to make a routing tree in sensor network by a decentralized way. The authors design the utility functions of individual nodes such that the network objectives are met when the sensors maximize their individual utility functions. The problem here is to construct a routing tree rooted at the base station. Every node has a level in the network which is the number of hops from the sink node. A node must find and attach to a parent with fewer children than the current one. The decisions taken by a node at every level are independent of the decisions taken by nodes located at other levels. They describe a distributed algorithm to design the utility functions of individual nodes, such that when these utility functions are optimized by the sensor nodes, the overall objective of the network is met.

A reliable query routing scheme is proposed in [53], where it is suggested that the number of sensors working simultaneously to collaborate on aggregation should be chosen such that global objectives are achieved. The global objectives are defined as increasing network utilization, communication efficiency and energy consumption. For this purpose the authors use a game-theoretic approach. In this approach, sensors are modeled as intelligent agents cooperating to find optimal network architectures that maximize their payoffs. Payoff for a sensor is defined as benefits resulted from its action minus its individual costs. The problem is modeled as a reliable routing, in which a set of sensors are the players of the routing game. When the base station sends a query to the nodes, it is checked for a match with the attributes of the data sensed by the node. They model this idea by a value v_i that represents the closeness of the match. If $v_i = 0$, it implies that the query does not match any attributes. Data is routed to the sink node through an optimally chosen set of sensors. They call this game as the reliable query routing (RQR) game. Each sensor node is modeled as relaying a received data packet to only one neighbor and hence forms only one link between any pair of source and destination nodes. The strategy space of a sensor node is modeled in the form of a binary vector, $\{l_{i1}, l_{i2}, ..., l_{in}\}$, where $l_{ij} = 1/0$ represents decision of a sensor node s_i

to send or not to send a packet to sensor node s_j. Each node's payoff is a function of the reliability of the path between it and the base station and the expected value of information at that node. This results in a routing tree that is optimal, since if a sensor node decides to choose a different neighbor on another tree, it results in reduced payoffs to/from other nodes. Hence, this also forms the Nash equilibrium for PQR game. Since the network is unreliable, they use a path metric called the *path weakness* to evaluate various suboptimal paths. The path weakness determines how much the node would have gained by switching from its current path to an optimal one. A negative switching suggests that a node s_i is benefiting more from its given strategy, but at the expense of some other sensor. A positive switching indicates that the sensor node could have performed better.

In [54], the authors consider the problem of packet forwarding in wireless sensor networks using game theory. In classical game theory, players choose a particular strategy in response to strategies of other players and this strategy does not change over time. However, in [54] the frequency with which a player chooses a given strategy varies over the time in response to the strategies chosen by other players. This allows players to choose from a set of actions and strategies and use of only local information. The authors assume a heterogeneous sensor network from different classes, where any two non-neighboring classes communicate via multi-hop routing. Nodes can be selfish. They consider inter-class relaying, when a class can cooperate and forward packets or decline to relay. They model the game as that of non-cooperative repeated N-player game between classes of nodes, where nodes participate repeatedly in games with other nodes. In repeated games, a node's action in a given round is influenced by the actions of other nodes and corresponding payoffs in previous rounds. Thus, a repeated game offers ways to punish nodes that do not cooperate by decreasing their payoffs at the end of the game. This can be done by making bad reputation or decrease in incentives resulting in reduced payoffs at the end of the game. Cooperation is similarly rewarded, by examining the payoffs after repeated rounds of the game. Nodes with richer history of cooperation have better reputation, accumulate incentives faster and are included in routes. In transmitting or forwarding a packet, classes spend battery energy b and gain an incentive c. If classes refuse to retransmit, they gain nothing and there is no cost to them. They show that for packet forwarding between stationary classes, Nash equilibrium is achieved if each player plays the Patient Grim strategy [54] and the discount factor is approximately close to unity.

In general, to the best of our knowledge, the investigation of node heterogeneity in the literature is mostly limited to two aspects such as nodes with different load, or two kinds of deployed nodes in the network with different capabilities like different communication ranges for each type. Considering two different types of sensor nodes in the network is a big help towards network clustering and localization of data processing and communicating in most of nodes. However, uniform distribution of powerful nodes acting as cluster heads among other normal nodes is a challenging problem in practice, especially in situations where human interference in not possible. In such cases some cluster heads may have larger number of nodes in their clusters, while other have less, and the existing load balancing algorithms cannot achieve a good performance. Even, it is possible that a collection of nodes cannot find a nearby cluster head, and have to connect in multi hop way to the closest cluster head. Increasing number of cluster heads relative to number of normal nodes is a solution, but it inevitably ends up with more expensive outcome. Thus, considering load balancing among all nodes looks important. This is also true in large clusters which may be considered as small networks.

Also, it looks very interesting if an algorithm can adapt itself according to different application needs. While some applications are more sensitive to delay, for some others more quality even with the cost of more delay is preferred. Thus, just consideration of prolonging the lifetime of the network is not enough and a proper solution should consider application needs too.

By considering pros and cons of the reviewed works, two decentralized utility-based approaches for the load balancing data routing are explained in details and evaluated in two later chapters. These algorithms are decentralized algorithms with the main goal of prolonging network lifetime by balancing load among nodes. Because the sensor nodes can be considered as independent interacting entities in the network and due to the advantages of game theory and its compatibility with the nature of sensor networks, two proposed approaches in this thesis are utility-based with the benefit of using game theory concepts.

The first algorithm is a Utility-based Distributed Balanced Communication (UDBC) algorithm. It is simple, fast and accurate algorithm for generating load balanced trees in homogenous environments. The seconde one is Heterogenous Balanced Data Routing (HBDR) algorithm. It is specifically designed for heterogenous environments, although it can support homogenous environment when all nodes are considered the same. In the assumed heterogenous environment, nodes are different in terms of the amount of their produced traffic, energy resources, bandwidth and transmission rate. HBDR algorithm is more complicated and produce less balanced tree is some cases in homogenous environments, compared to UDBC algorithm, but it can support all kind of aggregation functions by adopting a solution to model them. However, HBDR algorithm can be adapted based on time sensitivity or quality (in terms of bandwidth) sensitivity. Also, while it can be used for flat networks to cover homogenous case by considering all nodes with same capabilities, it can still be used for hierarchical networks, where clustering is applied. In the latter case, cluster heads are nodes with more power, or higher transmission rate and bandwidth. By considering these capabilities for some nodes in the network, they are automatically considered as cluster heads, and other normal nodes will tend to join them in a higher priority than other normal nodes.

As mentioned in chapter 1 proposing of two different algorithms for homogenous and heterogenous environments is useful, as many applications rely only on perfect aggregation functions, and load variety of nodes does not happen for them. Thus, deploying a fast and simple, but accurate method for those cases is beneficial. On the other hand, for the other applications which may have different kinds of queries along with different kinds of deployed nodes, the design of a pervasive method is necessary.

Chapter 3

Load Balancing Algorithms: Game Theoretical Approach

In this chapter two algorithms for balanced data routing in wireless sensor networks are explained in details. Both approaches are based on are based on game theoretic concepts by defining of utility functions for nodes as players of the games. Nodes' actions are based on utility function so that they gain more benefits according to utility function. The most important of advantage of using game theory, is that nodes are viewed as indepndent entities having their local knowledge, which facilitates the distributed implementations of algorithms. Two algorithms are proposed for homogenous and heterogenous environments which referred as UDBC and HBDR algorithms. To study and discuss the mentioned algorithms, this chapter is organized as follows: Section 3.1 provides the definition and formulating of load balancing problem in two contexts of homogeneity and heterogeneity of nodes. Section 3.2 contains definitions and assumptions used in two algorithms. As both algorithms run on a communication infrastructure, the construction process of this infrastructure is explained in section 3.3. UDBC algorithm is studied in section 3.4. Finally, the detailed description of HDBR algorithm is provided in section 3.5.

3.1 Problem Definition

A wireless sensor network can be modeled as a graph $G = (V, E)$ where sets V and E correspond to nodes in the network and the connections between them respectively. An edge $e_{ij} \in E$ exists between nodes $v_i, v_j \in V$, if they are located within each other communication range. Due to similar communication ranges, edge e_{ij} is considered as a bidirectional link which can be used for both transmitting and receiving purposes. Streaming of data from nodes towards the base station is supported by a routing infrastructure made by flooding mechanism starting from the base station. During this flooding, each node may accept several flooding messages from other forwarding nodes in its adjacency. However, based on an adopted parent selection policy, it finally selects only one of the sender nodes as its parent. At the end of the parent selection process a topology is formed which can usually be stated by a spanning tree $T = (V, E_T)$ where $E_T \subseteq E$ denotes *parent-child* relationship among nodes. The root of this tree is the base station. There is a variety of policies for parent selection in the literature. However, taking the energy constraint of nodes into account is highly critical for network longevity. From the energy consumption point of view, as sensor networks are deployed in an ad hoc manner, their connectivity is highly dependent on node lifetimes.

As mentioned before, using aggregation functions is highly considered in sensor networks as it decreases the data traffic in the network. Two major groups of aggregation functions are perfect and imperfect aggregations. A high percentage of applications rely on perfect aggregations. However, in some other applications, queries impose the use of imperfect aggregations, where in-network aggregation cannot be applied on the acquired data. As perfect aggregations are suitable for many applications, and implementation of algorithms based on perfect aggregations are much simpler than imperfect aggregations, an algorithm first is proposed in our work which efficiently works for applications using perfect aggregations. Then, another algorithm is proposed that is more comprehensive and covers all types of aggregations, but is more complicated and impose more computational overhead.

For the case of perfect aggregation the following assumptions are made. Each node transmits β data units per time unit, where β is a constant. This assumption is possible because of using perfect aggregations, such as maximum function leading to one data item transmitted from each node, or average leading to two data items sent out from each node. As it is known, transmission and reception operations are two major causes of energy exhaustion in nodes [10, 14]. Thus, we omit energy consumption of computational operations as it is negligible compared to transmitting and receiving energy. As a result, the load on a node in a certain time is a function of data units it transmits or receives. If load on node $v \in V$ with $C(t)$ children at time t is shown by $L_v(t)$, then it can be determined using (3.1):

$$L_v(t) = \beta \times L_v^s + \textstyle\sum_0^{C(t)-1}(\beta \times L_v^r), \tag{3.1}$$

where L_v^s and L_v^r are energy overhead required to send and receive one unit of data. If $\overline{L(t)}$ is the average of load on nodes at time t and n is the number of nodes in the network, then *load factor* corresponding to tree T this time, denoted by $\sigma_T(t)$, is the standard deviation of load on all nodes $v_i \in V$ and is defined as follows.

$$\sigma_T(t) = \sqrt{\frac{\sum_{i=1}^{n}(L_{v_i}(t) - \overline{L(t)})^2}{n}} \tag{3.2}$$

From now on, σ_T will be used as load factor of tree T at termination time i.e., $t = \tau$.

From the game theoretic point of view, the game is the construction of tree $T^* = (V, E^*)$ with $E^* \subseteq E$, where the global utility function would be $\frac{1}{\sigma_T}$. Thus σ_{T^*} should be less than the load factor of every other tree $T = (V, E_T)$. However, maximizing of output for the above mentioned function must result in the best utility for individual nodes as the players of the game. A player can have two roles: parent for one lower level or child for one upper level. For each role, a different utility function is defined. Before describing the utility function, let us describe some notations.

Consider node $u \in V$ and set $P_u \subseteq V$ where $\forall v \in P_u$, u accepts a flooding message.

Then, P_u is the set of potential parents for u, and u is a common child for its members. Also, each $v_1 \in P_u$ is a connected parent via u for each $v_2 \in P_u$. For a node $v \in V$, set C_v including all nodes that accept a flooding message from v makes the set of v's potential children.

Let N_v be a set consisting of node v and its connected parents via members of C_v. As a parent, utility function for node $v \in V$ at time t is shown by $U_v^p(t)$ and defined as:

$$U_v^p(t) = \frac{1}{\sigma_{N_v}(t)}, \tag{3.3}$$

where $\sigma_{N_v}(t)$ is the load factor of members of N_v at time t.

As a child, the utility function can be interpreted as its available bandwidth for data transmission, similar to [22]. This bandwidth is allocated by the node's parent. If for a parent $v \in V$, the total available bandwidth that it can allocate to its children for transmitting their data is 1 unit and it has c children, then a fair bandwidth allocation procedure assigns $\frac{1}{c}$ of bandwidth to each child. More available bandwidth means less delay and more quality in data transmission. Therefore, each node as a child tends to select a parent that offers more bandwidth. If parent of node u is declared by v and its number of children at time t is $|C_v(t)|$, then u as a child, has the utility function $U_u^c(t)$ at time t, which can be determined by (3.4):

$$U_u^c(t) = \frac{1}{|C_v(t)|} \tag{3.4}$$

Since each node can have two different roles and consequently two different utility functions, its total utility function at time t would be:

$$U_v(t) = U_v^p(t) + U_v^c(t) \tag{3.5}$$

Minimization of load factor implies more available bandwidth for children, and more balanced load on parents. In some practical scenarios where applications are real time and in-network aggregations are not possible, query results can not be determined unless all the data are collected at the base station. In such cases, each parent has to send its children results separately. As a result, higher number of children means longer time to wait for sending their received data. Although the solution for the perfect aggregation case cannot satisfy the imperfect aggregation case, still joining to a parent with less number of children have desirable effects as it imposes less delay.

According to the definition of utility functions, it can be inferred that a node in the parent role is a cooperative player which helps its connected parents to justify the number of children. On the other hand, a node in the child role is a selfish player trying to join a parent which has less number of children and provides more bandwidth. These two functionalities together cause the achievement of the global goal which is the creation of an optimum load balanced tree for that the load factor is minimum. Finally, a formal definition of the problem of load balancing when the perfect aggregation is applied and consequently nodes can be considered homogenous in terms off their outgoing traffic, is given as follows:

Given a wireless sensor network, modeled by graph $G = (V, E)$, the game consisting of nodes as player with two different roles (cooperative in parent role and selfish in child role), is to construct an optimum balanced routing tree $T^* = (V, E^*)$, $E^* \subseteq E$, for which the load factor (σ^*_T) is minimized, and the utility gained by each player $v \in V$ at termination time (τ), i.e. $U_v(\tau)$ is maximized.

For the imperfect aggregation case, because different amounts of outgoing traffic from nodes are considered as one of the factors in node heterogeneity, in the following explanations load variety of nodes is assumed. Let $E(v)$ show the initial energy of a node v. A portion of this energy is consumed for sensing and processing purposes and the rest will be used for communication. Again, as the energy required to sense and process data is negligible comparing to communication energy consumption, it is omitted in the computations. A simple and well-known model is used to model radio communication between sensor nodes. Let E_{elec} is the energy needed to operate transmitter and receiver circuit, and ε_{amp} is the required energy to amplify transmission for achieving an acceptable signal to noise ratio. Then, energy required for sending k bits to distance d, and the energy required to receive these k bit can be determined through (3.6) and (3.7).

$$E_{Tx}(k, d) = E_{elec}k + \varepsilon_{amp}kd^2 \tag{3.6}$$

$$E_{Rx}(k) = E_{elec}k \tag{3.7}$$

To find out the value of k, consider $\mathbf{C}_v$ as the set of v's children with $|\mathbf{C}_v|$ members, and each node $u \in \mathbf{C}_v$ sends $L(u)$ bits. Then, $\varepsilon_R(v)$ shows the consumed energy for reception of all data units by v from its children, and can be determined by (3.8).

$$\varepsilon_R(v) = E_{elec} \sum_{i=0}^{|\mathbf{C}_v|-1} L(u_i) \tag{3.8}$$

After receiving data, v performs aggregation function on its local data and the received data. Aggregation functions on k bits of data can be modeled by

$$Agg(\lambda_1, \lambda_2) = \lambda_1 k + \lambda_2, \tag{3.9}$$

where $0 \leq \lambda_1 \leq 1$ is the *compression factor* and $\lambda_2 \geq 0$ is the *overhead factor*. Compression factor shows the average amount of compression obtained by applying the aggregation function, and the overhead factor shows the amount of data imposed by applying aggregation function, regardless of the input data amount. For example maximum or minimum functions can be shown by $Agg(0, 1)$, average by $Agg(0, 2)$, and median by $Agg(1, 0)$.

Now, if v's local sensed data is $S(v)$ bits, and it has $|\mathbf{C}_v|$ children, the amount of data after applying aggregation function, $L(v)$, which will be sent by v is

$$L(v) = \lambda_1(S(v) + \sum_{i=0}^{|\mathbf{C}_v|-1} L(u_i)) + \lambda_2 \tag{3.10}$$

Thus, the energy required to transmit the received data along with its local data is shown by $\varepsilon_S(v)$ and defined as:

$$\varepsilon_S(v) = E_{Tx}(L(v), r) = E_{elec}L(v) + \varepsilon_{amp}L(v)r^2, \tag{3.11}$$

where r is the v's communication range. Considering (3.8) and (3.11), the energy that v spends to receive, process and transmit data is determined by (3.12):

$$\varepsilon(v) = \varepsilon_R(v) + \varepsilon_S(v) \tag{3.12}$$

Then, the remaining energy of v is

$$\varepsilon'(v) = E(v) - \varepsilon(v) \tag{3.13}$$

Also, v's lifetime can be defined as

$$\Delta(v) = \frac{E(v)}{\varepsilon(v)} \tag{3.14}$$

Network lifetime is defined as the time to the first node failure. By this definition, the longer node lifetimes are, the longer network lifetime is.

