Distributed TDMA Scheduling in Tree based Wireless Sensor Networks with Multiple Data Attributes and Multiple Sinks

by

TEJAS MUKESHBHAI VASAVADA
201121011

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

December, 2018

Declaration

I hereby declare that

i) The thesis comprises of my original work towards the degree of Doctor of Philosophy at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) Due acknowledgment has been made in the text to all the reference material used.

Tejas Mukeshbhai Vasavada

Certificate

This is to certify that the thesis work entitled DISTRIBUTED TDMA SCHEDULING IN TREE BASED WIRELESS SENSOR NETWORKS WITH MULTIPLE DATA ATTRIBUTES AND MULTIPLE SINKS has been carried out by TEJAS MUKESHBHAI VASAVADA for the degree of Doctor of Philosophy at *Dhirubhai Ambani Institute of Information and Communication Technology* under my supervision.

Prof. Sanjay Srivastava
Thesis Supervisor

Dedicated to my family

Acknowledgements

I am heartily thankful to my thesis supervisor Prof. Sanjay Srivastava for his constant guidance, support and motivation. He is always ready to discuss with me and clear my doubts. Without his guidance, I would not be able to produce this work.

I extend my thanks to my research progress committee members, Prof.V.Sunitha and Prof. Srikrishnan Divakaran. Their suggestions during Research Progress Seminars helped me to find appropriate directions for research work.

I can't forget Dr. Manish Chaturvedi, my senior in PhD studies (and now faculty member at Pandit Deendayal Petroleum University, Gandhinagar). He has been always ready to answer my technical questions. His moral support made my task easier.

I am also thankful to DA-IICT help-desk people who helped me to use high speed computing servers to execute simulations.

I thank my wife Rutvi who gave me moral support to continue my studies. Lastly, I thank my parents and almighty god whose blessings are always with me.

Contents

Abstact

Data collection is an important application of wireless sensor networks. Sensors are deployed in given region of interest. They sense physical quantity like temperature, pressure, solar radiation, speed and many others. One or more sinks are also deployed in the network along with sensor nodes. The sensor nodes send sensed data to the sink(s). This operation is known as convergecast operation.

Once nodes are deployed, logical tree is formed. Every node identifies its parent node to transmit data towards sink. As TDMA (Time Division Multiple Access) completely prevents collisions, it is preferred over CSMA (Carrier Sense Multiple Access). The next step after tree formation is to assign time slot to every node of the tree. A node transmits only during the assigned slot. Once tree formation and scheduling is done, data transfer from sensors to sink takes place.

Tree formation and scheduling algorithms may be implemented in centralized manner. In that case, sink node executes the algorithms and informs every node about its parent and time-slot. The alternate approach is to use distributed algorithms. In distributed approach, every node decides parent and slot on its own. Our focus is on distributed scheduling and tree formation.

Most of the researchers consider scheduling and parent selection as two different problems. Tree structure constrains efficiency of scheduling. So it is better to treat scheduling and tree formation as a single problem. One algorithm should address both in a joint manner. We use a single algorithm to perform both i.e. slot and parent selection. The main contributions of this thesis are explained in subsequent paragraphs.

In the first place, we have addressed scheduling and tree formation for single-sink heterogeneous sensor networks. In a homogeneous network, all nodes are of same type.

For example, temperature sensors are deployed in given region. Many applications require use of more than one types of nodes in the same region. For example, sensors are deployed on a bridge to monitor several parameters like vibration, tilt, cracks, shocks and others. So, a network having more than one types of nodes is known as heterogeneous network.

If all the nodes of network are of same type, the parent selection is trivial. A node can select the neighbor nearest to sink as parent. In heterogeneous networks, a node may receive different types of packets from different children. To maximize aggregation, appropriate parent should be selected for each outgoing packet such that packet can be aggregated at parent node. If aggregation is maximized, nodes need to forward less number of packets. So, less number of slots are required and energy consumption would be reduced. We have proposed AAJST (Attribute Aware Joint Scheduling and Tree formation) algorithm for heterogeneous networks. The objective of the algorithm is to maximize aggregation. The algorithm is evaluated using simulations. It is found that compared to traditional approach of parent selection, the proposed algorithm results in 5% to 10% smaller schedule length and 15% to 30% less energy consumption during data transfer phase. Also energy consumption during control phase is reduced by 5%.

When large number of nodes are deployed in the network, it is better to use more than one sinks rather than a single sink. It provides fault tolerance and load balancing. Every sink becomes root of one tree. If finer observations are required from a region, more number of nodes are deployed there. That is, node deployment is dense. But the deployment in other regions may not be dense because application does not require the same. When trees are formed, tree passing through the dense region results in higher schedule length compared to the one passing through the sparse region. Thus schedule lengths are not balanced.

For example, trees are T_1 and T_2. Their schedule lengths are SH_1 and SH_2 respectively. Every node in tree T_i will get its turn to transmit after SH_i time-slots. If there is a large difference between SH_1 and SH_2, nodes of one tree (having large value of SH_i) will wait for very long time to get turn to transmit compared to the nodes of the other tree (having small value of SH_i). But if SH_1 and SH_2 are balanced, waiting time would be almost same for all the nodes. Thus schedule lengths should be balanced. Overall schedule length (SH) of the network can be defined as max(SH_1,SH_2). If schedule lengths are balanced, SH would also be reduced.

We have proposed an algorithm known as SLBMHM (Schedule Length Balancing for Multi-sink HoMogeneous Networks). It guides every node to join a tree such that the schedule lengths of resulting trees are balanced. Through simulations, it is found that SLBMHM results 13% to 74% reduction in schedule length difference. The overall schedule length is reduced by 9% to 24% compared to existing mechanisms. The algorithm results in 3% to 20% more energy consumption during control phase. The control phase involves transfer of control messages for schedule length balancing and for slot & parent selection. The control phase does not take place frequently. It takes place at longer intervals. So, additional energy consumption may not affect the network lifetime much. No change in energy consumption during data transmission phase is found.

The schedule lengths may be unbalanced also due to difference in heterogeneity levels of regions. For example, in one region, two different types of sensors are deployed. But in the other region, four different types of sensors are present. When heterogeneity is high, aggregation becomes difficult. As a result, more packets flow through the network. Thus schedule length of the tree passing through region of two types of nodes will have smaller schedule length than the tree passing through the region of four types of nodes.

We have proposed an algorithm known as SLBMHT (Schedule Length Balancing for Multi-sink HeTerogeneous Networks). It is an extension of SLBMHM. The proposed algorithm is capable of balancing schedule lengths no matter whether imbalance is caused due to difference in density or difference in heterogeneity. It is also evaluated through simulations. It is found that the SLBMHT algorithm results in maximum upto 56% reduction in schedule length difference, maximum upto 20% reduction in overall schedule length and 2% to 17% reduction in energy consumption per TDMA frame during data transfer phase. It results in maximum 7% more energy consumption during control phase. As control phase does not take place very frequently, increase in energy consumption during control phase can be balanced by reduction in energy consumption during data phase. As a result, network lifetime is going to increase.

List of Abbreviations

MAC	Medium Access Control
TDMA	Time Division Multiple Access
CSMA	Carrier Sense Multiple Access
GPS	Global Positioning System
DRAND	Distributed RANDomized Algorithm
DD-TDMA	Deterministic Distributed TDMA
RBCA	Receiver Based Channel Access
LBCA	Link Based Channel Access
GLASS	Grid based LAtin Square Scheduling
FlexiTP	Flexible TDMA Protocol
MOSS	Many to One Sensors to Sink
DICA	Distributed Algorithm for Integrated tree Construction and data Aggregation
TSCH	Time-Slotted Channel Hopping
DETA	Delay Efficient Traffic Aware
FTS	Fault Tolerant Slot
MFS	Multi Function Slot
AAJST	Attribute Aware Joint Scheduling & Tree Formation
SLBMHM	Schedule Length Balancing for Multi-sink HoMogeneous networks
SLBMHT	Schedule Length Balancing for Multi-sink HeTerogeneous networks

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents an introduction to the problem domain. Here the data collection process is explained. Effectiveness of data collection process depends on choice of topology control method and MAC protocol. Different topology control methods like tree and cluster are explained. Two important MAC protocols, CSMA (Carrier Sense Multiple Access) and TDMA (Time Division Multiple Access) are explained and compared. As our work is about multi-sink heterogeneous networks, this chapter also explains importance of deploying multiple sinks and need of heterogeneous networks.