From the delay point of view, if node v can receive and transmit with rate $R(v)$ data units, the time required to receive data from its children is shown by $T_R(v)$ and computed by

$$T_R(v) = \tfrac{1}{R(v)} \textstyle\sum_{i=0}^{|\mathbf{C}_v|-1} L(u_i) \tag{3.15}$$

Then, v processes the received data and aggregates them with its local data. Assuming that the processing time is negligible, the time in terms of epochs, required to transmitting v's data can be determined by

$$T_S(v) = \tfrac{L(v)}{R(v)} \tag{3.16}$$

Thus, the total delay that v incurs to receive, process and transmit all data is defined by:

$$T(v) = T_R(v) + T_S(v) \tag{3.17}$$

The delay amount should be specifically considered for applications that are realtime and delay is important for them, such as safety or security monitoring systems. A special case can be obtained when each node v has the communication rate of one data unit, $R(v) = 1$, and produces one data unit per time slot, $L(v) = 1$, and the aggregation function is perfect, with function $Agg(0, 1)$. In this case $T_R(v) = |\mathbf{C}_v|$ and $T_S(v) = 1$ leading to the total delay $T_R(v) = |\mathbf{C}_v| + 1$ imposed by v.

On the other hand, when the reliability is more important and some delays can be tolerated by the application, then a proper utilization of bandwidth should be considered. The most common way suggests the division of available bandwidth among children relative to the amount of their sent data. Thus, the available bandwidth for data transmission gained by node u with parent v, shown by $B_T(u)$, is:

$$B_T(u) = L(u)\frac{B_R(v)}{\sum_{i=0}^{|\mathbf{C}_v|} L(u_i)}, \tag{3.18}$$

where $B_R(v)$ is the total bandwidth that v can allocate to its children for data transmission. In the special case, as mentioned before, and when $B_R(v)$ is considered with unit of one, $B_T(u)$ is equal to $|\mathbf{C}_v|^{-1}$.

A combination of delay and bandwidth functions forms the utility function for nodes when they play child role. This combination is able to support different kinds of application needs for delay and reliability where response time and bandwidth have specific importance based on the application. Utility function for child u with parent v is defined through

$$\begin{aligned}\Gamma(v, \varepsilon'(v), |\mathbf{C}_v|) &= U_C(u) \\ &= \varepsilon'(v)\left[F_t \frac{R(v)}{\lambda_1(L(v) + \sum_{i=0}^{|\mathbf{C}_v|} L(u_i)) + \lambda_2} + F_f \frac{B_R(v)}{\sum_{i=0}^{|\mathbf{C}_v|} L(u_i)}\right], \\ & \quad 0 \le F_t \le 1, 0 \le F_f \le 1, \end{aligned} \tag{3.19}$$

where F_t and F_f are time and frequency coefficients correspondingly and indicate the importance of them. Γ shows the bid generated and sent by the parent v, with remaining energy $\varepsilon'(v)$ and $|\mathbf{C}_v|$ children. Each child in a selfish manner tends to join a parent by which it obtains the highest possible utility.

Load factor corresponding to an arbitrary set $\mathbf{S}$ consisting of $|\mathbf{S}|$ nodes is defined as the standard deviation of loads on nodes $v \in \mathbf{S}$, and formulated as follows:

$$\sigma_{\mathbf{S}} = \sqrt{\frac{\sum_{i=0}^{|\mathbf{S}|}(\varepsilon(v_i) - \overline{\varepsilon(v_i)})^2}{|\mathbf{S}|}} \tag{3.20}$$

where $|\mathbf{S}|$ is the number of $v \in \mathbf{S}$ and $\overline{\varepsilon(v_i)}$ is the average of loads on them. For a node v when it plays parent role, the utility function can be defined as

$$U_P(v) = \frac{1}{\sigma_{\mathbf{N}_v}}, \tag{3.21}$$

where $\mathbf{N}_v$ is the set of nodes two hops away in the same level as of v in the graph G including v, or in other words, the set of connected parents to v including v. Minimizing $\sigma_{\mathbf{N}_v}$ leads to maximizing the output of $U_P(v)$ which is the individual benefit for node v in parent role.

As nodes may have both child and parent roles based on their location in the network, the utility function for an arbitrary node v is defined by

$$U(v) = U_C(v) + U_P(v) \tag{3.22}$$

At a top level, minimizing load factor with a set equal to the set of all nodes in the network can be considered as the global goal in construction of routing tree. In this case, no extra load is imposed on any node and all of them have as closest as possible lifetime to each other. Although in this case the lifetime for all nodes is not the maximum possible lifetime, no node suffers from extra load imposed by other nodes. Because the maximum life time for a node can be achieved when it behaves selfishly and does not accept children to join itself. However, this behavior has the cost of joining its possible children to other parents resulting in an extra load on them and shortening their and network lifetimes. For simplicity of explanation $\mathbf{V}_T$ is considered equal to $\mathbf{V}$, though it does not limit the generality of the solution. Thus, the game is to create a routing tree for which $U_{\mathbf{V}}$, defined by (3.23), is maximum.

$$U_{\mathbf{V}} = \frac{1}{\sigma_{\mathbf{V}}}, \tag{3.23}$$

The difference between various nodes in terms of their lifetime, imposed delay and offered bandwidth to their children arises from the number of children. If some parents have extra number of children or loads and some others have less, then based on the mentioned definitions, the delay imposed by them in data transmission and the bandwidth offered by them to their children for data transmission are also uneven. This unbalancing leads to a longer delay and poorer quality for children of nodes with higher load, but less delay and richer quality and even unused resources like energy or bandwidth for children of nodes with lighter load. Thus, adjusting loads among parents is also a critical problem for quality of service guarantee.

Totally, the problem of load balancing with considering of heterogeneity of nodes can be defined as follows: *given a wireless sensor network, modeled by graph $G = (V, E)$, with heterogenous nodes in terms of produced load (L), power supplies (E), and transmission speed (R) and capacity (B) the problem is the assignment of children to parents in a way that load factor of the network ($\sigma_{\mathbf{V}}$) is as little as possible leading to maximum $U_{\mathbf{V}}$, while each individual node $v \in V$ achieves its maximum possible individual benefit ($U(v)$), resulting in* *prolonging network lifetime* (Δ).

3.2 System model and Definitions

For a better description of the algorithms, some definitions and system model are provided in this section.

• $G = (V, E)$: a graph which models the network and includes a set of nodes (**V**) and their adjacency (set **E**).

• $T = (V_T, E_T)$: a tree that models the routing tree, where $\mathbf{V}_T \subseteq \mathbf{V}$ is the set of its constitutive nodes with their adjacency set (E_T) where $\mathbf{E}_T \subseteq \mathbf{E}$.

• *Node Level*: Level of node $v \in V$ in graph G is its distance to the root in number of hops.

• $\mathbf{P}_v^p$: Set of potential parents for node v in graph G, including nodes from which v accepts flooding message during the construction process of graph G.

• $\mathbf{P}_v^c$: Set of potential children for node v in graph G, including nodes that receive flooding message from v during the construction process of graph G.

• $\mathbf{N}_v^u$: Set of nodes which are located in the same level as v and are two hops away from it with the intermediate node u between v and them.

• $\mathbf{N}_v$: Set $\mathbf{N}_v^u$ $\forall u \in \mathbf{P}_v^c$, i.e. all two hops away nodes from v which are located in the same level as of v.

• $\mathbf{E}_v$: Set of children for node v in tree T, called v's existing children.

• $P(v)$: v's parent in tree T.

The construction of tree T is implemented on graph G. The construction of graph G itself is explained in section 3.3. By the end of construction of graph G each node v knows sets $\mathbf{P}_v^p$ and $\mathbf{P}_v^c$.

For homogenous case, where aggregation functions are perfect, nodes are considered similar, in terms of the amount of their generated traffic, and other features of sensor nodes such as initial energy, data transmission range and etc. On the other hand, for heterogenous case, where aggregation functions are considered imperfect, nodes are considered heterogenous in terms of their initial energy, data transmission rate, the amount of their produced data, and the total bandwidth which they can provide their children with, to transmit their data.

The proposed solutions to the mentioned problems consider Stackelberg model for the game. With this model nodes playing parent role are leaders of the game and nodes with child role are followers. Parents have cooperative behavior, while children have selfish behavior. Leaders make decisions before followers, since they can predict decision of followers. The behavior of parents influences the behavior of their children, so that even with selfish actions of children as followers, they still help to the global benefit of the game.

3.3 Construction of Communication Infrastructure, Graph G=(V,E)

As mentioned in the previous section, it is assumed that the graph $G = (V, E)$ exists. Thus, before going through the details of algorithms, the construction process of this graph or in other words infrastructure of both algorithms, is explained in this section. Algorithm 1 shows the construction process of communication infrastructure which is modeled by graph $G = (V, E)$.

The construction is based on the node distances to the base station, and starts with broadcasting a flooding message from this point as the *root* in level 0 of the tree. Receiving nodes of this message set their level to 1, their potential parent to the root, inform the root about their setting and re-broadcast the flooding message. Upon receiving the flooding message by other nodes in the network, each receiver node goes through a series of investigations. Firstly, it accepts the flooding message if its level is not determined yet. By accepting this message, the node sets its level one more than the sender's level, adds the sender to its potential parents set, and informs the sender of this new setting. When a sender of flooding

Algorithm 1 Graph G Construction, run in each node $v \in V$

```
if v is root then
    $v.level \leftarrow 0$
    broadcast $floodingMsg$
else
    if $v.level = null$ and $receive(floodingMsg)$ then
        $v.level \leftarrow floodingMsg.sender.level + 1$
        $\mathbf{P}_v^p \leftarrow floodingMsg.sender$
        send $stateAck$
        broadcast $floodingMsg$
    else if $v.level \neq null$ and $receive(floodingMsg)$ then
        if $floodingMsg.sender.level \geq v.level$ then
            discard $floodingMsg$
        else if $floodingMsg.sender.level = v.level - 1$ then
            $\mathbf{P}_v^p \leftarrow floodingMsg.sender$
            send $stateAck$
        else if $floodingMsg.sender.level < v.level - 1$ then
            remove all members in $\mathbf{P}_v^p$
            send $updateAck$
            $v.level \leftarrow floodingMsg.sender.level + 1$
            $\mathbf{P}_v^p \leftarrow floodingMsg.sender$
            send $stateAck$
            broadcast $floodingMsg$
        end if
    end if
end if
if $receive(stateAck)$ then
    $\mathbf{P}_v^c \leftarrow stateAck.sender$
end if
if $receive(updateAck)$ then
    remove $stateAck.sender$ from $\mathbf{P}_v^c$
end if
```

message receives a related setting message from another node, it adds the sender of the setting message to its potential children set. If a node accepts the flooding message for the first time, it re-broadcasts that message. Due to flooding nature, it is generally possible that a node receives multiple flooding messages from different nodes. Thus, a node may receive a flooding message after determination of its level raising one of the following cases: (i) The sender has a higher level than the level of node's potential parents. In this case, receiving node ignores the received message. (ii) The sender has a level equal to the level of node's potential parents. In this case, receiving node accepts the sender as another potential parent for itself and notifies it by sending a state acknowledgment message. (iii) The sender has a lower level than the level corresponding to potential parents of the node. In this case, receiving node removes all previous parents from its potential parents set and informs them about this action, so they remove the node's ID from their potential children set. Then, it adds the sender ID to its potential parents set as its new potential parent, sets its level one more than the sender's level, informs the sender of this new setting, and re-broadcasts the flooding message. The flooding process continues until all nodes receive this request and do not broadcast it anymore. Eventually, each node $v \in V$ has a set of potential parents ($\mathbf{P}_v^p$) and a set of potential children ($\mathbf{P}_v^c$).

3.4 Utility Driven Balanced Data Communication (UDBC) Algorithm

UDBC algorithm is an algorithm proposed and designed for homogenous case of load balancing problem. Similar to the works in [21,22], the problem of making a load balanced routing tree can be relaxed to independent games in each layer, and each of them can be viewed as assigning balanced numbers of children to each parent. This is possible due to applying in-network perfect aggregation [10,20] where data from children of a certain parent can be aggregated with its local data. In this case, all nodes produce the same loads. For example, if the query is to find the maximum or minimum of some parameters in the network, in a simple scheme each node sends one unit of data corresponding to each parameter. If the query is to find the average, each node can send two units of data per parameter; one is the sum of its local acquired data and its children data, and the other is the number of nodes. However, the proposed method does not provide the optimum solution for queries for which in-network aggregation is not possible, and if one likes including them precisely, has to consider the number of nodes in all levels beneath each node in the network. The solution provided for heterogenous case in the later section considers the mentioned point. An example of the queries for which in-network aggregation is not possible is the median function. A complete study of different kind of queries is provided in [20].

Algorithms 2 and 3 show the process of parent selection for nodes in child and parent roles respectively.

For clarity, the child of a parent in graph G and the child of that parent in the balanced

Algorithm 2 Parent Node $v \in V$

```
 1: if it is initial state then
 2:     broadcast stateMsg
 3: end if
 4: while $e(v) > 0$ do
 5:     broadcast stateMsg
 6:     if at least one joinReq message is received then
 7:         minChild = Min{number of children of connected parents}
 8:         for $i = e(v)$ to max($minChild + 1$, $g(v)$) do
 9:             maxChild ← a potential child which has a potential parent with maximum existing children.
10:             send acceptMsg to maxChild
11:             remove maxChild from the potential children.
12:         end for
13:         send(rejectMsg) to other potential children.
14:     end if
15:     if at least one joinAck message is received then
16:         update(state)
17:     end if
18: end while
```

Algorithm 3 Child Node $u \in V$

```
 1: while $P(u)$ = null do
 2:     if received(stateMsg) then
 3:         update(statesQ)
 4:         if statesQ.size= 1 then
 5:             $P(u)$ ← statesQ[0].sender
 6:         else if at least one stateMsg=<1, 0> exists in statesQ then
 7:             $P(u)$ ← a potential parent which sent this state
 8:         else if it is initial state then
 9:             Broadcast joinReq
10:         else if statesQ.size> 1 then
11:             bestStatesVector ← null;
12:             bestStatesVector ← all states in statesQ with minimum existing children
13:             bestStatesVector ← all states in bestStateVector with maximum common children
14:             send joinReq to senders of states in bestStateVector
15:         end if
16:     else if received(acceptMsg) then
17:         node.parent ← one of acceptMsg.sender
18:         broadcast(joinAck)
19:     end if
20: end while
```

tree T are differentiated. As mentioned earlier, for the former the *potential child* and for the latter *existing child* notations are used. Tree construction is implemented in a number of rounds.

Parent selection process is possible using some informative records, called *state* of nodes. State of node $v \in V$ is presented by a record $<g(v), e(v), c(v)>$ where $g(v)$ initially is the grade of v in graph G defined as the number of its potential children, $e(v)$ is the v's grade in tree T defined as the number of its existing children, and $c(v)$ is the number of v's common potential children with other parents in graph G. The state message ($stateMsg$) from node v includes two state fields, i.e. $g(v)$ and $e(v)$. Each receiving node can determine $c(v)$ by subtracting $e(v)$ from $g(v)$.

Two main phases are considered for the whole game. (i) *Initial round*, where initial states of parents are received by their potential children, and children with one potential parent or best possible parent joins their only possible parent. (ii) *Subsequent rounds*, where all common children gradually and in a balanced way join parents in their strategic space i.e. set of their potential parents.

Since at the initial state, tree T does not exist, field $e(v)$ has the value of 0. Thus, $stateMsg$ of node v at initial state is $<g(v), 0>$ and indicates the start of construction. The construction starts by flooding this state from the root. Upon receiving it, each receiver node broadcasts its initial state to its potential children and starts the execution of the algorithm.

3.4.1 Initial Round

When a child $u \in V$ receives the initial state from only one potential parent $v \in V$, it immediately joins that parent using a *join* message, as v is the only choice for being u's parent. Also, if u receives a $stateMsg$ containing $<1, 0>$ from at least one potential parent it immediately joins one of the senders of such state, since this state indicates that the sender has just one potential child which is u. Thus, by joining to that sender, the maximum bandwidth is guaranteed for u. Upon joining, parent v increases $e(v)$ by 1. On the other hand, if u receives the initial state messages from multiple potential parents $v \in \mathbf{P}_u^p$, where none of them is <1, 0>, then it sends a null join request (*joinReq*) message to all of those potential parents. The reason is that initial states are not the actual states of nodes and its lack of information avoids optimum parent selection by children. By sending the null join request, nodes just inform potential parents of their commonality. Each parent node increases its common children by 1 per reception of such null request. At the end of initial round, each node v has specific $e(v)$ and $c(v)$ where $e(v) + c(v)$ is equal to $g(v)$. From this point, the actual state of each node is formed which will be sent by that node starting the next round. The "potential" notation for a child implies that it can become an existing child for its potential parent or leave it and join another potential parent with better situation. Either a child joins or leaves a parent, it exits from the set of common children of that parent. If a child leaves a potential parent, it goes out from the potential children set of that parent. In the consequent rounds of the game, $g(v)$ can be determined by the sum of $e(v)$ and $c(v)$ for a node $v \in V$.

3.4.2 Subsequent Rounds

According to child utility function defined by (3.4), a child u tends to join a parent that eventually has the minimum number of existing children leading to higher gained utility for the child. For this purpose, the child node considers some priorities for the investigation of fields in the received states from its potential parents. The operator $\gg$ is defined which indicates the priority of its operands. Given two operands a and b, a has higher priority than b if and only if $a \gg b$. In UDBC algorithm, the priority for investigation of fields in different states is $e(v) \gg c(v)$. The behind logic is that, in a greedy manner, a child $u \in V$ chooses parents with minimum number of existing children. Then from these chosen parents, u chooses parents with maximum number of common children or the highest $c(v)$ in their states, since it is more probable some of their common children to join other parents in the future and the utility gained by u to become more. Selected parents after these two steps of investigation are the winners. Then, u shows its tendency to all winner parents by sending *joinReq* messages. A *joinReq* message from child u to parent v, shown by $joinReq_u(v)$, is a record containing three fields $<p_m(u), c_{max}(u), c_{min}(u)>$, where $p_m(u)$ is a potential parent of u, except v, which has the maximum number of existing children, $c_{max}(u)$ is this maximum number, and $c_{min}(u)$ is the minimum number of existing children among u's potential parents, except v. If some of u's potential parents have $c_{max}(u)$ existing children, one of them will be included in the *joinReq* message. Although not including the rest in the messages may cause non-optimum results in a few cases, including one parent ID in the messages considerably prevents uneven child acceptance by parents resulting in almost optimum selections.

If a parent $v \in V$ receives some *joinReq* messages from some potential children, it chooses a child u which has sent the maximum $c_{max}(u)$ among others. The parent utility function explains this cooperative behavior; parent $v \in V$ accepts common children with parent $v_1 \in V$ which has maximum existing children among other connected parents to reduce load factor and help the network longevity. If there are some similar *joinReq*, parent v selects children evenly from their relevant potential parents. This selection is based on the received parent IDs in *joinReq* messages. Child selection continues if at least one child without any parent exists in parent's neighborhood or in other words $c(v) > 0$. However, the upper bound for the number of children that a parent can accept in each round of child selection is one child more than the minimum of $c_{min}(u)$ values received through the *joinReq* messages. The reason for taking the upper bound for the number of children in each round is to avoid a problem mentioned in [40]. The problem is the formation of a path, called *alternating path*, that starts from a parent in one level of graph G and ends to another parent with less number of children in the same level. The intermediate nodes in the path between two endpoint parents can be a series of children and parents with edges in T with direction from parents to children and edges in G with direction from children to parents. Thus, each intermediate child in this path has one predecessor parent which is its current parent, and one successor parent which is one of its potential parents. An alternating path has unbalanced endpoint parents with a difference in number of children larger than 1, leading to unbalanced trees. Unbalanced number of children among two endpoint parents can be balanced when each child in the path switches from its predecessor parent to its successor parent. Taking the mentioned upper bound into account in UDBC prevents formation of such alternating paths. When a parent selects a child to join, it sends an *acceptMsg* message to it indicating the acceptance of its request. However, the parent might have a number of requests more than its upper bound for number of children acceptance in a specific round. Thus, for those requests that have not chosen in a certain round, the parent sends *rejectMsg* message to their senders, showing that their request is rejected for this round.

Upon receiving *acceptMsg* messages by a child node, it attaches to the sender parent and broadcasts a *joinAck* message to not only confirm its attachment to the selected potential parent, but also to inform its other potential parent about its leaving to update their states. All potential parents including the one that the child is joining and others that the child left them, update their states by receiving this acknowledgement. Let x be the number of children that join parent $v \in V$ and y be the number of children that leave v in a certain round, where $x + y < c(v)$. In this case, if $<g(v), e(v), c(v)>$ indicates current v's state, it updates that state to $<g(v) - y, e(v) + x, g(v) - x - y>$. If a child receives multiple acceptance messages from some parents, it chooses one of them randomly. When a change occurs in a parent's state due to the joining or leaving of its potential children, it re-broadcasts its state in the next round of the parent selection. On the other hand, when a child receives an *rejectMsg* in response to its *joinReq*, it waits for the next round of parent selection to choose a parent again.

Based on the position of a node in the network, it can be a leaf or intermediate node. A leaf node has just the child role, but an intermediate node has both child and parent roles. In child role, a node finishes executing the algorithm when it finds a parent, and in parent role, nodes stop running the algorithm when no child exists in its neighborhood without having any parent. Formally, parent $v \in V$ can recognize this situation when in its state $g(v) = e(v)$ or $c(v) = 0$. The game ends, when every nodes in the network has a parent or in the other words for every parent no common child exists.

3.4.3 Illustrative Example

Consider Figure 3.1 as an example, which shows the initial topology for a set of parent and child nodes. The following descriptions and figures illustrate how the game is played.

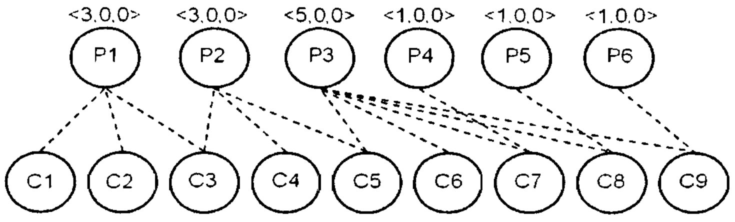

Figure 3.1: The initial topology of two layers in the network. Top and bottom rows indicate potential parents and potential children respectively. The state of each parent node is shown on its top. Dashed lines show node adjacencies.

Round 1: Each parent broadcasts its *stateMsg*. Nodes $C1$ and $C2$ join node $P1$, since $P1$ is the only potential parent from which they receive a state message. Node $C3$ has the same states from parents $P1$ and $P2$. Because this is the initial round, it broadcasts a null *joinReq* message. Node $C4$ only has the state from $P2$, thus it chooses $P2$ as its parent. Node $C5$ has two different states from $P2$ and $P3$, but because of the initial round it also sends a null *joinReq* message. Node $C6$ chooses $P3$. Nodes $C7$, $C8$ and $C9$ join nodes $P4$, $P5$ and $P6$ respectively, since they have *stateMsg* <1,0> from them indicating they are the only child of those parents. Parent nodes update their states based on the number of joining and leaving children. Games ends for nodes $P4$, $P5$, $P6$, $C1$, $C2$, $C4$ and $C6$ through $C9$.

Round 2: Figure 3.2 shows the topology and the node states at the beginning of round two.

Nodes $P1$ through $P3$ broadcast their states, since their states are changed and also the game still continues for these nodes. Node $C3$ sends *joinReq* $<P1, 2, 2>$ to $P2$, since $P2$

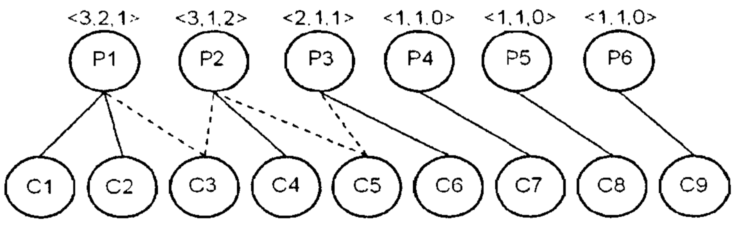

Figure 3.2: The topology of two layers at the beginning of round two of parent selection. Top and bottom rows indicate potential parents and potential children respectively. The state of each parent node is shown on its top. Dashed lines show node adjacencies. Solid lines show parent-child relationships.

has a less existing number of children than $P1$. This message contains the maximum and minimum number of existing children among its other potential parents. Node $C5$ sends *joinReq* including $<1,1>$ to $P2$, since it has greater number of common children than $P2$. Minimum existing children of connected parents in N_{P_2} is 1 which belongs to $P3$. Thus, $P2$ can have maximum number of two existing children at the end of this round. However, it has two *joinReq* messages from $C3$ and $C5$. Thus, it accepts one of them and rejects another. The accepted child is $C3$ and the rejected child is $C5$, since based on its received *joinReq* messages, $c_{max}(C3) = 2$ is greater than $c_{max}(C5) = 1$.

The only join request for $P2$ is from $C3$ that results in $C3$ becoming a child for it. In a similar way, node $C5$ becomes a child for node $P3$. Parent nodes update their states. Game ends for nodes $P1$ through $P3$ and nodes $C3$ and $C5$, leading to termination of the whole game. Figure 3.3 shows the final balanced routing tree formed by the proposed algorithm.

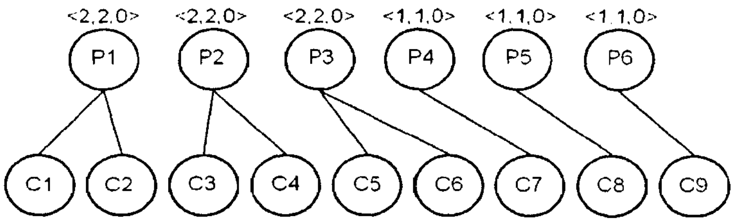

Figure 3.3: The final routing tree formed by the proposed algorithm. Top and bottom rows indicate potential parents and potential children respectively. The state of each parent node is shown on its top. Solid lines show parent-child relationships.

3.5 Heterogenous Balanced Data Routing (HBDR) Algorithm

In this section details for the other algorithm, called HBDR, is explained. The proposed algorithm is not only suited for the case that imperfect aggregation functions are used and nodes are heterogonous, but also works for homogenous case as well where perfect aggregation functions are used. However, HBDR algorithm is more complicated than UDBC algorithm and needs more resources to run. HBDR algorithm includes a series of games, each is played in one level of graph G. These games are played in a sequential order. The first game is the game in the highest level among leaf nodes and their parents. As soon as this game terminates, the second game starts in one level lower, and after its termination, next game corresponding to one level lower can start. This procedure continues until the termination of game in the lowest level, among nodes in the first level and the root that is base station. In fact, playing order follows a bottom-up model from leaves towards root.

Each game includes a number of rounds. In each round some children join parents. Like UDBC algorithm, the first round of the games is called *initial round.* Generally, in this round, nodes gather information about their one-hop neighbors. In the following rounds, referred as *subsequent rounds*, children gradually join parents in a load balanced way. Algorithms 4 and 5 show the process of tree construction.

The first game start from leaves. Based on the information obtained through flooding, each node knows the set of its potential parent and potential children. If a node has just one potential parent, it immediately joins that parent, since the joined parent is its only option. On the other hand, if a node has some potential parents, it sends a join request (*joinReq*) message indicating it is a common children among some potential parents. Each *joinReq* message contains three fields. First field indicates the data amount which was sent by the sender of message. Two other fields, which will be discussed later in this section, do not have any value or in other words are *null* in the first round. Therefore, the *joinReq* message

Algorithm 4 Node $v \in \mathbf{V}$ in Parent Role

```
1: while $|\mathbf{P}_v^c| > 0$ do the following steps
2: if initial round then
3:     wait for all $u \in \mathbf{P}_v^c$
4:     update($\varepsilon(v)$)
5:     $\varepsilon'(v) \leftarrow E(v) - \varepsilon(v)$
6:     if receive($joinReq$) then
7:         update($\varepsilon_p(v)$)
8:         $\varepsilon_p'(v) \leftarrow E(v) - \varepsilon_p(v)$
9:     end if
10:    generate $bid(v)$ and $bid_p(v)$
11:    $bidPair(v) \leftarrow \{bid(v), bid_p(v)\}$
12:    broadcast $bidPair(v)$
13: else
14:    wait for $joinReq$ $timeout$
15:    if receive($joinReq$) then
16:        $nextbid(v) \leftarrow bid(v)$
17:        $oldMinParent \leftarrow null$
18:        boolean $skipFlag \leftarrow$ false
19:        while $nextbid(v) > bid(w)$, $\forall w$ of received $bids$ and
             $skipFlag$ = false do
20:            $minBid \leftarrow$ minimum(received $bid_{Min} \in joinReq$)
21:            $selectedChild \leftarrow joinReq.sender$
22:            $minParent \leftarrow$ the parent which sent $minBid$ to $selectedChild$
23:            if $minParent = oldMinParent$ then
24:                $skipFlag \leftarrow$ true
25:            else
26:                $oldMinParent \leftarrow minParent$
27:                compute $Nextbid(v)$ based on new load
28:                $acceptMsg(v, selectedChild) \leftarrow bid(v)$
29:                send($acceptMsg(v, selectedChild)$, $selectedChild$)
30:            end if
31:        end while
32:        send $rejectMsg$ to the rest requesters
33:        wait for $joinAck$ $timeout$
34:        for all received $joinAck(u, w)$ do
35:            $\mathbf{P}_v^c \leftarrow \mathbf{P}_v^c - \{u\}$
36:            if $v = w$ then
37:                update $\varepsilon(v), \varepsilon_p(v), \varepsilon'(v), \varepsilon_p'(v)$
38:            else
39:                update $\varepsilon_p(v), \varepsilon_p'(v)$
40:            end if
41:        end for
42:        generate $bidPair(v)$
43:    else
44:        wait for $joinAck$ $timeout$
45:    end if
46:    broadcast($bidPair(v)$)
47: end if
```

Algorithm 5 Node $u \in \mathbf{V}$ in Child Role

```
while $p(u) = \phi$ do the following steps
if $initial\ round$ and $v$ is $leaf$ then
    if $|\mathbf{P}_v| = 1$ then
        join($P(v)$)
    else if $|\mathbf{P}_v| > 1$ then
        send($joinReq, \mathbf{P}_v^p$)
    end if
else if receive($bidPair$) then
    wait for all $v \in \mathbf{P}_v^p$
    set BestBids $\leftarrow \phi$
    set BestParents $\leftarrow \phi$
    BestBids $\leftarrow$ received bids with maximum $bid$
    BestBids $\leftarrow$ bids $\in$ BestBids with maximum $bid_p$
    BestParents $\leftarrow bid.sender$, $\forall bid \in$ BestBids
    for all $w \in BestParents$ do
        $bid_{max} \leftarrow$ maximum($bid(v)$), $\forall v \in \mathbf{P}_v^p - \{w\}$
        $bid_{min} \leftarrow$ minimum($bid(v)$), $\forall v \in \mathbf{P}_v^p - \{w\}$
        $joinReq(u,w) \leftarrow \{L(u), bid_{max}, bid_{min}\}$
        send($joinReq(u,w), w$)
    end for
else if receive($acceptMsg$) then
    wait for all involved parents
    $bestChoice \leftarrow$ minimum($bid(v)$) in all received $acceptMsgs$
    $bestParent \leftarrow bestChoice.sender$
    join($bestParent$)
    broadcast($JoinAck(u, bestParent)$)
else if receive($rejectMsg$) then
    wait until next round
end if
```

in the initial round is referred as *null joinReq*, and is used to inform potential parents about their potential children status.

Each parent $v \in V$, updates its load ($\varepsilon(v)$) and potential load ($\varepsilon_p(v)$) based on joining nodes and received *joinReq* messages respectively, using (3.12). In the computation of $\varepsilon(v)$, $\mathbf{C}_v = \mathbf{E}_v$, and in the computation of $\varepsilon_p(v)$, $\mathbf{C}_v = \mathbf{E}_v \cup \mathbf{P}_v^c$. In fact, $\varepsilon_p(v)$ shows the load on v when all its potential children join it, in the worst case. After determining the remaining energy and potentially remaining energy, v computes its bid, i.e. $bid(v) = \Gamma(v, \varepsilon(v), \mathbf{E}_v)$ and potential bid, i.e. $bid_p(v) = \Gamma(v, \varepsilon_p(v), \mathbf{E}_v \cup \mathbf{P}_v^c)$, which it can offer to its children. These bids are broadcasted to its potential children in the form of $\{bid(v), bid_p(v)\}$ pairs, called *bid pair*.