1.1 Data Collection

Sensors are tiny electro-mechanical devices. They can sense different physical quantities. They are placed in the area where parameters of interest are to be measured. There are various types of sensors like temperature sensor, pressure sensor, humidity sensor, solar radiation sensor and many more. As mentioned in [1], some real deployments of sensor networks are for habitat monitoring ([2]), environmental monitoring ([3],[4]), volcano monitoring ([5],[6]), water monitoring ([7]), civil health monitoring ([8],[9]) and weather monitoring ([10]).

The sensors sense the environment and send the readings to the sink. As mentioned in [12], this many-to-one data transfer is known as convergecast. The opposite to convergecast is broadcast (or multicast) where source is one and destinations are more than one.

The data collection operation may be periodic or aperiodic. In periodic data collection, sensors send readings after every fixed period. In aperiodic data collection, sensors send the readings when asked by sink or when a certain event occurs. For example, temperature sensors are continuously sensing temperature of a region. In periodic data collection, temperature values are sent to sink after certain specific period. But in aperiodic collection, sensors send the reading only when temperature crosses certain threshold.

Data transfer from sensors to sink can take place in following two different methods: (i) Multihop data transfer (ii) Using mobile nodes. In case of multihop data transfer, a path is established from data sending sensor to the sink. The path consists of one or more other sensors. Data generated by the source sensor is forwarded by nodes in the path towards the sink. In the other method, some mobile node visits all the sensors one by one and collects the data. Finally, it goes to the sink and submits the data collected from all the sensors.

Mobile element based data gathering is surveyed in detail in [11]. The main research issues are : (i) Minimizing data collection latency : Total time taken to collect data from all sensors should be minimized. (ii) Reducing mobile element discovery time: Every sensor has to discover presence of mobile node in its proximity. Then only data transfer can begin. Discovery time should be minimized to reduce overall data collection latency. (iii) Mathematical modeling of data collection: Data collection process is modeled as a queuing system. Different performance measures are analyzed using queuing theory. One such parameter is average waiting time i.e. how long a node has to wait until it can transfer packets to mobile element.

It is not always feasible to use mobile elements for data collection. For example, mobile elements may not be able to work in very harsh environment. Multihop forwarding is not affected by harshness of environment. But it requires reasonable density of nodes for the sake of connectivity. That is, every node should have at least one path to sink node. Otherwise some nodes may be disconnected. The readings sensed by them may not reach the sink node. Our focus is on multi-hop data transfer.

1.2 Topology Control

Formation of network topology is the primary requirement of multi-hop data transfer. Once nodes are deployed, some logical topology must be formed. Tree and Cluster are two well-known topology control mechanisms. In a cluster, nodes are divided into geographical groups. Each group has a group leader (known as cluster head). All nodes in the group send packets to cluster head. Cluster head may be directly connected to sink or it may send packets to sink via multi-hop path passing through other cluster heads. Within a cluster, a node may be directly connected to the cluster head or there may be a multihop path from node to the cluster head.

In Figure 1.1, cluster based topology is illustrated. It is seen that node filled with black color is a sink. Nodes filled with dashed lines are cluster heads. Cluster is represented as a circle. All nodes present in a circle are in same cluster. It is also possible that clusters may be overlapping. In that case, some nodes may be member of more than one clusters. This is true for nodes present in peripheri of a cluster.

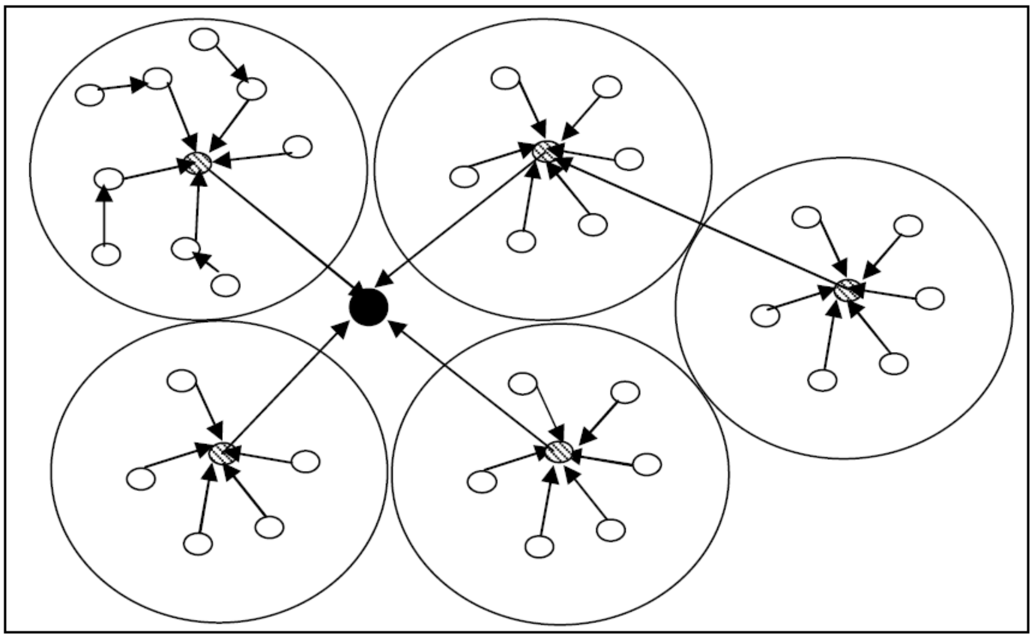


Figure 1.1: Illustration of Cluster based Logical Topology

In case of tree topology, sink node acts as the root of the tree. Every node selects one neighbor node as parent. Here neighbor means a node within radio radius. That is, at a single hop distance. The parent of a node is frequently nearer to sink i.e. at a smaller hop distance. Every node sends its packet to parent node. The parent forwards the packet to its parent. Thus packet finally reaches the sink. In Figure 1.2, tree based topology is shown. The figure is self explanatory.

It is seen from Figure 1.1 that node may have multihop path to cluster head. In other

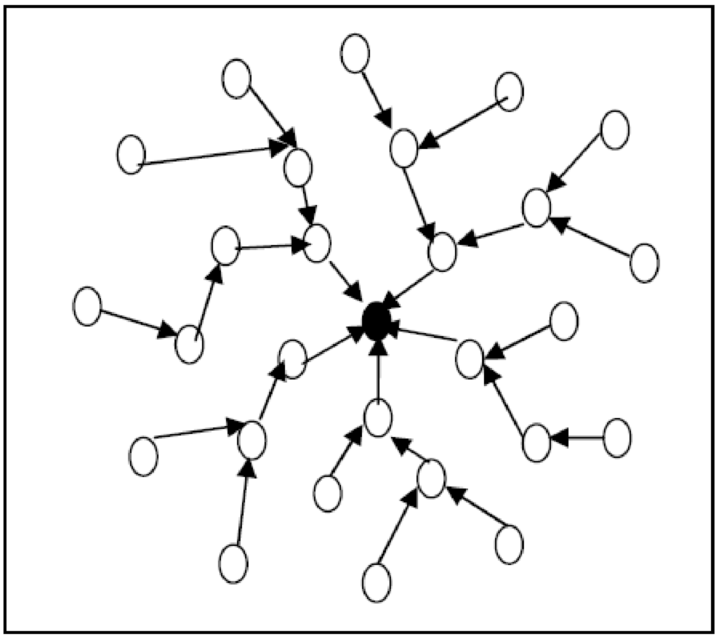


Figure 1.2: Illustration of Tree based Logical Topology

words, every cluster can be considered as a tree rooted at cluster head. Thus tree is a very basic topology. So, we are focused on tree based networks.