Each child node can receive some bid pairs according to the number of its potential parents. By receiving these pairs, child node $u \in V$ selects parent $v \in \mathbf{P}_u^p$ which offers the highest benefit or bid. Based on the formula for bid, the highest bid from a node implies better quality of service in longer time meaning more stability of the offer than others. If for some parents the offered bids are the same, their potential bids are used for breaking ties. This means from parents with equal offered bids, those are chosen which have the maximum the potential bid. Thus, in the worst case when all potential children join that parent it still has better benefit for u. The chosen parents after these two steps investigation are the best choice for children. Child u shows its willingness for joining those parents by sending *joinReq* messages to them. As mentioned before, a *joinReq* from node u to node v contains three fields. The first field includes the amount of data which u will send to v in each time unit. The second and third fields indicate the maximum and minimum bids respectively among the received bids by u from its parents except v.

A *joinReq* from child u to parent v is an indication about the load on other parent in N_v^u. Based on the third field in the received *joinReq* messages, v chooses a child which has a parent with minimum offered bid indicating a combination of high load and short lifetime for that parent. This cooperative behavior relaxes the load on the second parent and helps it to have a longer lifetime. After choosing u, based on its *joinReq* message, v adds the amount of data indicated as the first field in *joinReq* to its current amount data, and computes the benefit based on this new amount of data showing the benefit in the case that u finally joins v. This predicted bid is called *nextBid*. Also, v sends an acceptance message to u to inform it. Each accept message contains the *nextBid* value. Then, v compares *nextBid* to the second fields of received *joinReq* messages which show maximum bids of other potential parents. If the *nextBid* is larger than all other maximum bids, v continues with child acceptance in a same way as before and recomputes the *nextBid*. Child acceptance continues unless one maximum bid larger than *nextBid* can be found. In that case, v stops child acceptance and sends the reject message to the rest of potential children which request joining.

A child may receive some acceptance messages. By receiving acceptance messages, child u chooses a parent with maximum included *nextBid* in its acceptance message. When u joins a parent, it broadcasts a join acknowledge message to not only inform the joined parent, but also to inform other parents about leaving them.

Let child u joins parent v. Then, u is added to $\mathbf{E}_v$ and also removed from $\mathbf{P}^c_w$, $\forall w \in \mathbf{P}^p_u$. By reception of every *joinAck* message which is destined for v, v re-computes its remaining energy, bid and potential bid, and re-broadcasts the bid pair. From this point and by broadcasting the new bid pair to the its potential children, the next round begins with similar steps mentioned above.

When a child joins a parent the game terminates for it. For a parent v, when $|\mathbf{P}^c_v| = 0$ the games ends. As mentioned before, games are played in a sequence from leaf nodes towards the base station. An intermediate node has two roles: parent for its children in one level higher and child role for its potential parents in one level lower. Thus, it plays two games. Because the sequence of games is from higher level to lower levels, such nodes first play a game in parent role with nodes in one level higher, and after its termination they start playing another game in child role with nodes in one level lower. This way may effect synchronization among nodes in different levels of the graph, as some nodes may finish their first game faster than others, and start the second game sooner. Thus, in the initial round of each game a node in parent role waits to receive null *joinReq* from all its potential children and then each child node waits to receive *bidPair* from all its potential parents. Considering such scheduling helps to synchronization of related nodes to each other and guarantees when a nodes continues the game it has received all necessary data. Let node v_1 waits for v_2. To make sure that v_2 has not failed and is waiting for reception of data from some other nodes, v_1 and v_2 can exchange a Hello messages on some certain intervals to differentiate between node failure and waiting status.

Algorithm terminates when the last games ends, i.e. when nodes in the second level join the parent in the first level.

The performance of two algorithms, are examined in the next chapter through analytical and experimental evaluations.

Chapter 4

Analytical and Experimental Evaluation

In this chapter UDBC and HBDR algorithms are evaluated through analytical and experimental evaluations. Some theoretical aspects of the algorithms are proved through analytical evaluation, such as reaching to Nash equilibrium and some upper-bonds for the time and number of message exchanges needed for trees construction. Experimental evaluation is aimed to investigate the performance of algorithm under the different conditions when some parameters are varying. This chapter is organized as follow: sections 4.1 and 4.2 are assigned to analytical evaluation of UDBC and HBDR methods, respectively. Then, experimental results from conduction of simulations for two methods are provided in sections 4.3 and 4.4.

4.1 Analytical Evaluation of UDBC Algorithm

In this section, validation of some statements about the algorithm are proved first, and then the worst case of some performance parameters are studied.

4.1.1 Theoretical Analysis

Lemma 1: *Given graph $G = (V, E)$, for a node $v \in V$ with state $<g(v), e(v), c(v)>$, $g(v)$ is non-increasing and $e(v)$ is non-decreasing in number of parent selection rounds.*

Proof: According to UDBC algorithm, $g(v)$ changes iff k potential children leave v and join other potential parents. Then,

$$g'(v) = g(v) - k, \tag{4.1}$$

where $g'(v)$ is the new grade of v after leaving. This is the only case when $g(v)$ changes. Therefore, $g(v)$ is non-increasing in number of parent selection rounds.

Also, the number of existing children for v increases from 0 at the initial state to $g(v)$ at the algorithm termination. $e(v)$ changes iff k' potential children join v. Then:

$$e'(v) = e(v) + k', \tag{4.2}$$

where $e'(v)$ is the new number of v's existing children. This is the only change made on $e(v)$. Thus, $e(v)$ is non-decreasing in number of parent selection rounds. □

Lemma 2: *Given graph $G = (V, E)$ and the parent selection game, for a parent $v \in V$ with state $<g(v), e(v), c(v)>$ that receives at least one joinReq message in round r, at least one of its potential children joins or leaves it in that parent selection round. Also, $c(v)$ is non-increasing in number of rounds.*

Proof: Consider the following notations:

P_v : set of potential parents for node v,

P_v^a : set of potential parents that accept v's join request,

P_v^r : set of potential parents that reject v's join request,

r_i : the first (initial) round,

r_s : an arbitrary round in subsequent rounds,

with the following conditions: $P_v^a \subseteq P_v$, $P_v^r \subseteq P_v$, $P_v^a \bigcap P_v^r = \phi$, $P_v^a \bigcup P_v^r = P_v$.

Based on the UDBC algorithm, nodes as potential children send *joinReq* message to a subset of their potential parents in each round, unless they are attached to a parent. Also, for a parent in subsequent rounds there are some children in its potential children set which still do not join a parent. Consider $u \in V$ as a potential child at round r_s. Node u sends *joinReq* to a subset of its potential parent. Upon receiving *joinReq*, some potential parents reject u's request (P_v^r), and some accept it (P_v^a). Node u selects a node $v \in P_v^a$ as its parent. Then,

$$\text{joining } u \text{ to } v \Rightarrow \; e'(v) = e(v) + 1, g'(v') = g(v') - 1, \forall v' \in P_u - \{v\} \tag{4.3}$$

Upon attaching u to v, $e(v)$ increases by 1, and $\forall v' \in P_u$ other than v, $g(v')$ decreases by 1, since u leaves them. Thus, a potential child u joins a potential parent in P_u and leaves the rest in that set. As a result, for a parent v which does not terminate algorithm execution in subsequent rounds, at least one of its common children joins or leaves it per round.

Now, assume that all $v \in V$ reject u's request. This means they have more critical *joinReq* messages from other potential children, and send the acceptance message for them. Then, selected children join them or leave them after acceptance. In both cases number of potential children will decrease for them. Thus, $C(v)$ decreases $\forall v \in P_v^r$.

Number of common children for parent v, $c(v)$, can be determined by $g(v) - e(v)$ in each round of parent selection. Based on Lemma 1, $g(v)$ and $e(v)$ are non-increasing and non-decreasing in number of rounds. Consequently, $c(v)$ becomes decreasing in number of parent selection rounds. $\square$

Lemma 3: *At the termination of algorithm, a Nash Equilibrium exists where neither parent nodes want to get more children, nor child nodes want to switch to another parent.*

Proof: On termination of the algorithm, a parent node $v \in V$ has a final state $<g^*(v), e^*(v), 0>$, where $g^*(v) = e^*(v)$. Switching from v to another potential parent v' with state $<g^*(v'), e^*(v'), 0>$ by a child u is desirable if and only if $e^*(v) - e^*(v') > 1$. In that case, switching form v to v' decreases the load variance factor of both parents and increases the utility gained by u. However, we claim that every potential parent v' of u, has at least $e^*(v') = e^*(v) - 1$ children. This can be proved by contradiction. Assume that v_q is a potential parent of u that has $e^*(v_q) = e^*(v) - 2$ children. Let r_v^* be the final round of parent selection where v terminates algorithm execution. According to Lemma 1, $e(v)$ and $e(v_q)$ are non-decreasing in number of parent selection rounds. Thus, there would be a round $r_v \leq r_v^*$ for v such that $e(v) = e^*(v) - 1$ and v_q has $e(v_q) \leq e^*(v) - 2$. Then, based on the assumption u should join v resulting in $e^*(v)$. However, this is in contradiction with the algorithm logic, where v has an upper bound for selecting a child in each round that is $\text{Min}\{e(v')\} + 1, \forall v' \in N_v$. Therefore, node v can not take additional children more than $e(v_q) + 1$ at round r, and rejects u which joins another parent in the next round. $\square$

Lemma 4 *The routing tree $T = (V, E')$ formed by the proposed algorithm is a local optimum tree in terms of load factor, i.e. $\sigma_{N_v}(\tau)$ is optimum $\forall v \in V$.*

Proof: Different forms of the routing tree can be constructed by replacing edges in T with some other edges in G, e.g. replacing a parent of a child with another potential parent of that child. Let $v \in V$ be the parent of node $u \in V$ at the algorithm termination. According to Lemma 2, all other potential parents of a child u have at least $e(v) - 1$ children. We distinguish potential parents of u with number of existent children equal to $e(v) - 1$ (set P_1) from other potential parents with number of existent children greater than $e(v) - 1$ (set P_2). Switching from v to a parent $v' \in P_1$ does not change load factor of v and v', since in that case $e(v)$ changes to $e(v) - 1$, which is equal to previous $e(v')$ and $e(v')$ changes to $e(v') + 1$ which is equal to previous $e(v)$. Therefore, the load variance does not change. Also, switching from v to a parent $v'' \in P_2$ results in a larger variance of load on v and v''. Thus, every other load factor is equal to or greater than the load factor at the termination time. This means that all children are distributed among parents uniformly as much as possible at that time. $\square$

4.1.2 Worst Case Analysis

In the following the worst cases parent selection delay and number of message exchanges when running the algorithm are evaluated.

• **The worst case of parent selection delay for a child:** This delay is defined as the number of iterations required for the construction of tree. Game terminates when every potential child joins a parent, or in other words $\forall$ parent $v \in V, c(v) = 0$. Based on Lemma 2, $c(v)$ is non-increasing in number of iterations. Thus, for a child $u \in V$ the worst case occurs if it sends *joinReq* message to a $v \in P_u^p$ in each round and it rejects u. However, based on Lemma 2, number of common children decreases for those rejecting parents, lets say by 1 per round in the worst case. At some round, say r, finally one of u's potential parents will have just u as its only remaining common children, and when u send a *joinReq* to it, it will accept u. Thus, the worst time for u is $k\gamma$ rounds waiting, where k is the number of u's potential parents ($|P_u^p|$) and γ is the minimum number of potential children $\forall v \in P_u^p$. Therefore, the worst case of delay in the whole network is the maximum of $k\gamma$, $\forall v \in V$ or in the other words in the order of $O(k\gamma)$.

• **The worst case for number of message exchanges:** Consider the following notations.
M_v: set of node v's common children,
$|P_v|$: number of potential parents of node v,
n_l^1: number of potential children in level l with exactly one potential parent,
n_l^{1+}: number of potential children in level l with multiple potential parents.

Based on previous explanations, the worst case of delay for a child $u \in C_v$ to join a parent is $\lambda_u = \text{Min}\{c(v)\}$, $\forall v \in P_u$. Also, each child at round r has sent *joinReq* message r times. If child u faces λ_u delay for attaching to a parent, it means that $\lambda_u - 2$ common children had been attached before u. Moreover, it can be inferred that u sent λ_u messages, and another child that joined before it sent 1 to $\lambda_u - 1$ messages. This results in

$$\lambda_u + \lambda_u - 1 + \lambda_u - 2 + \cdots + 2 + 1 = \frac{\lambda_u \times (\lambda_u + 1)}{2} \tag{4.4}$$

messages sent by children. From a parent's point of view, each $p_u \in P_u$ has to send its state unless u attaches to a parent. This totally causes $|P_u| \times \lambda_u$ messages for set P_u.

Note that, $r - 1$ joins of other common children had occurred before the selection of a child u in round r. This guarantees that the total number of messages sent by all children in level l to join parents in level $l - 1$ is always less than $n_l^{1+} + \sum_{i=1}^{n_l^1} \lambda_{u_i}$.

According to previous discussion, if a parent v_{l-1} has C_v common children, the maximum delay caused by its children is $\Delta_v^{l-1} = \text{Max}(\lambda(u_i))$, where $1 \leq u_i \leq |Cv|$. Thus, considering messages sent by their parent in the upper level to them, the number of message exchanges never reaches $n_{l-1} \times \Delta_v^{l-1} + n_l^2 + \sum_{i=1}^{n_l^1} \lambda_{u_i}$. Therefore, the upper-bound for number of message exchanges in a graph $G = (V, E)$ with L levels is:

$$\sum_{l=1}^{L} [n_{l-1} \times \Delta_v^{l-1} + n_l^{1+} + \sum_{i=1}^{n_l^1} \lambda_{u_i}] < n\gamma^2, \tag{4.5}$$

where n is number of nodes in the network, and γ is the maximum degree of nodes or in other words maximum number of potential children for nodes.

4.2 Analytical Evaluation of HBDR Algorithm

For HDBR algorithm, the game starts from leaves towards the root. In an arbitrary tree topology, all leaves are not located in the same level. This difference leads to nodes, located at the same level, begin the game in various times. If a parent has some children with different starting times of the game, it has to wait unless it receives data from all its children;

meanwhile, it should inform its other children of this waiting if they start the game sooner. When a waiting parent receives all the required data, it informs other children that are in waiting status. In general, when a node goes into waiting mode, it replies with a wait message to each received message. Also, during this waiting it saves the information in the received massages for the later usage, i.e. after coming out from waiting status.

Because of different starting and terminating times for different nodes at the same level, computing the exact termination time for each node in every run is not possible and it depends on when its beneath nodes terminate the game. Thus, determining an upper-bound for the game's termination time is important to make sure that the game always finishes before the determined time. This time is equivalent to time complexity of the proposed algorithm.

A *fraction* is shown by F and defined as a bipartite graph $(V_1 \cup V_2, E_F)$ such that V_1 is a set of nodes located at level l, V_2 is the set of nodes located at level $l-1$, $V_1 \bigcup V_2 \subseteq V$, $E_F \subseteq E$ and each $v_1 \in V_1 \cup V_2$ is reachable via a path form any $v_2 \in V_1 \cup V_2$. A path from node v_1 to node v_2 consists of a subsequence of edges, with intermediate nodes and two endpoints v_1 and v_2 in V_1 and V_2, respectively. Since edges are considered as directional, if v_1 is also reachable from v_2, v_2 also is reachable from v_1.

Based on the above definition, for two different fractions $F_1 = (V_{11} \cup V_{21}, E_{F1})$ and $F_2 = (V_{12} \cup V_{22}, E_{F2})$, none of $v_1 \in V_{11} \cup V_{21}$ is reachable from any $v_2 \in V_{12} \cup V_{22}$, and vice versa. Otherwise, if a node can be found in one of them so that it can be reached from another node in the other fraction, all nodes can be reached from every node in both fractions. Because those two nodes are reachable from every node in their corresponding fraction. Thus, in fact those fractions are not separate and form a larger fraction together.

Two separate fractions at the same level do not have any dependency to each other, as no path can be found connecting a node from one of fractions to another node in the other fraction. This independency guarantees that playing game in any of them does not affect on the game played in the other fraction. Therefore, the game at one level can be considered as some independent games per fraction.

On the other hand, termination time in one level depends on the termination time in its beneath layers. Thus, the latest game termination time of fractions in one level can be considered as the game termination time for that level, since game is played concurrently and independently in each fraction.

For determining maximum delay in a fraction, let $u \in V_1$ has P_u^p potential parents. The worst case of delay for this child is when it sends *joinReq* to its potential parents based on the received bids from them, and bid values induct a permutation of candidate parents for sending *jonReq* from u. In this case if $v \in V_2$ rejects u' join request because it has another requesting child that has a worse condition than u, it will receive another *joinReq* from u, $|P_u^p|$ rounds later.

According to HBDR algorithm, if a parent has some *joinReq*s in a round, it accepts at least one of them. On the other hand, if a child has some *acceptMsg* it surely accepts one of them. This means, if a parent v rejects u in a round, it accepts another child, say w. Child w will attach to v or rejects its *acceptMsg* and attach to another parent. In either case it comes out from v's common children set. This procedure continues when v accepts all of its children except u, meaning that $P_v^c = u$. In this case, if u sends a *joinReq* message to v for sure it will be accepted, and it will either join v or other parents, and the game terminates for it.

For child u, the maximum delay is equal to time needed for one of its potential parents to have only u as its potential children. The worst case of this condition occurs when bids from u's parents induce an order on selected parents for sending *joinReq* messages from u, in which u sends *joinReq* to one parent in each round, and it rejects u. Also, whenever number of potential children of one parent reaches to one, it means u is its only potential child. In this case, if the parent's bid is worse than others, u does not select it. In the worst case, this procedure continues until all of u's parents have only u as their potential child. Then, when u sends a *joinReq* to any of them it will be accepted and the game terminates for it.

Thus, the maximum delay for child u is

$$D_u = 1 + \sum_{i=1}^{|P_u^p|} (|P_{v_i}^c| - 1) \text{ where } v_i \in P_u^p \tag{4.6}$$

When the game ends for all children of a parent it can start the game in one layer lower as a child. Therefore, for a parent $v \in V_2$ the maximum delay to finish the game in parent role is

$$D_v = \text{Max}\{D_{u_i}\} \ \forall u_i \in P_v^c, \tag{4.7}$$

Also, termination time of game in level l, independent of other level, is

$$D_l = \text{Max}\{D_{u_i}\} \ \forall u_i \text{ located in level } l \tag{4.8}$$

Since starting time of the game in level l depends on termination time of the game in level $l + 1$. The total time needed for termination of the game is

$$D = LD_l, \ 0 < l \leq L, \tag{4.9}$$

where L is the number of levels in graph G. Thus, the order of algorithm is $O(LD_l)$.

In the following it is shown that the solution reached by the proposed algorithm is a Nash equilibrium. In HBDR algorithm child nodes selfishly choose the best bids sent from parents. Thus no child will change its strategy, since it has already received the maximum possible benefit. Also, parents in cooperative way help other parents to reduce the load on them and increase their lifetimes. Based on their utility function, choosing a parent to help another parent implies that the selected parent has offered the worst bid among its connected parent which indicates the worst condition for it in terms of load, remaining energy, delay and offered bandwidth. If helping parent chooses another parent to help, its benefit and also the benefit of children for the not selected parent will be decreased. Thus, the benefit of any other choice is lower than the current choice, since it will not help to prolong the lifetime of the node which is in the bad condition and the consequent network lifetime is shorter than the network lifetime resulting from the current strategy.

4.3 Simulation Results for UDBC Algorithm

In this section, simulation results for UDBC algorithm are presented. A network with size $500m \times 500m$ is used to compare the performance of UDBC method to other related work in [21, 22] and [40]. The brief description of both methods was provided in chapter 2. For the comparison purpose, the proposed method in [40] and [21, 22] are referred as *centralized* and *decentralized* methods respectively. The sensor node model is based on the Mica2Dot which is commercially produced and its characteristics are available in [55]. Specifically, node communication ranges are considered according to this device. During simulation experiments, one node is considered per square meter. Performance of methods are compared by investigation of load factor at termination time, number of iterations and number of message and data item exchanges required for algorithm termination, where variable factors are number of nodes and communication ranges.

Figure 4.1 shows three randomly produced sample networks used in simulations. Each sample shows different number of nodes and communication ranges. In all experiments, a node with the least Euclidean distance to the top left corner is considered as the sink node, and then by issuing a flood from this node, adjacency and levels of other nodes are determined. At the end of flooding, communication infrastructure, i.e. graph G is formed, as shown in Figure 4.1 by gray colored edges. Three compared methods run on this infrastructure. Both variable factors have direct effect on overlaps among communication range of nodes. The higher overlap among nodes results in more dense network where existence of alternating paths [40] are more probable. Generally, the difference between algorithms arise from this point. In the following, different scenarios are discussed to compare the performance of the methods. To obtain more reliable results, each experiment is repeated 20 times and the average of results are reflected in the corresponding figures.

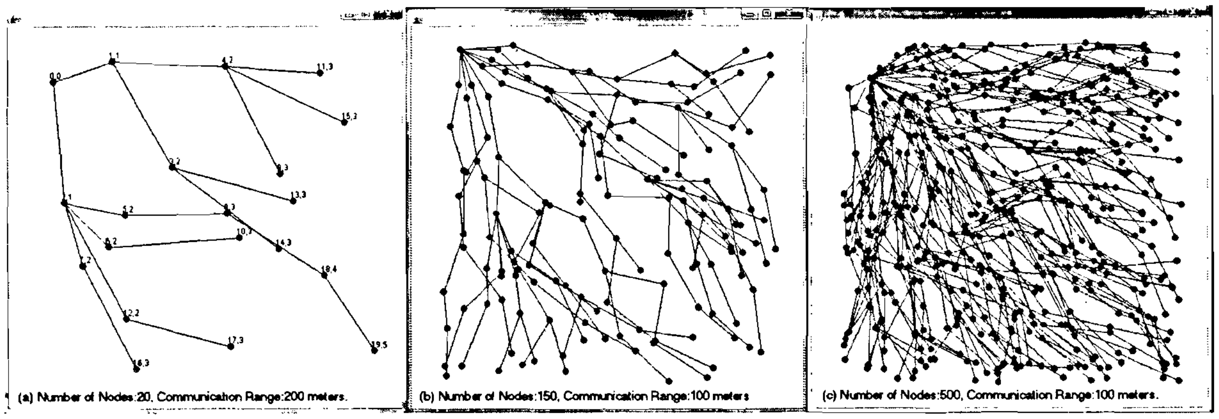
(a) Number of Nodes:20, Communication Range:200 meters.
(b) Number of Nodes:150, Communication Range:100 meters
(c) Number of Nodes:500, Communication Range:100 meters.

Figure 4.1: Three sample networks used in the simulations using UDBC method. Network area is $500m \times 500m$. Nodes are shown by black circles. Dashed gray lines show adjacency of nodes. Black lines show parent-child relationships. (a) Network consisting of 20 nodes with communication range of 200 meters. The first number on top of each node indicates its ID, and the second shows its level in the tree. (b) Network consisting of 150 nodes with communication range of 100 meters. (c) Network consisting of 500 nodes with communication range 100 meters. Note that in figures (b) and (c), ID and level of nodes are excluded to avoid indistinctive figures.

4.3.1 Load Factor Comparison

In the first scenario, the effect of number of nodes on load factor is investigated. Since centralized method always produces the optimum tree, it is considered as the benchmark method for comparison between two other methods. First, the effect of number of nodes on load factor (σ_T) is investigated and Figure 4.2 shows the results. In this scenario node communication ranges are considered fixed at 100 meters, and number of nodes is increased from 50 to 500 nodes in steps of 50. Then, the effect of communication range on load factor is evaluated and Figure 4.3 shows the results. For this scenario, number of nodes is fixed at 300 nodes, and the communication ranges are increased from 25 to 300 meters in steps of 25.

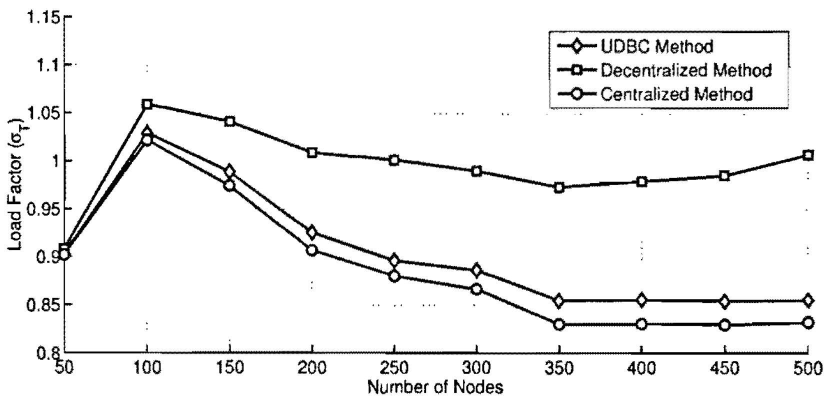

Figure 4.2: Number of Nodes vs. Load Factor.

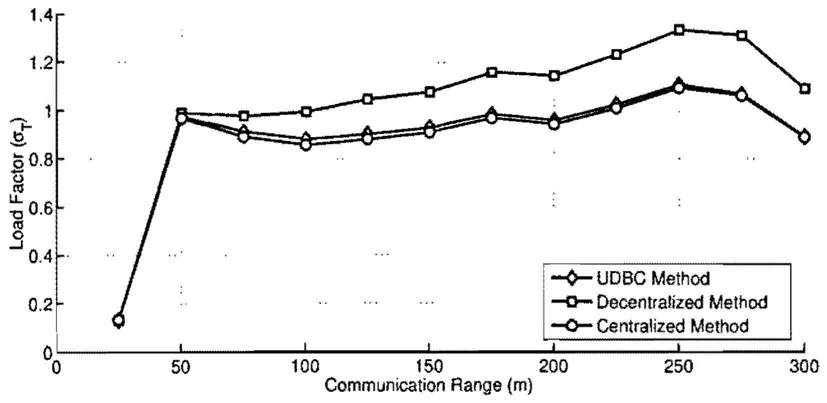

Figure 4.3: Communication Range vs. Load Factor.

When the number of nodes or communication ranges increases, their communication range overlap also increases. The increase in communication range overlap causes more adjacent nodes resulting in more dense network. In a dense network the probability of formation of alternating path increases. If these kinds of paths are not considered during the tree construction process, even nodes are locally balanced but nodes at both ends of such paths are not. Thus, more number of alternating paths means more unbalanced tree. The difference between UDBC method and decentralized method arises from this point. In the decentralized method, a number of children can join a parent in each round of parent selection regardless of the number of children for its connected parents. This can generate alternating paths. However, in UDBC method in each round, a parent can accept only one child more than the minimum number of children among its connected parents. This limitation prevents the mentioned problem in most cases. Thus, as shown in both Figures 4.2 and 4.3, the load factor of UDBC method is much closer to load factor of the centralized method, implying that trees produced by it are more balanced, comparing to decentralized method.

The obtained figures can be interpreted as follows. For the scenario with variable number of nodes, when number of nodes is 50 the network is sparse. Thus, the amount of node adjacencies and consequently the probability of alternating path generation are low. In this case, load factors of the produced tree by three methods are almost the same. However, when the number of nodes is 100, because of the increase in adjacencies, load factors of three methods also increase. Further increment in number of nodes leads to a general decrement in load factors, since the growth of number of nodes (n) in the denominator is more than the corresponding numerator in the load factor formula. However, for the variable communication range scenario, the number of nodes is constant. Thus, the denominator of load factor formula is a constant value. In this case, when the communication increases, increment in amount of adjacencies causes an overall growth of formula's numerator which leads to increment of load factor for the compared methods. Note that in the simulations a portion of load factor is imposed by the network topology and the location and adjacency of nodes, since networks are produced by randomly deployed nodes. Because of this reason the base for comparison is the centralized approach.

4.3.2 Number of Iterations Comparison

Next, the impact of number of nodes on the number of iterations required for the termination of algorithms is investigated. Note that, in the following explanation, the term iteration is used equivalently to the term round. Because in the centralized approach a general view of the network is assumed, and having this view leads to generation of a balanced tree in one round, the centralized approach is excluded for this section. For two other methods, i.e. UDBC and decentralized approach, the number of iterations are important as it causes not only delay in the construction of tree, but also communication overhead. The reason for communication overhead is that each round of parent selection includes a number of message exchanges between potential parents and children. This message exchange results in node energy drain which eventually leads to network connectivity loss. Based on the proposed algorithm in decentralized method, if one parent has at least one departure or attachment of children in one iteration, it will not send any message in the next round. However, our simulation results show that when the number of nodes exceeds 50 nodes, some message are exchanged in each round. Again, simulations are done under two different scenarios: variable number of nodes (Figure 4.4) and variable communication ranges (Figure 4.5). The effect of number of nodes on number of iterations is shown in Figure 4.4. In this scenario, node communication ranges are fixed to be 100 meters, and the number of nodes is increased from 50 to 300 nodes in steps of 50. The impact of variation in communication range on the number of iterations is investigated using Figure 4.5. In this scenario, number of nodes is fixed to be 300, and their communication ranges increase from 25 to 300 meters in steps of 25.

As discussed before, increase in either number of nodes or node communication ranges

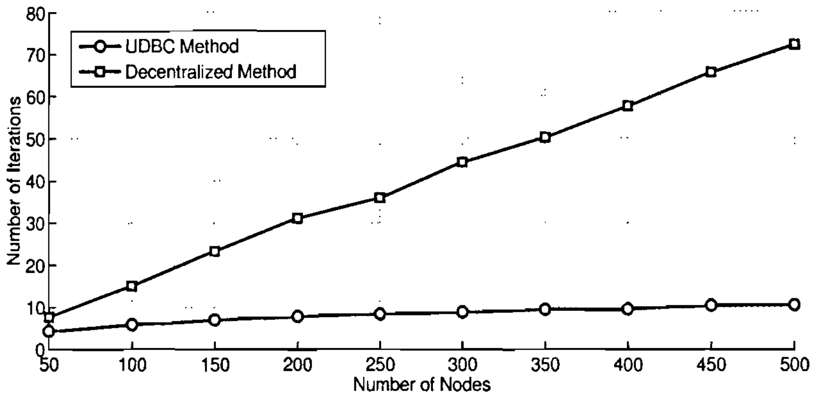

Figure 4.4: Number of Nodes vs. Number of Iterations.

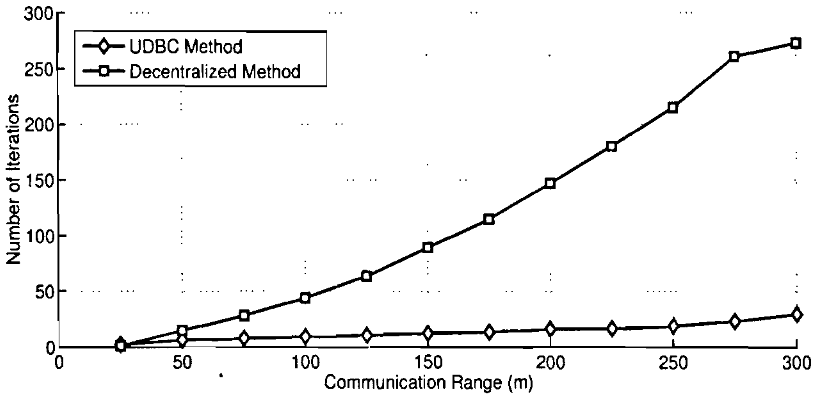

Figure 4.5: Communication Range vs. Number of Iterations.

lead to increase in the amount of their communication ranges overlaps. Referring to Figures 4.4 and 4.5, the number of iterations for UDBC method, especially when the amount of overlaps is high, is much better than the decentralized approach. The reason can be explained by considering the theoretical bounds for number of iterations in both methods. As proved in Section 4.1.1, the theoretical upper bound for number of iterations in UDBC is $O(k\gamma)$, where k is the number of potential parents for children, while for utility-based approach, as mentioned in [21], is $O(M\gamma^2)$, where M is the number of parents in a game. Higher overlaps cause higher amount of adjacency among nodes, which itself results in increase in the number of parents and the maximum degree (number of potential children) among nodes. Increase in the number of parents is not effective on UDBC method. Also, while the order of γ is 1 for UDBC, its order for decentralized approach is 2, making more rapid increase in number of iterations for decentralized method. Tables 4.1 and 4.2 show the values of experimental and theoretical upper bounds for number of iterations in UDBC and decentralized methods. The theoretical bounds for each experiment are determined using the mentioned orders of algorithms and appropriate values of γ, k and M from networks in simulations. Table 4.1 includes values when the number of nodes is variable and the communication range is fixed and corresponds to scenario of Figure 4.4. On the other hand, Table 4.2 indicates values when node communication ranges are different and number of nodes is fixed, which is corresponding to scenario of Figure 4.5.

Table 4.1: Number of Iterations with Variable Number of Nodes.

Number of Nodes	UDBC (Experimental)	UDBC (Theoretical)	Decentralized (Experimental)	Decentralized (Theoretical)
50	4.38	5.37	7.74	294.23
100	5.85	10.43	15.15	2582.01
150	7.97	14.93	23.44	8036.35
200	7.85	19.65	31.20	18701.25
250	8.45	22.87	35.95	32668.10
300	8.85	26.75	44.35	52375.21
350	9.45	30.40	50.11	78470.15
400	9.55	35.35	57.55	122966.68
450	10.45	37.85	65.55	162384.63
500	10.63	43.85	72.05	250935.56

As inferred from both tables, the theoretical bound of decentralized method increases

Table 4.2: Number of Iterations with Variable Communication Range.