1.3 MAC Protocols

Once topology is formed, next issue is choice of Medium Access Control (MAC) mechanism. Following two methods are widely used in sensor networks: CSMA (Carrier Sense Multiple Access) and TDMA (Time Division Multiple Access). In case of CSMA, nodes contend for channel. Whereas in TDMA, every node is assigned one transmission slot. Node transmits to its parent in the assigned slot only. As nodes take turns to transmit, collisions do not occur. In contrast to CSMA, TDMA completely prevents collisions. If collision occurs, sender node has to retransmit its packet. Thus more energy is consumed.

TDMA results in better energy savings than CSMA. TDMA has some limitations also. During a TDMA cycle if a node has no packet to transmit, its time-slot remains unutilized. At low traffic load, TDMA results in wastage of bandwidth. It is meaningful to use TDMA when traffic at every node is moderate or high. In this work, we are focused on TDMA scheduling of tree based wireless sensor networks. Every node is assigned one or more time slots to transmit packets to the parent node.

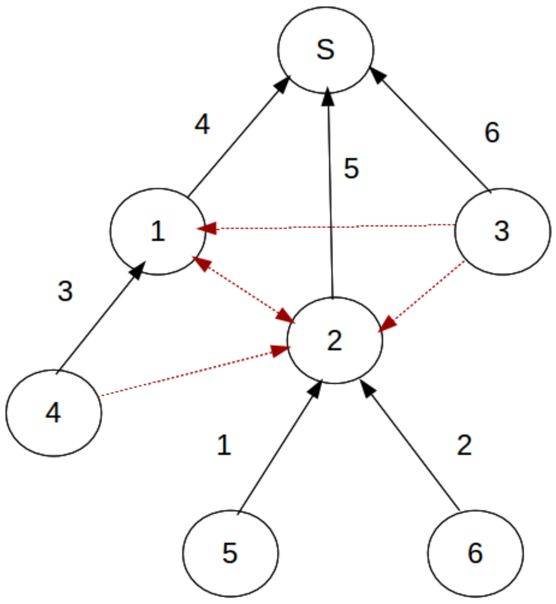


Figure 1.3: Illustration of Scheduling in Aggregated Convergecast [12]

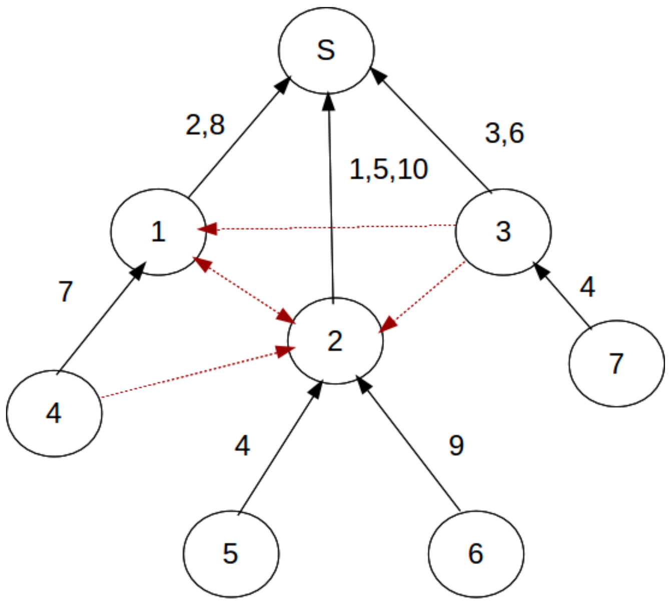


Figure 1.4: Illustraion of Scheduling in Raw Convergecast [12]

As mentioned earlier, data transfer from sensors to sink(s) is known as convergecast operation. As defined in [12], there are two types of convergecast: (i) Raw (ii) Aggregated. In raw convergecast, a node forwards all incoming packets to outgoing edge. If node has n incoming packets, it sends out $(n+1)$ packets. Here n packets have come from children and 1 packet is generated by node itself. So count is $(n+1)$. Whereas in aggregated convergecast, a node merges all incoming packets with its own packet and sends out only one packet. Possible aggregation functions are sum, count, average, max, min and many others. It depends on application whether raw convergecast is needed or aggregated convergecast. If every reading is important, raw convergecast should be used. If users are interested only in summarized data, aggregated convergecast should be used.

Figures 1.3 and 1.4 illustrate assignment of time-slots to edges of a tree in case of aggregated convergecast and raw convergecast respectively. Sink is denoted by S. Sink node is at the root of the tree. Straight lines indicate child-parent relationship. Whereas dashed line between two nodes indicate that both are in interference range of each other. In case of aggregated convergecast, every edge is assigned one time slot. The reason is that every node merges all incoming packets into one packet and sends out a single packet. In case of raw convergecast, every node forwards all incoming packets to output and also its own packet. Thus an edge is assigned multiple time slots. We are focused on TDMA scheduling of tree based wireless sensor networks for aggregated convergecast.

1.4 Centralized v/s. Distributed Algorithms

Once sensor nodes are deployed, tree formation and scheduling algorithms are executed. Based on place of execution, the algorithms can be classified as follows: (i) Centralized (ii) Distributed. The algorithms presented in [13], [14], [15], [16] and [17] are few examples of centralized algorithms. The centralized algorithms run at the sink node. Sink knows the entire topology. Sink executes scheduling and parent selection algorithms. Once parent and slot are decided for every node, sink has to spread this information within the network so that each and every node would know its parent and slot. In case of distributed algorithms, every node decides its parent and slot itself. Sink does not play any role. Every node should know IDs of its neighbors only, not entire topology.

Centralized algorithms are easy to implement but they require sink to know entire network topology. When nodes are randomly deployed, every node has to inform its location to sink. Thus GPS (Global Positioning System) support should be available at every node. But GPS coverage may not be available in some odd locations. The other way for sink to derive topology is that every node should send its neighborhood information to sink. The information may contain list of neighbors and estimated distance from each neighbor. Every node may broadcast a special packet known as PROBE packet. From signal strength of PROBE received from neighbor, distance from neighbor can be estimated. Based on neighborhood information received from all the nodes, sink could derive entire topology.

Distributed algorithms do not require sink to know entire topology. They have other challenges like avoiding race conditions. That is, more than one nodes may attempt parent/slot selection simultaneously. They end up selecting the same slot which causes interference at other nodes. Distributed algorithms need to prevent racing. This requires message exchange between neighbors. Thus extra traffic is generated.

When there is a change in topology i.e. new nodes are added or some of existing nodes die, tree-repairing and rescheduling both are required. In case of centralized algorithms, sink has to perform tree-repairing and rescheduling because sink has originally generated tree and schedule. But in distributed algorithms, repairing and rescheduling can be done locally.

As such lifetime of a network consists of repeated cycles of two phases: control phase and data phase. During control phase, scheduling and tree formation takes place. During data phase, actual data is transferred from nodes to sink. The control phase takes place after longer intervals when substantially large number of nodes die or new nodes are added. That is, only when complete rescheduling is required. In both centralized and distributed algorithms, extra traffic is generated during control phase. In centralized algorithms, nodes have to propagate location information to sink and sink has to send slot/parent information back to nodes. In distributed algorithms, neighbors need to perform some message exchange for selection of collision-free slots. But when few nodes die, distributed algorithms seem more effective because repairing can be done locally.

Centralized algorithms can not capture real-time interference relationships. Due to advantages like scheduling and tree formation using knowledge of only neighbors and local repairing, we are focused on distributed scheduling and tree formation algorithms.

Centralized algorithms may be used for bench-marking.

1.5 Homogeneous v/s. Heterogeneous Networks

In a homogeneous network, only one type of nodes are present in the entire region of interest. For example, temperature sensors are deployed in an area to sense variation in temperature.

In many real applications, it is required to sense more than one quantities at the same time and place. For example, sensors are deployed on a bridge to monitor several parameters like vibration, tilt, cracks, shocks and others. A single type of sensor may not be able to sense all the parameters. So different types of sensors need to be deployed. Often sensor networks have more than one type of nodes present. For example, temperature sensors, pressure sensors, solar radiation sensors all are present in a single network. Such a network is known as heterogeneous network.

Different types of sensors are deployed together. They are part of the same tree. So, one type of sensor has to forward packets coming from different types of sensors. In homogeneous networks, every node may receive multiple packets (one packet from each child), but it sends out only packet. All incoming packets can be aggregated with node's own packet. But in heterogeneous network, perfect aggregation is not possible. If a temperature sensor receives a pressure reading, temperature and pressure readings can not be aggregated. For raw convergecast, every node has to forward all incoming packets in both homogeneous and heterogeneous networks.

As mentioned in [45], often term 'Attribute' is used to refer to the type of reading present in the packet. That is, temperature reading and pressure reading are considered two different attributes. If a network has two types of nodes present (e.g. temperature sensors and pressure sensors), it can be said that two different attributes are present in the network.

1.6 Multiple sinks Networks

When large number of nodes are present in the network, it is desirable to deploy more than one sinks. If only one sink is present, all the nodes would send packets to that sink only. In other words, all the nodes would join the same tree i.e. rooted at the only sink. As network size grows, size of the tree in terms of diameter would also grow. So packets generated by nodes far from root node would take long time to reach the sink.

If multiple sinks are deployed, workload can be distributed between sinks. If two sinks are deployed, some nodes would join the tree rooted at first sink and the others may join the tree rooted at second sink. Thus when multiple sinks are present, size of an individual tree is reduced. Thus time required to deliver packets to root is also reduced. Other advantage of multiple sinks is fault tolerance. If one sinks fails, nodes may start sending data to different sink.

In this work, we have considered presence of multiple sinks in the network.

1.7 Major Contribution

We have worked on distributed TDMA scheduling of multi-sink tree based heterogeneous wireless sensor networks. Following are main contributions of this thesis:

1. Parent selection method for heterogeneous sensor networks is proposed. The objective is to maximize aggregation. The method is incorporated into joint scheduling and parent selection proposed in [21]. The algorithm proposed in [21] is for homogeneous networks. It is modified to work with heterogeneous networks. It is suggested that every node should forward different type of packet to different node. That is, node should select parent based on type of packet. Our proposed algorithm is named as AAJST (Attribute Aware Joint Scheduling & Tree formation). It is shown through simulations that proposed algorithm results in reduction in schedule length and better energy conservation.

2. Schedule length balancing algorithm is proposed to balance schedule lengths of trees in case of multi-sink homogeneous networks considering that node distribution can be non-uniform. When node distribution is not uniform, trees rooted at different sinks may be of different size. As a result, their schedule lengths are likely to be different. The proposed algorithm guides every node to decide which tree (i.e. sink) to join such that when actual scheduling & tree formation algorithm runs, it results in balanced schedule lengths across the trees. As a final result, overall schedule

length of the network is reduced. The proposed algorithm is named as SLBMHM (Schedule Length Balancing for Multi-sink HoMogeneous networks).

3. It is possible that even if node distribution is uniform in the network, different trees have different levels of heterogeneity. That is, one tree has two different types of nodes. The other tree has four different types of nodes. It is likely that first tree has smaller schedule length than the second one. Algorithm proposed in step 2 above is extended to balance schedule lengths of heterogeneous multi-sink networks. The resulting algorithm is known as SLBMHT (Schedule Length Balancing for Multi-sink HeTerogeneous networks).

1.8 Summary

In this chapter, data collection in sensor networks is introduced. Different topology control mechanisms and MAC protocols are discussed. Merits and demerits of centralized and distributed algorithms are also mentioned. Importance of heterogeneous networks is explained. Advantages of using multiple sinks are presented. At last, major contributions of the thesis are enlisted.

1.9 Organization of Thesis

Organization of the rest of the thesis is as follows: Related literature is reviewed in Chapter 2. In Chapter 3, problem definition is discussed along with all assumptions and objectives. The scheduling & tree formation in single sink heterogeneous networks is presented in Chapter 4. The schedule length balancing algorithms for homogeneous and heterogeneous multi-sink networks are presented in Chapter 5 and 6 respectively. The thesis is concluded in Chapter 7.

Chapter 2

Related Work

In this chapter, a review of related literature is presented. We are focused on distributed scheduling and tree formation algorithms in single-sink and multi-sink networks. In the first subsection, distributed scheduling algorithms for single sink networks are presented. Then mechanisms related to fault tolerance are discussed. Scheduling and tree formation for multiple sinks networks is presented next. It is followed by discussion of Research Gap.

2.1 Classification of Scheduling Algorithms

In Figure 2.1, classification of scheduling algorithms is given. The scheduling algorithms are categorized based on the type of convergecast addressed by them. Following are three categories: (i) Algorithms addressing aggregated convergecast (ii) Algorithms addressing raw convergecast (iii) Algorithms which can be adapted for use with any of the two types of convergecast i.e. General Algorithms.

Algorithms addressing aggregated convergecast assign single transmission slot to every node. As mentioned in [21], slot assignment is preferred to be bottom to top i.e. from leaf to root. That is, every parent node is assigned higher time slot than children. In aggregated convergecast, parent aggregates packets coming from children with its own packet and sends out a single packet. If time slot assigned to the parent is lower than children, parent can forward the aggregated packet only in next TDMA cycle. But if parent is assigned higher time slot, aggregated packet can be forwarded in the same cycle. Thus packet latency can be controlled by bottom-top slot assignment.

Raw convergecast requires assignment of multiple slots to non-leaf nodes. In raw convergecast, every non-leaf node forwards all packets of its children. Thus every non-leaf node needs more than one slot. In contrast to raw convergecast, slot assignment may proceed in top to bottom manner. Parent node need not wait for children to send their packets. Parent may send its own packet early. Packets coming from children may be forwarded as and when they arrive. Thus packet generated by parent is not unnecessarily delayed.

Algorithms categorized under General category do not address any specific type of convergecast. Even most of them are not designed for tree based networks. But they are targeted towards other issues like reducing control overhead of slots selection, use of multiple channels for better slot reuse etc.. The scheduling should take place in bottom-up manner in aggregated convergecast. In raw convergecast, scheduling should be done in top-down fashion. The methods categorized as General methods are not tuned to any specific convergecast. But they can be changed to work with any of the two types of convergecasts.

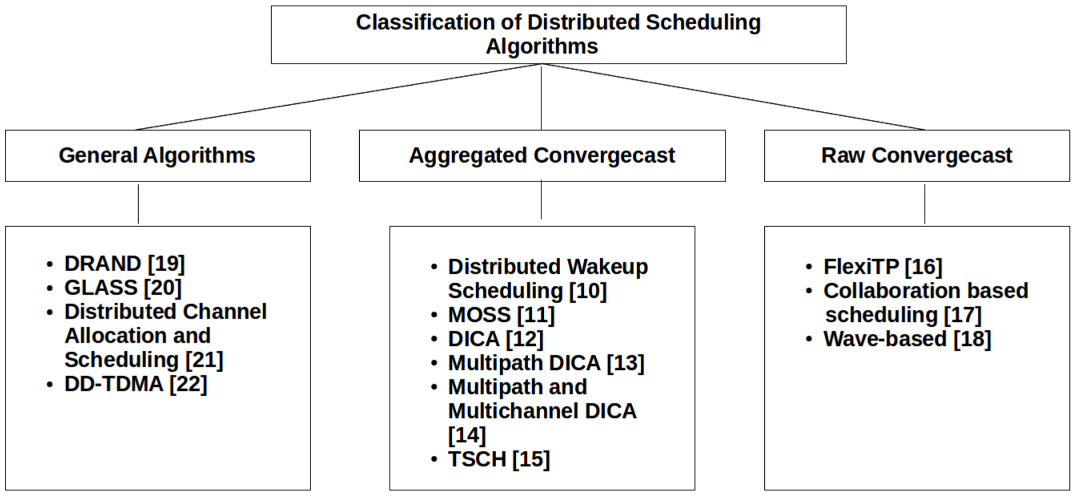

Classification of Distributed Scheduling Algorithms
General Algorithms
Aggregated Convergecast
Raw Convergecast
• DRAND [19]
• GLASS [20]
• Distributed Channel Allocation and Scheduling [21]
• DD-TDMA [22]
• Distributed Wakeup Scheduling [10]
• MOSS [11]
• DICA [12]
• Multipath DICA [13]
• Multipath and Multichannel DICA [14]
• TSCH [15]
• FlexiTP [16]
• Collaboration based scheduling [17]
• Wave-based [18]

Figure 2.1: Classification of Distributed Scheduling Algorithms

2.1.1 General Approaches

In [28], DRAND (Distributed RANDomized Algorithm) is proposed. Every node selects a time slot using three messages namely REQUEST, GRANT and RELEASE. First, node broadcasts REQUEST message. Upon receiving REQUEST message, its neighbors respond with GRANT message. The GRANT message sent by a node contains its identifier and list of slots used in its neighborhood. Thus the given node receiving GRANT message can prepare a list of slots already occupied by its two hop neighbors. Based on this list, it selects the lowest slot not used by any of the two hop neighbors. To inform its choice of slot, the given node broadcasts a message termed as RELEASE message.