Number of Nodes	UDBC (Experimental)	UDBC (Theoretical)	Decentralized (Experimental)	Decentralized (Theoretical)
25	2.45	2.95	1.39	14.77
50	6.18	9.45	14.53	3279.80
75	7.73	17.81	28.25	18465.64
100	9.15	27.55	44.54	55940.93
125	10.34	39.16	63.45	139790.65
150	12.15	53.85	89.15	300121.33
175	13.23	68.75	114.55	618844.54
200	16.05	87.23	147.05	988813.48
225	16.31	110.55	180.48	1774429.00
250	18.85	131.65	214.75	3049310.25
275	23.43	159.74	260.81	4903674.27
300	29.56	176.65	272.85	5841982.65

rapidly by increase in amount of node overlaps. This upper bound also caused higher experimental bound compared to UDBC method.

4.3.3 Number of Message Exchanges Comparison

Comparing number of iterations allows to investigate the amount of time overhead (or delay) imposed by methods. In the previous scenario we claimed that increase in number of iterations causes increase in communication overhead except the time overhead. This communication overhead is a result of message exchanges occurred in each iteration of parent selection. Also, it was discussed for both UDBC and decentralized methods, that each iteration is followed by a number of message exchanges. Thus, in this section, the number of messages required for tree construction in both methods are compared. This way, a more precise factor can be provided for communication overhead comparison.

The difference of messages in compared methods are *StateMsg* and *JoinReq* messages. The counterpart of *StateMsg* in UDBC method is *Bid* message in decentralized method [21] which is the guaranteed bandwidth from a parent for its children. Each *Bid* message contains one unit of data, while each *JoinReq* message contains two units of data. Moreover, *JoinReq* message in UDBC method has a corresponding message in decentralized method which is for showing the *will* of a child to join some parents. While *will* messages are just for informing of parents, each *joinReq* message carries two units of data to help more accurate decisions.

Thus, it can be argued that UDBC method reaches more balanced tree at the cost of adding more overhead to messages. In fact, this experiment examines the additional data units in messages. In the experiments, each message which does not carry any number is considered as one unit of data, and messages carrying some data units has the cost equal to the number of data units they hold.

For the decentralized method, number of sent data units and sent messages are the same, since each has the cost of one data unit. For UDBC method *StateMsg* and *JoinReq* messages have the extra cost of 3 data units. Thus the number of sent messages and sent data units are not equal. Therefore, both sent message and sent data units are provided in the simulation results for better comparison. Figures 4.6 and 4.7 show the simulation results under two different scenarios. The scenario of Figure 4.6 investigates the impact of variation in number of nodes on the number message exchanges, while Figure 4.7 indicates the effect of variation in transmission ranges.

As inferred from figures, number of sent message required for construction of load balanced tree in UDBC method is much less than the decentralized approach. The main reason for this difference is the difference in the number of iterations investigated before. Moreover, by comparing the number of sent data items, although the decentralized approach has a better performance first, when the communication overlaps increase the performance of UDBC becomes better. Also, additional information added by UDBC to messages is not too much and still the information can be sent through one longer message, instead of multiple messages. According to [9,10,14], transmission of one longer message for an amount of data is less energy consuming than sending some shorter messages for them. Thus, it can still be claimed that whenever the number of sent data units by two methods are close to each other, UDBC has a better performance, since it sends them via longer messages, but in a less number.

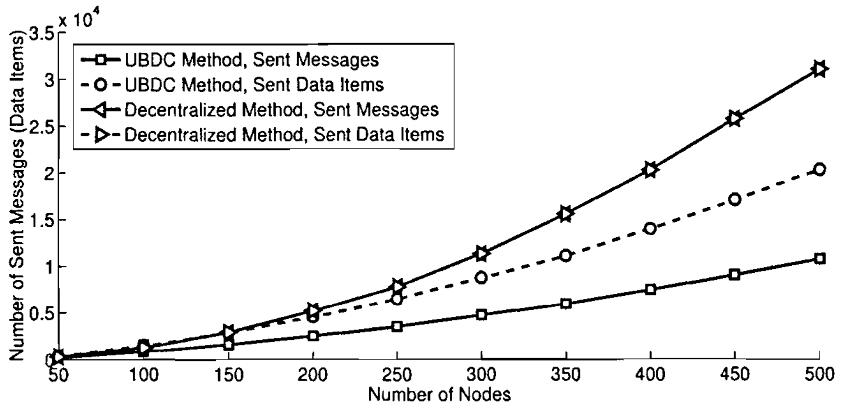

Figure 4.6: Number of Nodes vs. Number of Messages (Data items).

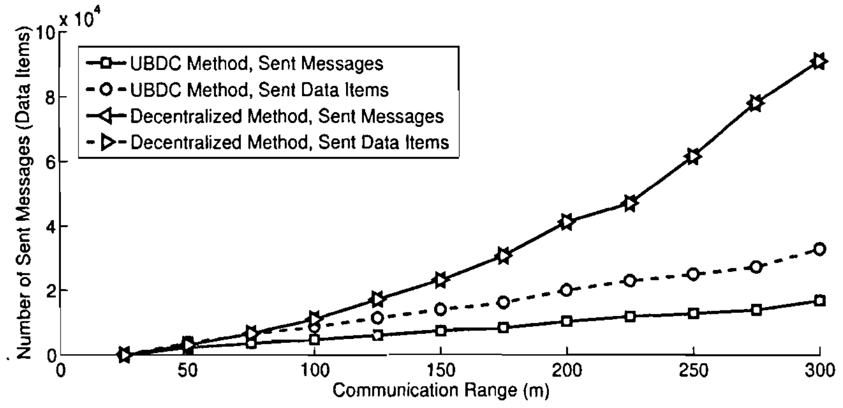

Figure 4.7: Communication Range vs. Number of Messages (Data Items).

4.4 Simulation Results for HBDR Algorithm

In this section the performance of HBDR is investigated and discussed. A network with size 500 m × 500 m is considered. Again, the node model is Mica2Dot. Comparing to UDBC algorithm, more parameters are involved in the HBDR algorithm. Changes in each of these parameters can effect on the algorithm performance. Thus, in the simulation different scenarios are considered through which one or some parameters are variable and the rest are fixed. Then, the impact of changes in the variable parameters on the performance of algorithms are investigated.

The performance of the algorithm is compared to the proposed algorithm in the [21] which explained in chapter 2 and we refer to it as cumulative algorithm, since it considers cumulative cost based the residual energy of nodes on the path from each node to the base station.

With the similar method for simulation of UDBC algorithm, graph G is generated first, and on top of this graph the compared algorithms are run. To gain more precise results, each experiment is run 20 times and the average of the results are reflected in the diagram of scenarios.

Table 4.3 includes default values of various parameters in simulations. Scenario-specific changes in this default values is mentioned in the corresponding section of each scenario. The following are explanations of different scenarios.

Table 4.3: Default values of parameters in the simulation of HBDR algorithm

Parameter	Value
Network Dimension	500(m)×500(m)
Number of Nodes	150
Node Communication Range	150(m)
Aggregation Function (λ_1,λ_2)	(1,0)
(Time Coefficient, Frequency Coefficient)	(1,1)
E_{elec}	50 nj
E_{Amp}	100 pj
Initial Energy	10 joule
Data Amount	128 bits
Data Rate	3 kbps
Bandwidth	30 KBuad (BPSK modulation is used)

4.4.1 Network Lifetime

Network lifetime is defined as the time from the start of running the network until the first node failure. Network lifetime is an important factor, since as mentioned earlier, sensor networks have ad-hoc topology and consequently network connectivity depends on aliveness and data forwarding of nodes. In this scenario the effect of changes in number of nodes and node communication ranges on network lifetime is investigated. Generally, increase in number of nodes or node communication range leads to higher adjacency among them which gives more chance of load balancing. In one scenario, number of nodes increase from 50 nodes to 300 nodes in steps of 50 and the communication ranges is fixed on 150 meters. In the other, communication range increases from 50 m to 350 m in the steps of 50 m and the number of nodes is fixed on 150 nodes. Figures 4.8 and 4.9 depict the results of these two scenarios respectively.

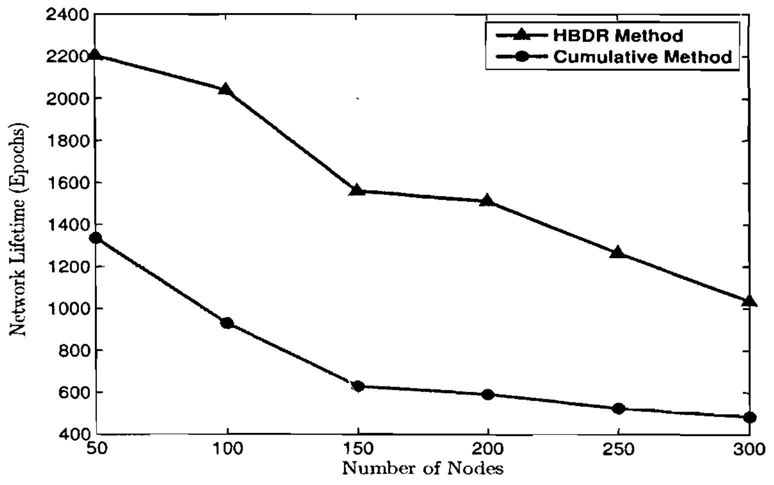

Figure 4.8: Number of Nodes vs. Network Lifetime.

As shown in the figures, HBDR method has a better performance than cumulative method in both cases. The reason is that cumulative method run proactive distance vector algorithm updating and regenerating routing tree periodically. Since it is implemented in a distributed way, and the flooding mechanism is used and during this flooding nodes choose a parent with minimum cost to join. Thus, it is probable that a bunch of potential children joins a

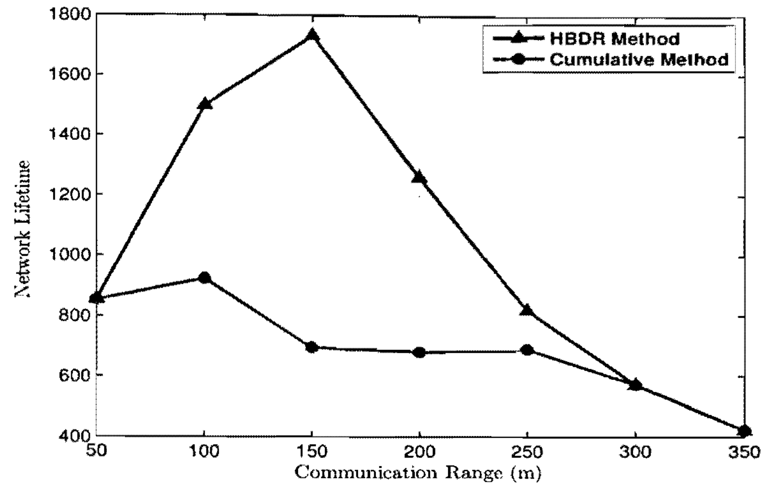

Figure 4.9: Node Communication Range vs. Network Lifetime.

specific potential parent because it has the best cost among others. Although the algorithms is able to update, the routing tree joining of number of nodes to one parent because of its appropriate offer results in faster energy drain of that node. This fast energy drain forces a sooner regenerating process of routing tree. However, for HBDR the balanced assignment of potential children based on their load and also the cost of the parent is considered.

For the scenario in which communication range is considered variable when the communication range is short because of less amount of adjacency among nodes, the strategy space of nodes for parent selection is limited. Thus, two methods have similar performance. While the communication range increases, the chance of selection among different parents also increases. This increases the chance of more load balancing specifical for HDBR algorithm, resulting in longer network lifetime. When the communication range becomes larger than 200 meters, since the network dimension is 500 $m \times 500$ m, more nodes can directly connect to base station, and this decreases the impact of load balancing, resulting in the same performance for both methods.

4.4.2 Average Node Lifetime

In this scenario, the effect of variation of number of nodes and node communication range on average node life time is investigated. Average node lifetime is determined by summation of lifetime of all nodes in the network, except the base station, divided by the their total number. Again, number of nodes increases from 50 nodes to 300 nodes in steps of 50 and the communication ranges is fixed on 150 meters. In the other, communication range increases from 50 m to 350 m in the steps of 50 m and the number of nodes is fixed on 150 nodes. Figures 4.10 and 4.11 show the simulation results for two cases.

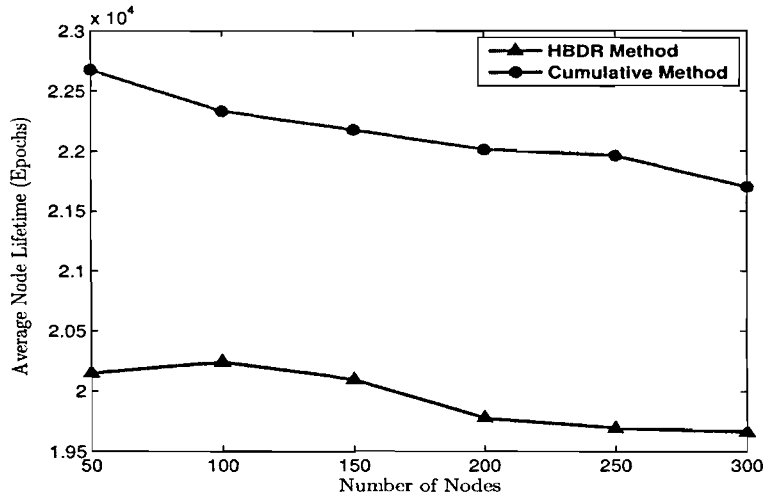

Figure 4.10: Number of Nodes vs. Average Node Lifetime.

As it can be inferred from the figures the average of node lifetimes for HBDR method is less than the cumulative method. The reason is, because in cumulative method the number of nodes which are under-used and have a longer lifetime is more than the HBDR method, and this higher number causes a better total average for nodes' life time. On the other hand, some nodes are over-used, which causes faster network disconnection and shorter network lifetime. In the next scenario, the variance of node life time is investigated to check the diversity of node lifetime from the average values.

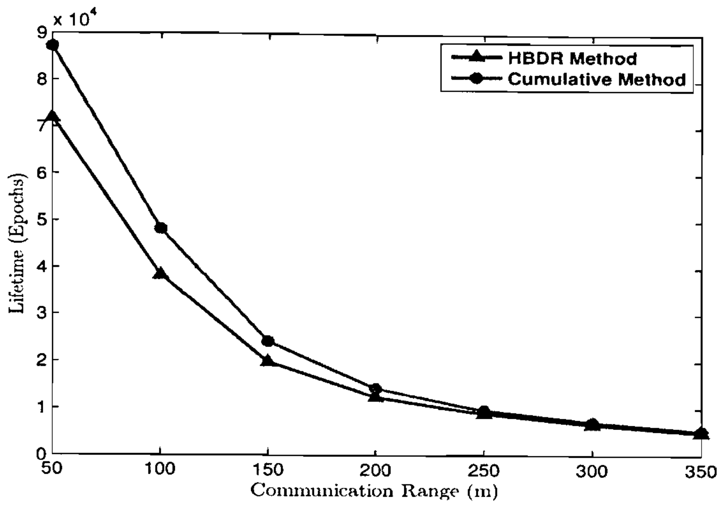

Figure 4.11: Node Communication Range vs. Average Node Lifetime.

4.4.3 Standard Deviation of Node Lifetime

As mentioned before, standard deviation from the average node lifetime is an indication of load balance efficiency of the methods. The used formula for computation of standard deviation is shown in (4.10).

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(l_i - \bar{l})^2}, \tag{4.10}$$

where n is the number of nodes, l_i is the node i's lifetime, and $\bar{l}$ is the average node life time, mentioned in section 4.4.2. Higher variation to the average implies shorter lifetime for some nodes leading to shorter network lifetime. Figures 4.12 and 4.13 shows the standard deviation of network lifetimes, when number of nodes and node communication range are variable respectively. All the parameters for simulation are the same as the previous scenario.

As shown in the figures, HBDR method has less standard deviation compared to cumulative method. This means that nodes that run HDBR algorithm have closer lifetime to each other, resulting from more balanced assignment of loads to nodes. For the case in which communication range is variable, when communication range relative to network dimension increases, more nodes are able to attach base station directly. As a result, both methods

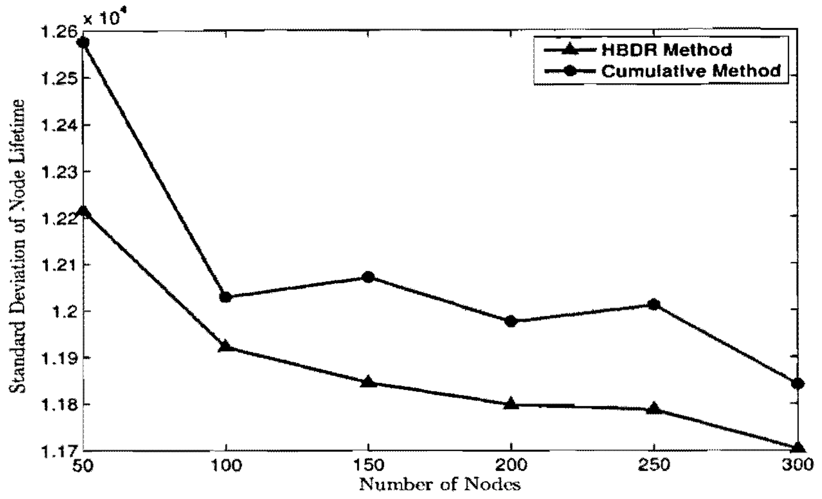

Figure 4.12: Number of Nodes vs. Standard Deviation Node Lifetime.

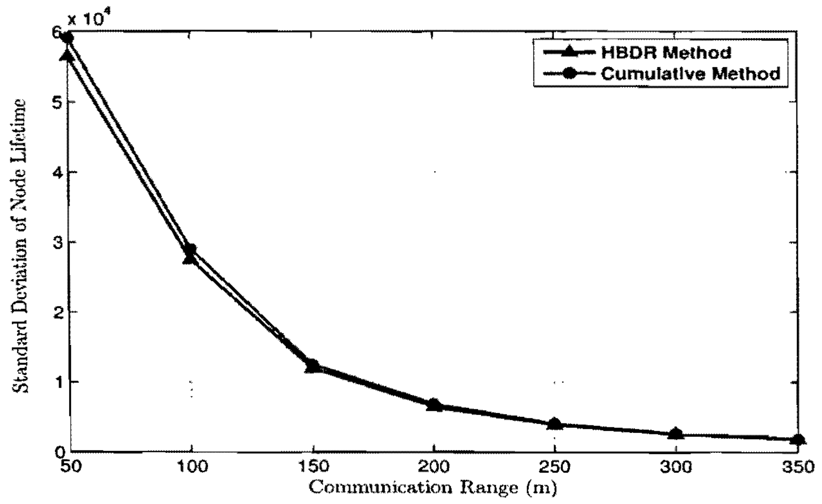

Figure 4.13: Node Communication Range vs. Standard Deviation Node Lifetime.

have close node lifetime leading to similar standard deviation of node lifetime.

4.4.4 Cumulative Distribution Function of Node Lifetime

Cumulative distribution function (CDF) describes the probability that a real-valued random variable X with a given probability distribution will be found at a value less than x. In our scenario random variable is node lifetime, and the Figure shows the probability of finding the node lifetime less than a specific number. Thus, a higher value of probability indicates more number of nodes have lifetime less than that value, implying less probability value is more desirable. For this simulation 500 nodes with communication range of 150 m are considered. The rest of parameters are as the same as default parameters.

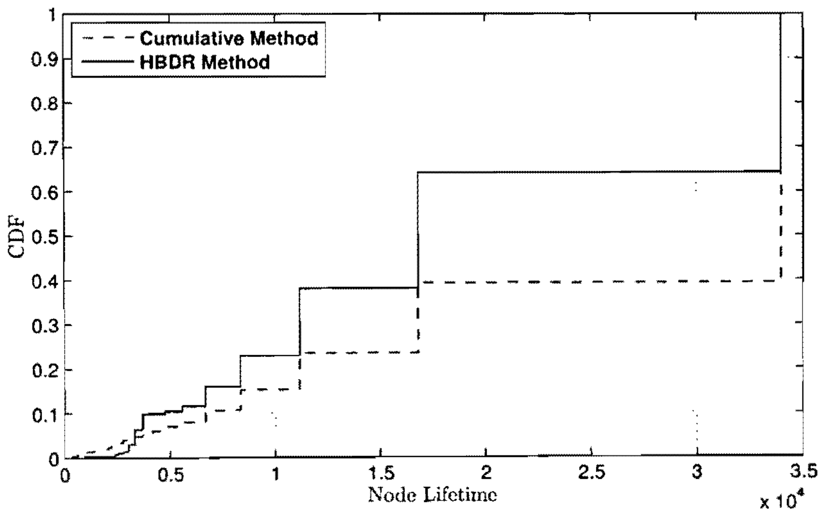

Figure 4.14: Cumulative Distribution Function of Node Lifetime.

As it can be seen in the figure, the probability of having nodes with longer lifetime than 3000 epoch for cumulative method is more than HBDR method. However, by checking two CDF graphs, the probability of finding nodes with lifetime less than 3000 epoch for cumulative algorithm is more than HBDR method. This implies that the probability of loosing the connectivity using cumulative method is higher than HBDR method. The fact that can negate the benefit of having longer lifetime for other nodes.

4.4.5 Number of Message Exchanges

As the communication consumes a considerable portion of nodes' energy, reducing the amount of communications between nodes is important. Each algorithm has communication overhead to run. This communication overhead is a result of transmission and receiving messages during the execution of the algorithms. Thus, less message exchange required by algorithms for construction of routing tree is more desirable. In this scenario, the number of exchange is compared between HDBR and cumulative algorithm. Variable parameters are number of nodes which is increased from 50 to 300 in steps of 50, and node communication range which is increased from 50 m to 350 m in steps of 50 m. For the reflection of results, total number of messages which exchanged in whole network among nodes using both algorithms is investigated and the results are shown in Figures 4.15 and 4.16.

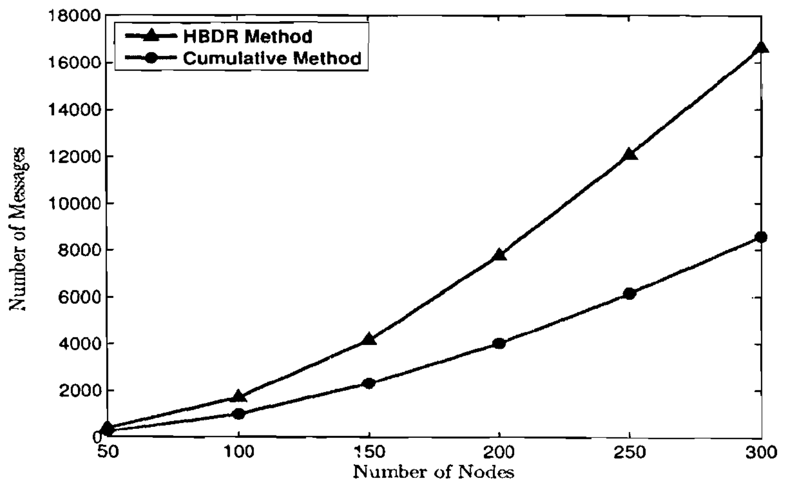

Figure 4.15: Number of Nodes vs. Number of Message Exchange.

As shown in the figure, HDBR suffers from more total number of message exchange for tree construction. Because in HDBR different types messages are used for construction of more balanced tree, such as *BidPair*, *joinReq*, *Accept*, and *Reject* messages. On the other hand, cumulative method uses the flooding mechanism which only requires messages that have the same usages as *BidPair* and *Accept* messages in HDBR. Therefore, total number of messages which are needed to be exchanged among nodes for tree construction in cumulative

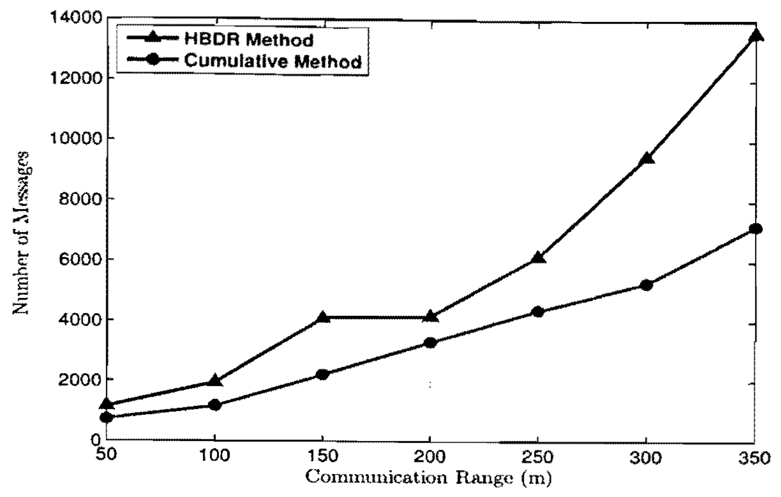

Figure 4.16: Node Communication Range vs. Number of Message Exchange.

method is less than the HBDR method.

4.4.6 Updating Factor

Although the total number of message exchanges is important, it is not a precise factor for communication overhead of algorithms. For more precise investigation, the frequency of exchanges should also be investigated. The reason is that it is probable a large number of message exchanges occurs but not frequently, and on the other hand less number of message exchanges occurs more frequently. The latter case may impose larger communication overhead, specifically when the difference of message exchanges is not high. The mentioned point is a good reason to define *updating factor* as below:

$$updatingfactor = \frac{\text{total number of message exchanges}}{\text{network lifetime}} \tag{4.11}$$

Network lifetime is a good indication of how fast it is needed that the tree is regenerated. Thus, updating factor is an indication of what amount of message exchanges is required in each epoch for keeping the tree up-to-date and network connected. The less the updating factor is, the less communication overhead per epoch is imposed. Again, simulation results are reflected under the condition of different number of nodes, and different node communication range respectively.

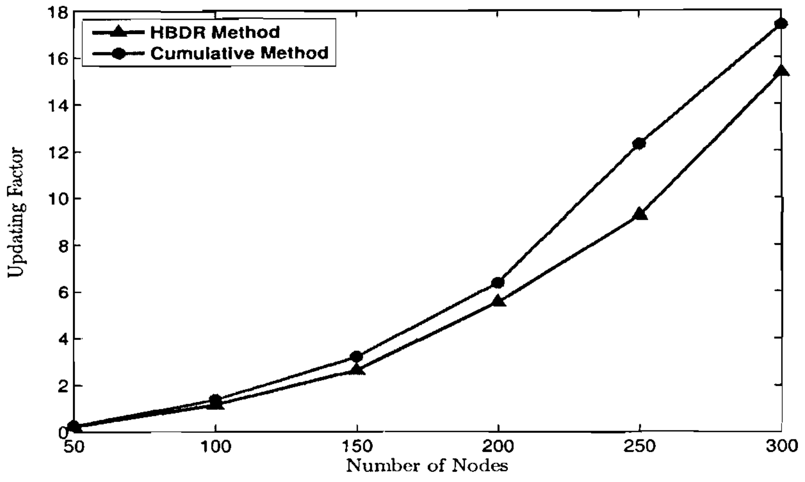

Figure 4.17: Number of Nodes vs. Updating Factor.

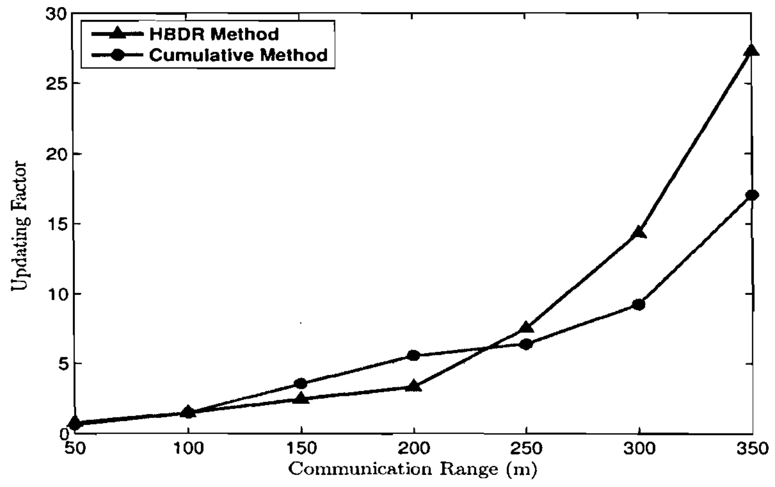

Figure 4.18: Node Communication Range vs. Updating Factor.

As it can be seen in figures, HDBR method mostly outperforms cumulative method. This is because, although cumulative method imposes less amount of message exchanges, it is not a more balanced algorithm than HDBR algorithm. Thus, over-used nodes have faster energy depletion and updating of the tree have to occur more often than HBDR method, resulting in a higher updating factor than HBDR method. However, as indicated in Figure , when the communication range is variable and it becomes close to the network dimension, more nodes can directly connect to base station. This direct connection causes less load on nodes in both approaches. Thus, both approaches updates trees less frequently. However, because the number of message exchanges needed in each construction of tree using cumulative method is less than the number of message exchanges required in each construction of tree using HBDR method, it outperforms HBDR method in this case.

4.4.7 Time and Frequency Factors

HDBR algorithm is capable to provide support based on application needs in terms of delay and reliability. As mentioned before, for some application which have realtime needs the delay in data delivery is critical, while for some others which can tolerate more delay, reliability of data delivery is more important. To support both cases, time and frequency coefficients are considered in the children utility function formula. By adjusting these coefficients, bids can be weighted based on time or bandwidth supplies. *time factor* and *frequency factor* are defined based on (4.12) and (4.13) respectively.

$$\text{Time Factor} = \frac{R(v)}{\lambda_1(L(v) + \sum_{i=0}^{|\mathbf{C}_v|} L(u_i)) + \lambda_2} \tag{4.12}$$

$$\text{Frequency Factor} = \frac{B_R(v)}{\sum_{i=0}^{|\mathbf{C}_v|} L(u_i)} \tag{4.13}$$

Time factor is an indication of the time that node should wait for forwarding its data by its parent toward the base station. Frequency factor indicates how much bandwidth needs to be assigned to the node from its parent to send transmit its data to that parent. In this scenario, the effect of changes in the time and frequency coefficients on the mentioned factors are investigated. In both cases, time coefficient is increased from 0 to 1 in steps of 0.1, and frequency factor is decreased from 1 to 0 in steps of 0.1. Node data transmission rate is considered 10, 15, 20, 25, 30 KBauad randomly, with BPSK modulation. Bandwidths which parents can assigns to their children for data transmission are randomly selected from {1, 2 and 3} Kbps. The rest of parameters are set as default values. Figures 4.19 and 4.20 shows the results.

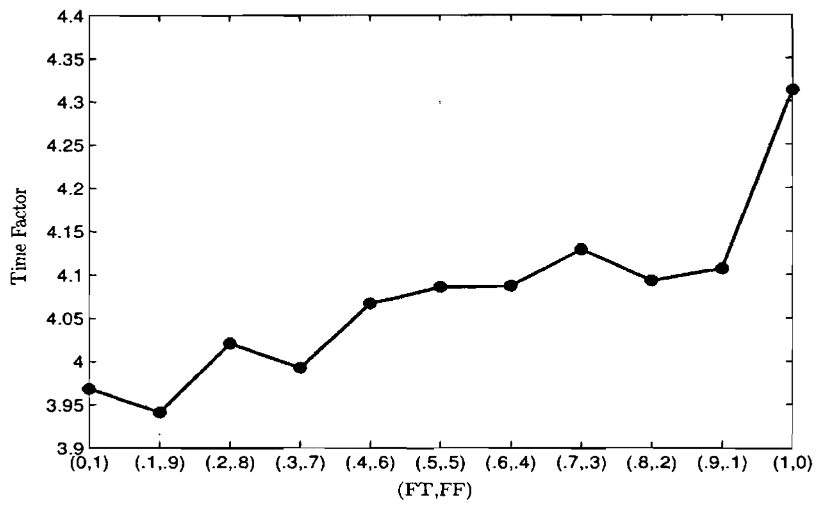

Figure 4.19: Time and Frequency coefficient vs. Time Factor.

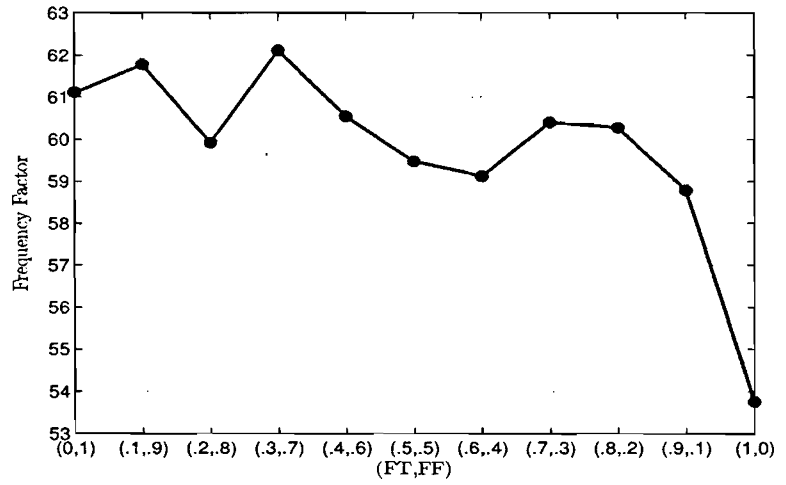

Figure 4.20: Time and Frequency coefficient vs. Frequency Factor.

As shown in the figures, when time coefficient increases the average of time factor for nodes in the network increase. On the other hand, when the frequency coefficient increases the effect of frequency factor increases and the effect of time factor decreases.