The core idea of DD-TDMA(Deterministic Distributed TDMA) proposed in [31] is to save the energy consumed due to frequent on/off switching. Scheduling algorithm is proposed such that the gap between the transmission and reception slots of a node is minimized. As the gap is very small, node does not go to sleep state between reception and transmission slot. But it remains in idle listening state. This results in energy savings. It is also shown that DD-TDMA results in smaller schedule length than DRAND. It also results in smaller running time and smaller control message overhead.

Distributed protocols for channel and slot allocation are presented in [30]. Two different approaches of channel allocation namely Receiver Based Channel Assignment (RBCA) and Link Based Channel Assignment (LBCA) are suggested. Both the approaches are based on idea of conflict graph. In the conflict graph, interfering vertices are connected by an edge. In RBCA, interfering vertices are receiver nodes. But they are senders in LBCA. The interfering nodes are assigned different channels. The use of multiple channels results in smaller schedule length.

All the distributed algorithms require control message exchange between neighbors for collision-free schedule formation. These control messages are an extra overhead. They also result in extra energy consumption. To reduce the control overhead, a protocol named as GLASS (Grid based LAtin Square Scheduling) in proposed in [29]. The description of the entire protocol would be too lengthy. So, only main objective is covered here.

2.1.2 Aggregated Convergecast

In [24], bottom-up scheduling algorithm for IEEE 802.15.4e [47] standard is proposed. Originally, 802.15.4 [47] does not include provision of time-slotted transmission. The IEEE 802.15.4e recommends use of Time-Slotted Channel Hopping (TSCH) i.e. use of slots and multiple frequencies. But it does not define how the slots should be assigned to nodes. A scheduling algorithm is proposed for the same in [24]. It is assumed that tree is formed using RPL[46] algorithm. At the end of tree formation process, every node is assigned a location based ID of following format: (x_1,x_2,....,x_h). Here h is height of tree. If given node is at level l, the value of x_l is position assigned by parent to given node. For $i > l$, x_i is 0. When $i < l$, x_i contains positions of ancestor nodes of given node. Scheduling takes place after tree is formed. Every node selects its time-slot on its own. The time slot selection is implemented as a function of ID of the node. Conflicting nodes select difference frequencies to avoid collisions.

In [19], two different algorithms are proposed. One is for tree formation and other is for slot selection. Tree formation works as follows. It works in top down manner. That is, nodes nearer to sink join the tree first. From candidate parents (i.e. those who have already joined the tree), given node chooses as parent the node with the least interference. This idea results in better slots reuse. Slot selection also works in top-down manner. Every node selects a slot smaller than slot of parent node. Slot selection is such that collision during packet transfer is avoided.

Normally tree formation and scheduling are implemented as two separate algorithms. First tree is formed and then slots are assigned. The other approach is joint scheduling & tree formation. In joint approach, every node selects slot and parent at the same time.

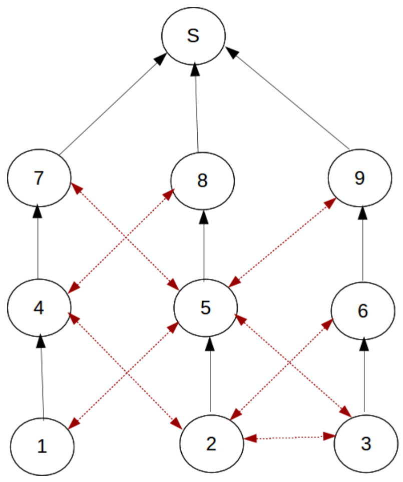


Figure 2.2: Sample Node Deployment to Illustrate Joint Approach [21]

In [21], DICA (Distributed algorithm for Integrated tree Construction and data Aggregation) is proposed. It uses joint approach. As mentioned in DICA, if tree is formed first, structure of tree puts limit on the performance of scheduling algorithm. In other words, two different trees formed over the same node deployment may result in different

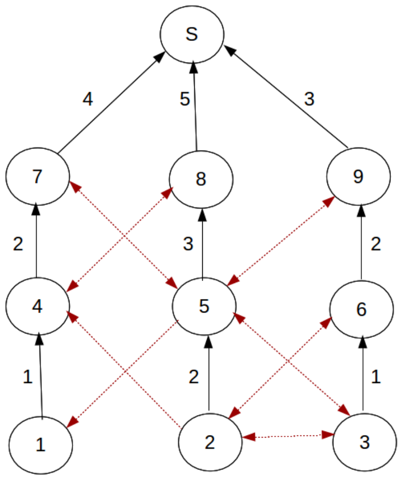


Figure 2.3: Sequential Approach: First Tree then Schedule [21]

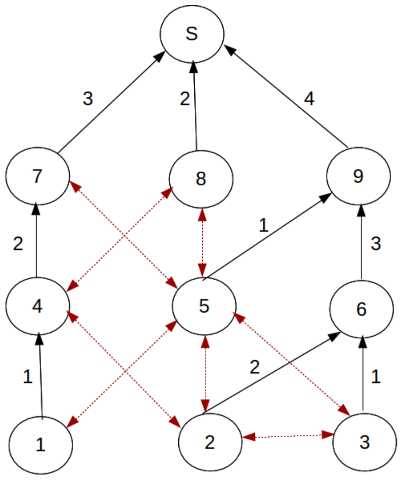


Figure 2.4: Joint Approach: Tree and Schedule Formation Together [21]

schedule lengths. In Figure 2.2, a sample deployment is shown. Continuous line from one node to the other indicates that former has selected later as parent. Dashed line (in red color) between two nodes indicate that the nodes are in radio range of each other.

In Figure 2.3 and 2.4, time slots are assigned to every node so that node can send packet to its parent during that time-slot. Time slots are assigned considering following points: (i) Parent is always assigned higher time-slot than children. This ensures aggregation freshness. It means that parent could forward packets coming from children in the same TDMA cycle. (ii) Collisions do not occur. That is, no node receives from more than one nodes in the same time slot.

The trees shown in Figures 2.3 and 2.4 are built over the same deployment of nodes. But both are different. Their schedule lengths are also different. In Figure 2.3, the highest time slot is 5. Whereas in Figure 2.4, it is 4.

As mentioned earlier, the core idea behind joint scheduling & tree formation is that every node should select slot and parent at the same time. Node should first identify the lowest possible time slot. This time slot meets following conditions: (i) It is higher than time-slots of children. (ii) No neighbor is receiving or overhearing in the slot. This condition prevents collisions.

Now node prepares a list of candidate parents. The candidate parents are the neighbor nodes which do not receive, overhear or transmit in the selected slot. If the list is empty, node has to try for next higher slot. If list is non-empty, the candidate parent with the least number of unscheduled neighbors should be selected as parent. If there are more than one candidate parents with the same number of unscheduled neighbors, node ID may be used to resolve the clash. The node with lowest ID may be selected as parent.

Thus every node will select the lowest possible time slot and the parent to whom it can transmit in the selected slot. This would reduce overall schedule length.

We have used DICA[21] for our research work. So, steps of the same are explained below in detail:

1. A special control packet known as HELLO packet is flooded in the network by the sink node. The HELLO packet contains a field 'LEVEL'. It is initialized to 0 by sink.

2. Every node receiving HELLO packet sets its level L to $LEVEL + 1$. Then it adds 1 into $LEVEL$ field and broadcasts the HELLO again. Thus this step helps every node to find its distance in hop count from the sink i.e. level.

3. Scheduling & parent selection proceeds in bottom-up fashion. A node at level L selects a slot & parent only after its neighbors at level $(L + 1)$ have completed slot & parent selection.

4. For the given minimum available slot TS, node prepares a candidate parent set, *candp_set*. The candidate parent set for slot TS is the set of neighbor nodes which do not transmit, receive or overhear in the slot TS.

5. From *candp_set*, one node p is found such that count of unscheduled nodes of p is the least compared to all other nodes of *candp_set*. The given node sets node p as

parent.

6. The given node broadcasts a message known as REQUEST message. The REQUEST message has two fields: selected slot and name of parent.

7. Upon receiving a REQUEST message, each candidate parent of the sender send REPLY message. The REPLY can be positive or negative. Assume that one of candidate parent is c. If some other node has already selected same slot as given node to transmit data to node c, the node c sends negative REPLY message to the given node. Otherwise, positive REPLY is sent.

8. If positive REPLY comes from each & every candidate parent, SCHEDULE message is broadcasted by the given node. The SCHEDULE message confirms slot & parent selection. Like REQUEST message, it has two fields: selected slot and name of parent.

9. If even a single REPLY is negative, node repeats all the steps starting from step 4. It tries to select next higher slot.

10. When a node receives a SCHEDULE message, it broadcasts a message termed as FORBIDDEN message. It has two fields: isparent and slot number. The isparent is 1 if sender of FORBIDDEN is selected as parent. Otherwise, it is 0. The slot number is the slot in which it is going to receive or overhear. Through FORBIDDEN message, neighbors are informed that the sender of FORBIDDEN message is going to receive/overhear a packet in the specified slot.

In Figure 2.5 slot & parent assignment as done by DICA [21] is illustrated. Scheduling and parent selection starts with leaf nodes. At last, one hop neighbors of sink are scheduled. Continuous edge between two nodes indicate child-parent relationship. A dashed edge (in red color) between two nodes indicate that they are in interference range of each other.

In [20] also, a joint scheduling & parent selection algorithm is proposed. It is termed as MOSS (Many to One Sensors to Sink). Unlike DICA[21], MOSS proposes top-down approach for scheduling & tree formation. Given node may have multiple candidate parents. It selects as parent the node which is at smallest distance form it. Thus transmission power can be saved.

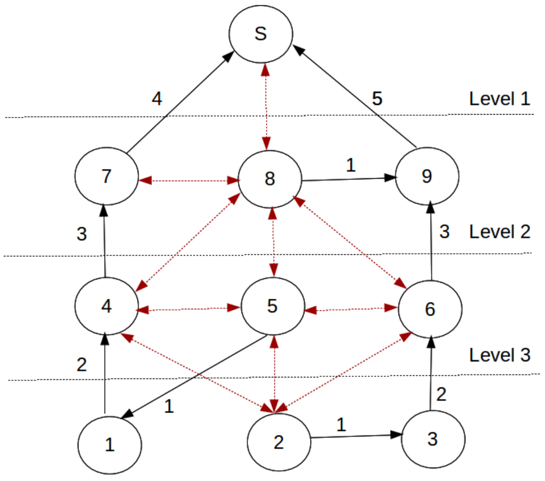


Figure 2.5: Illustraion of Bottomp-Up Scheduling and Tree Formation [21]

Paper	Approach	Parent selection criteria
Distributed Wakeup Scheduling [19]	Sequential	Interference
MOSS [20]	Joint (top-down)	Distance from given node
DICA [21]	Joint (bottom-up)	No. of unscheduled neighbors

Table 2.1: Summary of Aggregation Convergecast Scheduling Algorithms

The DICA[21] is extended in [22] and [23]. In [22], DICA is extended such that more than one parents are selected by each node. Whereas in [23], joint scheduling & tree formation using multiple paths and multiple channels is done. The use of more than one parents (i.e. paths) ensures fault tolerance and load balancing. The use of multiple channels results in smaller schedule length compared to single channel scheduling.

Discussion

In Table 2.1, different approaches of aggregation convergecast scheduling are summarized with key features. The approach of DICA[21] is more appropriate than the other two algorithms. Following are the reasons: (1) In DICA, node selects such a parent from its neighbors to whom it could transmit in the smallest possible time slot. The parent may be at the same level as the given node, one hop near to sink than given node or may be one hop far from the sink than the given node. In the other two approaches, parent must be one hop near to sink than the given node. As DICA is focused on selecting any neighbor as parent which can receive in the smallest selected slot, it is likely to result in the smallest schedule length compared to the other two algorithms. (2) Distributed scheduling algorithm proposed in [19] uses sequential approach. As explained earlier, joint approach is better than sequential approach. The MOSS [20] is based on joint approach. But it works in top-down manner. The top-down approach is not suitable for heterogeneous networks. In heterogeneous networks, all incoming packets need not be aggregated with given node's packet. Thus multiple packets can come out from given node. To identify the total count of outgoing packets and type of each outgoing packet, node must know the count of incoming packets and type of each incoming packet. This is possible only if nodes are scheduled from leaf to sink i.e. bottom to top. Thus in heterogeneous networks, bottom-up scheduling is more appropriate compared to top-down scheduling.

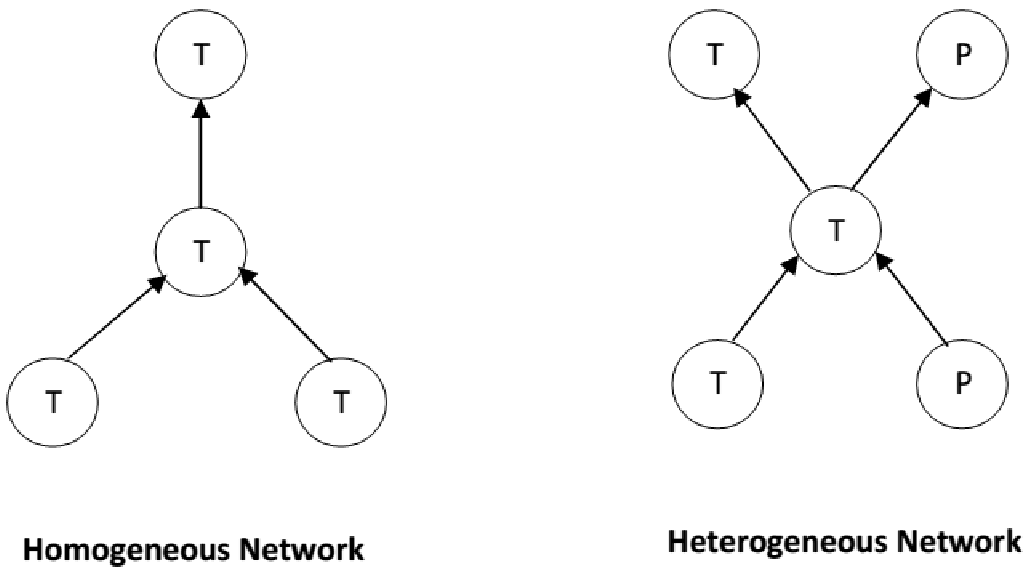


Figure 2.6: Illustration of Difference between Homogeneous and Heterogeneous Networks

We could not find any distributed scheduling algorithm designed specially for heterogeneous networks. Scheduling & parent selection should be done differently in heterogeneous networks than homogeneous networks.

In Figure 2.6, homogeneous and heterogeneous networks are presented. In homogeneous network, all the nodes are of same type. But in heterogeneous networks, different types of nodes are present. Homogeneous network has all temperature sensors (denoted by T). But heterogeneous network has pressure sensors (denoted by P) and temperature sensors.