4.4.8 Different Aggregation Functions

In the last scenario, network lifetime under the condition of running various aggregation functions is examined. Different functions can be modeled by 3.9 and two coefficients λ_1 and λ_2. Increasing in the value of each of this coefficients implies higher data overhead imposed by the aggregation function. In this scenario λ_2 has values 0, 0.5, 1, 2, 3 under the condition of λ_1 having the values of 0.5, 1 and 2. The values of other parameters are kept as default. Figure 4.21 indicates the results.

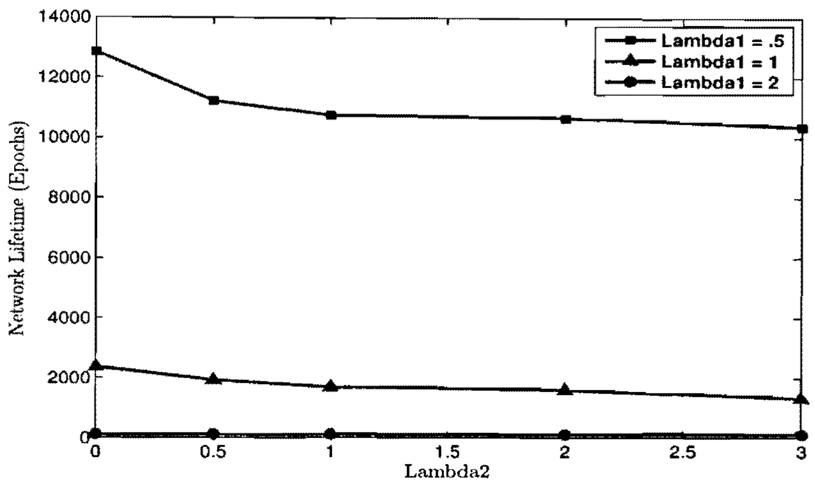

Figure 4.21: Network Lifetime vs. Different Aggregation Functions.

As it is inferred from the figure, where the value of λ coefficient increases, network life decrease. Because increases of λ causes higher amount of traffic going out from each node which leads to higher communication overhead and energy consumption. Coefficient λ_1 has more effect, since it has aggregating effect of the total data gathered from all nodes beneath a specific node. Hence, the smaller coefficient value means the better compression of data resulting much less traversing data in the network.

4.5 Summary

Based on the evaluations provided in this section, it is inferred that UDBC method has outperforms the related work. It improves the time overhead which is incurred by the algorithm and the amount messages needed to be exchanged among nodes for the construction of tree. UDBC is simple algorithm which is implementable in a distributed way. Compared to decentralized method, UDBC is able to generate more balanced routing tree, in terms of even assigning of children to parents. This more accurate result results in longer network connectivity and lifetime.

Comparison of HBDR to cumulative methods leads to a longer network lifetime. This lifetime is the result of generating more load balanced tree compared to cumulative method. This balancing results in less frequent update and re-construction of tree. HBDR needs more time for construction of tree, and impose more message exchange in each construction. However, since updates occurs less frequently when using the HBDR algorithm, the overall number of messages per epoch required to keep the tree up-to-date is less than the cumulative method. Also, defining of λ_1 and λ_2 factor was shown that can properly model different aggregation functions, and the amount of produced data by using them. Time and frequency coefficients, as simulation results shows, can be easily adjusted to meet the needs of different applications regarding to sensitivity to the delay or quality of data delivery.

Chapter 5

Conclusion and Future Research Directions

5.1 Summary

In this work, two algorithms based on game theoretic approach for load balanced data routing in wireless sensor networks are proposed and analyzed. Load is defined as the amount of energy required to receive, process and transmit information by nodes. Load balancing is critical for sensor networks as the only energy resource for sensor nodes is their limited battery supplies resulting in computation and communication limitations. The proposed algorithms specifically aim to solve the problem of constructing a load balanced tree for data routing.

The game theoretic approach implies designing of games in that, nodes as players have two roles as child and parent. In a child role, nodes are selfish players trying to gain more possible available bandwidth, while in parent role they act as cooperative nodes trying to decrease the load on their connected parents which are two-hop away nodes in the same level. Each parent is connected to its connected parents via nodes in one lower level.

The first approach is a Utility-driven Distributed method for Balanced data Communication in sensor networks, called UDBC. It is specifically designed for homogenous environments in terms of out-going traffic amount from nodes. The proposed algorithm considers the limitations of sensor networks and offers a distributed method which requires less communication and still has a simple implementation and computation cost. UDBC method can produce an optimally load balanced tree, where in-network perfect aggregation is applied. Analytical evaluations and simulation results show better performance of the algorithm in terms of producing more balanced trees and, time and communication overheads compared to other related work.

In the above work nodes are considered homogenous which produce similar amount of data. However, considering load heterogeneity of nodes opens a challenging area. Heterogeneous nodes in terms of their sending data is a solution for applications for which in-network aggradation is not possible. As this case is more general than the homogenous case, solving this problem will lead to a solution for all kinds of queries. For this purpose, another approach is proposed for Heterogenous Balanced Data Routing in wireless sensor networks, called HBDR. It not only considers different amount of data from each node, but also considers bandwidth and delay as the criteria to achieve a data routing scheme with better quality. For this purpose, besides the remaining energy of nodes, their bandwidth and data transmission rate are also imported as metrics in the nodes utility function. HDBR models different kind of aggregation and can adapt itself based on the application needs .

Although HDBR algorithm can support all kind of queries and homogenous and heterogenous environments, it has more complicated implementation and requires more computation and communicating resources rather than UDBC algorithm. As in most of applications, based on user queries use of only in-network perfect aggregation is sufficient, developing a simple, fast and more accurate algorithm to support homogenous environment is beneficial.

5.2 Future Research Directions

Considering a few number of mobile nodes in the network will add some challenging dynamics to the problem, where some nodes move from one communication area to another. This will cause different load for both nodes which are located in the communication range of the mobile node and mobile node itself, when it goes through communication range of different nodes.

Although the proposed algorithms are discussed for construction of routing tree, they can easily be adapted to support updating a portion of trees, i.e. subtrees during runtime. Because in a long duration of query running, it is probable from a certain time, initial conditions based on which the construction is conducted change and consequently the current tree topology becomes non-optimum. However, reconstruction of the whole tree may not be beneficial, and fixing the problem in a subtree of the whole tree solves the problem.

Also, nodes can have different communication ranges as another heterogeneity criterion. This difference may rise from different levels of residual energy, or deploying nodes with different cases. In both cases, importing different range of communication in the formulas looks practical. In the HDBR algorithm, considering the communication range as a metric in the energy consumption formula for data transmission can support this issue, when instead of equal values for transmission ranges of nodes, different values for every or some of them. For this purpose, a function can be defined which determines the communication range of a node, based on its remaining energy or based on a pre-configured mapping function.

Bibliography

[1] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, "Instrumenting the world with wireless sensor networks," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 2033 2036, May 2001.

[2] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *the Second International Conference on Embedded Networked Sensor Systems (Sensys)*, (Baltimore, MD), 2004.

[3] J. Yick, B. Mukherjee, and D. Ghosal, "Analysis of a prediction-based mobility adaptive tracking algorithm," in *the IEEE Second International Conference on Broadband Networks (BROADNETS)*, (Boston), 2005.

[4] M. Castillo-Effen, D. Quintela, R. Jordan, W. Westhoff, and W. Moreno, "Wireless sensor networks for flash-flood alerting," in *the Fifth IEEE International Caracas Conference on Devices, Circuits, and Systems*, (Dominican Republic), 2004.

[5] T. Gao, D. Greenspan, M. Welsh, R. Juang, and A. Alm, "Vital signs monitoring and patient tracking over a wireless network," in *the 27th IEEE EMBS Annual International Conference*, 2005.

[6] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, "Sensor networks for emergency response: challenges and opportunities," *Pervasive Computing for First Response (Special Issue), IEEE Pervasive Computing*, October/December 2004.

[7] G. Wener-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M.Walsh, "Deploying a wireless sensor network on an active volcano," *Data-Driven Applications in Sensor Networks (Special Issue)*, March/April 2006.

[8] V. Potdar, A. Sharif, and E. Chang, "Wireless sensor networks: a survey," in *WAINA '09*, (Australia), pp. 636 641, May 2009.

[9] J. Gehrke and S. Madden, "Query processing in sensor networks," *Pervasive Computing*, vol. 3, January/March 2004.

[10] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "Tinydb: An acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, vol. 30, pp. 122–173, March 2005.

[11] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman, "Multi-query optimization for sensor networks," in *the First IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2005)*, (Marina del Rey, CA, USA), June 2005.

[12] W. Yu, T. N. Le, J. Lee, and D. Xuan, "Effective query aggregation for data services in sensor networks," *Computer Communications*, vol. 29, pp. 3733 3744, November 2006.

[13] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, March 2002.

[14] Y. Yao and J. Gehrke, "Query processing in sensor networks," in *the First Biennial Conference on Innovative Data Systems Research (CIDR)*, (Asilomar, California), January 2003.

[15] L. Zhao, G. Liu, J. Chen, and Z. Zhang, "Flooding and directed diffusion routing algorithm in wireless sensor networks," in *the 5th International Conference on Hybrid Intelligent Systems*, vol. 2, pp. 235 239, August 2009.

[16] H. Dai and R. Han, "A node-centric load balancing algorithm for wireless sensor networks," in *IEEE Global Telecommunications Conference*, vol. 1, pp. 548 552, December 2003.

[17] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *Wireless Communications*, vol. 14, pp. 70 87, March 2007.

[18] J. Al-Karaki and A. Kamal, "Routing techniques in wireless sensor networks: a survey," *Wireless Communications*, vol. 11, pp. 6-28, 2004.

[19] F. Bouabdallah, N. Bouabdallah, and R. Boutaba, "On balancing energy consumption in wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 58, 2009.

[20] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *5th symposium on Operating systems design and implementation*, (Boston, Massachusetts, USA), December 2002.

[21] N. Sadagopan, M. Singh, and B. Krishnamachari, "Decentralized utility-based design of sensor networks," *Mobile Networks and Applications*, vol. 11, pp. 341 350, 2006.

[22] N. Sadagopan and B. Krishnamachari, "Decentralized utility-based design of sensor networks," in *2nd Workshop on Modeling and Optimization in Mobile Ad Hoc and Wireless Networks (WiOpt'04)*, (University of Cambridge, UK), March 2004.

[23] B. A. and A. A., "Utility driven balanced communication (udbc) algorithm for data routing in wireless sensor networks," in *25th IEEE Biennial Symposium on Communications*, (Queen's University, Canada), May 2010.

[24] D. Fudenberg and J. Tirole, *Game Theory.* MIT Press, 1991.

[25] D. Niyato and E. Hossain, "Radio resource management games in wireless networks: an approach to bandwidth allocation and admission control for polling service in ieee 802.16," *Wireless Communications*, vol. 14, pp. 27 35, 2007.

[26] A. B., MacKenzie, and L. A. DaSilva, "Game theory for wireless communications," 2006.

[27] I. Uliman, R. C. Pomalaza, I. Oppernann, and J. Lehtomaki, "Radio resource allocation in heterogeneous wireless networks using cooperative games," in *Nordic Radio Symposium 2004/Finnish Wireless Communications Workshop*, August 2004.

[28] D. E. Charilas and A. D. Panagopoulos, "A survey on game theory applications in wireless networks," *Computer Networks*, 2010.

[29] E. Jorswieck, E. Larsson, M. Luise, and H. Poor, "Game theory in signal processing and communications," *Signal Processing Magazine*, vol. 26.

[30] R. Rosemark, W. Lee, and B. Urgaonkar, "Optimizing energy-efficient query processing in wireless sensor networks," in *International Conference on Mobile Data Management*, (Mannheim, Germany), May 2007.

[31] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor nnetworks," *SIGMOD Record*, vol. 31.

[32] A. Munteanu, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "Multiple query routing trees in sensor networks," in *the IASTED International Conference on Databases and Applications (DBA05)*, (Innsbruck, Austria), pp. 145 150, February 2005.

[33] R. Muller and G. Alonso, "Efficient sharing of sensor networks," in *IEEE International Conference on Mobile Adhoc and Sensor Systems*, (Vancouver, BC, Canada), October 2006.

[34] R. Muller and G. Alonso, "Shared queries in sensor networks for multiuser support," *Technical Report 508*, February 2006.

[35] S. Xiang, H. B. Lim, and K. L. Tan, "Multiple query optimization for wireless sensor networks," in *23rd IEEE International Conference on Data Engineering (IEEE07)*, (Istanbul, Turkey), April 2007.

[36] S. Xiang, H. B. Lim, K. L. Tan, and Y. Zhou, "Two-tier multiple query optimization for sensor networks," in *27th International Conference on Distributed Computing Systems (ICDCS'07)*, (Toronto, ON, Canada), June 2007.

[37] N. Trigoni, A. Guitton, and A. Skordylis, "Routing and processing multiple aggregate queries in sensor networks," in *the 4th international conference on Embedded networked sensor systems*, (Colorado, USA), pp. 391–392, 2006.

[38] K. Lee, W. Lee, B. Zheng, and J. Winter, "Processing multiple aggregation queries in geo-sensor networks," in *the 11th International Conference on Database Systems for Advanced Applications (DASFAA06)*, (Singapore), April 2006.

[39] X. Zhang, X. Yu, and X. Chen, "Inter-query data aggregation in wireless sensor networks," in *International Conference on Wireless Communications, Networking and Mobile Computing (WCNM 2005)*, (Wuhan, China), September 2005.

[40] N. J. Harvey, R. E. Ladner, L. Lovasz, and T. Tamir, "Semi-matchings for bipartite graphs and load balancing," in *Workshop on Algorithms and Data Structures (WADS 2003)*, (Canada), July 2003.

[41] H. Huang, Y. Xu, Y. e Sun, and L. Huang, "Cluster-based load balancing multi-path routing protocol in wireless sensor networks," in *7th World Congress on Intelligent Control and Automation*, (China), June 2008.

[42] D. Mandala, F. Dai, X. Du, and C. You, "Load balance and energy efficient data gathering in wireless sensor networks," *Wireless Communications and Mobile Computing*, vol. 8, pp. 645–659, June 2008.

[43] M. Fyffe, M. Sun, and X. Ma, "Traffic-adapted load balancing in sensor networks employing geographic routing," in *8th IEEE Wireless Communications and Networking Conference*, pp. 4392–4397, March 2007.

[44] M. Ma and Y. Yang, "Clustering and load balancing in hybrid sensor networks with mobile cluster heads," in *the 3rd international conference on Quality of service in heterogeneous wired/wireless networks*, (Waterloo, Ontario, Canada), 2006.

[45] Y. Chu, C. Tseng, C. Hung, K. Liao, C. Ouyang, C. Yen, J. Jiang, Y. Wang, C. Tseng, and E. Yang, "Application of load-balanced tree routing algorithm with dynamic modification to centralized wireless sensor networks," in *IEEE Sensors*, (New Zealand), pp. 1392–1395, October 2009.

[46] C. Huang, R. Cheng, T. Wu, and S. Chen, "Localized routing protocols based on minimum balanced tree in wireless sensor networks," in *the 5th International Conference on Mobile Ad-hoc and Sensor Networks (MSN '09)*, (China), pp. 503–510, 2009.

[47] M. Gunes, U. Sorges, and I. Bouazizi, "Ara: The ant-colony based routing algorithm for manets," in *Internatinal Conference on Parrallel Processing Workshops (ICPPW 02)*, (USA), pp. 79–85, 2002.

[48] G. Hu, P. Zhang, and W. Zhang, "The optimal design of tree structure based on ant colony of wireless sensor networks routing," in *the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC '09)*, (China), pp. 772–776, December 2009.

[49] S. Nourizadeh, Y. Song, and J. Thomesse, "An adaptive hierarchical routing protocol for wireless ad-hoc sensor networks," in *the 3rd International Conference on Next Genera-*

tion Mobile Applications, Services and Technologies (NGMAST '09), (UK), pp. 452 460, September 2009.

[50] K. Daabaj, M. Dixon, and T. Koziniec, "Experimental study of load balancing routing for improving lifetime in sensor networks," in *the 5th International Conference on Wireless communications*, (China), pp. 3471 3474, 2009.

[51] Z. Han, Z. Ji, and K. R. Liu, "Non-cooperative resource competition game by virtual referee in multi-cell ofdma networks," *IEEE Journal on Selected Areas of Communications*, vol. 25, p. 10791090, 2007.

[52] S. Frattasi, H. Fathi, and F. Fitzek, "4g: A user-centric system," *Special Issue on Advances in Wireless Communications: Enabling Technologies for 4G*, 2006.

[53] L. Wang, Y. Xue, and E. Schulz, "Resource allocation in multi-cell ofdm systems based on non-cooperative game," in *17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2006.

[54] J. Suris, L. DaSilva, Z. Han, and A. MacKenzie, "Cooperative game theory for distributed spectrum sharing," in *IEEE International Conference on Communications*, 2007.

[55] "Mica2dot cataloug." abvailabe at: `http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2DOT_Datasheet.pdf`.