In homogeneous network, perfect aggregation is possible at every node. For example, in Figure 2.6, the temperature sensor in the middle receives two temperature packets. It will aggregate those packets with its own packet and will send out only one packet. But perfect aggregation is not possible at every node in heterogeneous network. As shown in Figure 2.6, the temperature sensor in the middle receives one temperature packet and one pressure packet. It can not aggregate pressure packet with its own temperature packet. So, it has to send out two packets: one temperature packet and the other is pressure packet. As shown in the figure, node may send each outgoing packet through a different parent node to maximize aggregation. The temperature packet can be forwarded to the temperature sensor. The pressure packet can be forwarded to the pressure sensor. Thus multiple pairs of slots and parents are required.

We consider that a joint scheduling & parent selection algorithm for heterogeneous networks may be designed based on the idea mentioned above. The objective is to maximize aggregation.

2.1.3 Raw Convergecast

In [25], a protocol known as FlexiTP (Flexible TDMA Protocol) is proposed. The protocol addresses three aspects: parent selection, slot selection and topology maintenance. First phase is tree formation (i.e. parent selection). The scheduling (i.e. slot selection) takes place second. At the beginning of tree formation, sink generates a special packet termed as ‘token’. It sends the token to the neighbor whose ID is the smallest among the other neighbors. That neighbor becomes child of the sink. Then that node broadcasts a special beacon message. The nodes which receive beacon message become children of the sending node. Now the sending node generates token and gives to the child with the smallest ID. Again that child broadcasts beacon message. This is how parent-child relationship is established and tree is formed. At the end, the token reaches back to sink. Now next phase i.e. slot selection begins.

During slot assignment also token is used in the same way as tree formation. Slot selection is performed by a node if it has the token. After selecting a slot, node sends the ID of selected slot to one hop and two hop neighbors. This helps the neighboring nodes to select slots such that data transmission does not create collisions. As scheduling takes place in top-down manner, nodes may have to perform slot selection more than once.

Whenever a node at level l selects a slot, all its ancestors from level $(l-1)$ till the sink must select a slot to forward the packet coming from the given node.

In [26], collaboration based distributed scheduling is presented. Every node knows the number of packets received from children and number of packets generated by itself. Accordingly it calculates number of required transmission slots. The given node selects a sequence of slots based on its knowledge of slots used by neighbors. It broadcasts a packet known as REQUEST packet. The REQUEST packet contains the list of selected slots. The REQUEST packet is received by all the neighbors of the given node. If all the selected slots are unique i.e. are not going to create interference at neighbors, all the neighbors send GRANT message back to the given node. If any of the selected slot is going to create interference at any of the neighbors, the concerned neighbor does not send GRANT message. If GRANT is received from all the neighbors, the given node confirms the slots by broadcasting ACK message. Else it selects different slots and repeats the process.

In [27], a joint channel and slot assignment is proposed. Every node is assigned multiple channels and slots. As more than one channels are assigned to node, node can transmit using lowest possible time slot. The slot selection process is similar to the one used in previously described algorithms. That is, node selects a slot based on the slots used in its two hop neighborhood. In the beginning, sink assigns slots and channels to every node. For this, sink must know traffic requirement and complete topology. As data generation rate (i.e. traffic requirement) may change with time, nodes themselves select extra slots if required. Thus initial slot and channel assignment is done centrally but later slot adjustment is done locally.

2.2 Fault Tolerance and Adapting to Workload Variation

Sensors are battery operated devices. They loose energy with time due to transmission and reception of packets. It is possible that some nodes become non-functional because energy is completely drained. If parent of given node dies, the node has to select new parent. Now not only the given node but also all the nodes in the sub-tree rooted at the given node transmit the packets via new parent. As a result, now that parent node has to forward more number of packets. So, it has to select additional slots. This situation can happen in raw convergecast or in aggregated convergecast in heterogeneous networks.

Often new nodes are added in the network after initial deployment to ensure full coverage and connectivity. As newly added nodes join the tree, more packets pass through the tree. As a result, additional slots need to be selected. Many applications like environmental monitoring require the sensors to continuously sense the environment. Data is transmitted at some fixed time period. But if some important event occurs, data need to be transmitted at an increased frequency. Thus more slots are required.

From above paragraphs, it can be concluded that scheduling & parent selection is not just one time process. When some nodes fail or new nodes are added or traffic requirement changes, slot & parent selection should be executed. In other words, tree repairing and schedule adjustment is done. It is required that tree repairing and schedule adjustment should be done quickly so that data collection operation does not suffer. Also the control overhead of maintenance should be as much low as possible.

In [40], an approach to handle dynamic traffic pattern is proposed. In many applications, nodes continuously transmit data i.e. at fixed frequency. So, traffic pattern remains static. But in some applications, it is not required to send every reading. For example, nodes needs to transmit when some event has taken place or there is a significant difference between the readings sent previously and the readings sensed currently. In such cases, traffic is not generated at a constant rate. But slots are assigned assuming that nodes always have data to send. If a node does not transmit in the allocated slot, its parent keeps waiting and spends energy in unnecessarily sensing the channel.

The work done in [40] addresses the above problem. It is suggested that a parent should be scheduled after all its children. As a result, parent will transmit its packet only after all its children have sent the packets. Every node is assigned a series of time slots to transmit its own data and data coming from its children. The time slots assigned to a node are such that first it would forward packets coming from children and then it would forward its own packets. It is possible that children of the given node have no data to transmit. In that case, node will transmit its own data in the initial time slots and then would go to sleep. The parent of the given node when receives data belonging to the given node in initial slots would understand that the node does not have more data to send.

So, the parent also goes to sleep once the given node finishes the transmission. Thus the parent node does not waste its energy in waiting for the child to send the packets.

In [41], an improvement over the approach suggested in [40] is presented. The algorithm is named as DETA (Delay Efficient Traffic Adaptive) algorithm. In [40], packets generated by a node face longer delay as the node is scheduled after its parents. The DETA algorithm aims to reduce this delay. It also results in smaller schedule length compared to [40]. The details are not presented here but only main objective is mentioned.

In [42], an idea to quickly recover from node failure is presented. The idea is that every node should not select one pair of slot and parent. Instead, it should find redundant pairs of slot and parent. Whenever current parent fails, node should start transmitting via alternate parent in the respective slot. The benefit is that node need not perform slot and parent selection at the time of parent failure. Alternate slot-parent pair is selected at the time of scheduling & parent selection itself. Thus tree repairing and schedule maintenance is done faster. The demerit of this approach is increase in number of slots required to schedule the tree. As a result, schedule length also increases. As redundant slots are reserved by nodes, more number of slots are needed to schedule the tree.

In [43] also similar idea is presented. The authors of [43] are not focused on scheduling. It is suggested in [43] that during tree formation itself alternate parent should be identified. When actual parent dies, node should automatically switch to the alternate parent.

In [44], an improvement over the scheme suggested in [43] is proposed. In [43], tree repairing is pro-active in nature. It means that before the failure of current parent, alternate parent is already found. This idea does not work well when multiple nodes fail. For example, both current parent and alternate parent fail. To quickly recover from failure of multiple nodes, notion of redundant nodes is proposed. It is proposed that in a network, some nodes are active and others are redundant. The active and redundant both the types of nodes are part of the tree. The data sensing is done only by the active nodes. The deployment is such that around each active node few redundant nodes are present. Whenever parent of an active node fails, that active node starts sending data through the nearby redundant node. As redundant node is already connected to the tree, data collection operation does not suffer due to failure of the parent node.

In FlexiTP([25]) also fault tolerance is addressed. Four different types of slots are proposed: FTS (Fault Tolerance Slots), MFS(Multi Function Slots), Transmission and Reception Slots. To recover from faults, FTS are used. The MFS are used by a node to synchronize time with the children and share scheduling information with the children. If a node does not receive any packet from its parent for two back to back Multi Function Slots, it is an indication that parent is failed. So, the given node sends a beacon message during next Fault Tolerant Slot. The beacon is heard by all the neighbors. They all will send an ACK back to the given node. The given node selects as parent the neighbor which is nearest to the sink. The given node intimates the new parent. The newly selected parent node assigns a time slot to the given node. It also selects a time slot for itself to forward the packets coming from the given node. As such all the nodes in the path from the given node to the sink have to select additional time-slots. Every node informs its one and two hop neighbors about newly selected slots. This is required to form collision-free schedule.

2.3 Scheduling and Tree Formation in Multiple Sinks Networks

When large number of nodes are present in the network, it is desirable to deploy more than one sinks. If only one sink is present, all the nodes would send packets to that sink only. In other words, all the nodes would join the same tree i.e. rooted at the only sink.

As more and more nodes are deployed in the network, diameter of the tree increases. In other words, tree height increases. So distance of farthest nodes from the sink also increases. Thus average latency faced by packets also goes up. Moreover, as number of nodes increases and tree height increases, number of slots used to schedule the tree increases. When raw convergecast is used, every node forwards all the packets coming from the children. So, increase in tree size results in more workload on nodes one hop away from sink. These nodes have to forward maximum number of packets. They are likely to quickly loose energy and die. This phenomenon is termed as funneling effect ([1]).

When aggregated convergecast is used, every node aggregates all incoming packets with its own packet. So, only one packet comes out of every node. Increase in tree size does not put extra burden on one hop neighbors of the sink. But still schedule length and packet latency increase.

If multiple sinks are deployed, workload can be distributed between sinks. If two sinks are deployed, some nodes would join the tree rooted at first sink and the others may join the tree rooted at second sink. As a result, size of an individual tree is reduced. Thus time required to deliver packets to root is also reduced. Other advantage of multiple sinks is fault tolerance. If one sinks fails, nodes may start sending data to different sink.

It is desirable that workload of sinks (i.e. trees) remain balanced. In other words, equal number of packets should pass through the trees rooted at sinks. If more packets pass through a tree, energy of nodes in that tree would be consumed more. The nodes may quickly die. Due to funneling effect [1], direct neighbors of overloaded sink may die and rest all nodes of that tree may be disconnected from sink. Also schedule lengths of different trees may not be balanced if trees are of different size.

In following paragraphs, some papers addressing techniques to avoid funneling effect are discussed.

An algorithm for load balancing for target tracking application is proposed in [32]. It uses fuzzy logic for load balancing. Whenever a target is detected, sensors send video of the target towards sink. As multiple sinks are present, sensor sends the packet to the sink towards whom traffic is the least.

In [32], fuzzy logic-based load balanced routing is proposed for target tracking in video sensor networks. Whenever a sensor detects a target, sink selection is done to send corresponding video packets. The selection depends on traffic density in the direction of sink.

In [33], it is proposed that for each packet sender node should find forward factor for each of the neighbors. The forward factor of a node is defined as the ratio of residual energy and distance from sink. The sender node forwards its packet to the neighbor with the highest forward factor. As residual energy keeps changing, different packets are likely to be sent through different nodes. The nodes with very less energy are not likely to be selected as forwarders. This method indirectly distributes load across sinks as different neighbors are likely to be connected to different sinks.

In [34], Load Balanced Routing (LBR) is proposed. For each sink, every node finds the ratio r of number of neighbor nodes of the sink and distance in hop count from given node to the sink. The given node sends the packet towards the sink with the highest ratio r. The given node may have multiple neighbors through which the selected sink can be reached. For each packet different neighbor is selected probabilistically.

In [35] and [36], it is proposed to change the path towards sink when energy of nodes in the current path goes beyond specific threshold. In [37], the concept of electrical potential field is used to perform load balancing. When a sink finds itself overloaded (i.e. receives too many packets), it informs the nodes in its tree to transmit data to some other sink.

In [38], SMTLB (Spanning Multi Tree Load Balanced routing) algorithm is proposed. The aim of the algorithm is to balance the workload across the subtrees of the given tree. Each one hop neighbor of a sink becomes root of a subtree. The tree is formed in top-down fashion. Different nodes may generate packets at different rate. Nodes are gradually added in the subtrees such that total number of packets passing through the subtrees remain almost same.

In [39],tree formation and scheduling are considered as two different problems. It is assumed that more than one sinks are present. Two different methods of tree formation are proposed: (i) In the first method, concept of voronoi diagrams is used. Every sink becomes root of exactly one voronoi region. In a given voronoi region, exactly one tree is present. (ii) In the second method, every node is connected to the tree rooted at the sink at smallest hop distance from the given node.

Discussion

In aggregated convergecast, funneling effect [1] is not likely to happen because every node sends out only one packet. So every direct neighbor of sink also sends out one packet. Following paragraphs illustrate issues in aggregated convergecast when multiple sinks are deployed.

If nodes are uniformly deployed, every node joining the nearest sink would automatically result in schedule length balancing. Some times nodes may not be uniformly deployed. More number of nodes may be deployed in an area where finer observations are needed. But in other areas where precise readings are not needed, node deployment may be sparse. Thus some trees would be dense and others would be sparse. The dense trees would need larger schedule than the sparse ones. For example, two trees T_1 and

T_2 are formed with schedule lengths SH_1 and SH_2 respectively. Every node in T_i will get a chance to transmit its packet after every SH_i slots. If schedules are not balanced, nodes in one tree would wait for long time to get their turn to transmit. Thus packet latency would be high. On the other hand, in a tree with small schedule length, latency will be low. If schedule lengths are balanced, nodes of both the trees would suffer equal packet latency. Overall schedule length (SH) of the network would be max(SH_1, SH_2). Thus balancing the schedule lengths of individual trees would also reduce overall schedule length.

The other cause of difference in schedule lengths of trees is different levels of heterogeneity in different regions of the network. For example, in one region, two types of nodes are present. In the other region, six types of nodes are present. The region with two types of nodes is likely to result in better aggregation compared to the other region. As a result, the tree passing through the region with two types of nodes has smaller schedule length than the other tree passing through region having six types of nodes.

To the best of our knowledge, none of the papers available in literature focus on schedule length balancing for aggregated convergecast in multi-sink sensor networks. Most of the papers in area of load balancing try to reduce funneling effect [1] or distribute workload across one hop nodes of sender. In addition, scheduling is not attempted by most of the papers. There are two references close to our work. They are [38] and [39]. In [38], aim is to balance load across sub-trees. When nodes are not distributed uniformly, load may be balanced across sub-trees present in dense region. These sub-trees may be part of a single tree. When tree present in dense region is scheduled, its schedule length is likely to be more than that of tree formed from nodes present in sparse region. Thus this work is not likely to result in schedule length balancing of trees. In addition, it is a centralized algorithm and does not attempt scheduling. As mentioned earlier, centralized algorithms are less desirable compared to distributed algorithms. In [39], it is proposed to divide the entire region into voronoi sub-regions with respect to sinks. When nodes are not distributed in uniform manner, some voronoi regions have more number of nodes and others have less number of nodes. Thus the resulting trees and corresponding schedules are likely to be unbalanced.

We plan to design a distributed algorithm to balance schedule lengths of trees in multi-sink sensor networks. The reason of imbalance in schedule length can either be uneven distribution of nodes or difference in heterogeneity between different regions of the network.

2.4 Research Gap

From previous discussions it is found that following are two research gaps:

1. Present aggregation convergecast scheduling algorithms assume presence of homogeneous network. Existing algorithms should be modified to take heterogeneity into account with the objective of maximizing aggregation.

2. Many algorithms for load balancing in multiple sinks sensor networks are proposed. But most of them try to eliminate funneling effect. There is no algorithm present in literature addressing balancing of schedule lengths of trees rooted at different sinks.

Our focus is to design a distributed algorithm which runs in multiple sinks networks with non-uniform node distribution and heterogeneous nodes. The algorithm should help every node to take a decision about which sink (i.e. tree) to join. The objective of the algorithm is that schedule lengths of trees should be balanced and overall schedule length should be reduced.

2.5 Summary

In this chapter, literature review is presented. Scheduling algorithms are categorized based on type of convergecast addressed. Algorithms designed specifically for multiple sinks networks are also explained. As fault tolerance is an important aspect of tree formation and scheduling, important methods for the same are also discussed. Based on reviewed literature, some open issues are identified. They are presented in section on Research Gap.

Chapter 3

Problem Definition

Multiple trees are formed in multi-sink networks. When node distribution is not uniform, schedule lengths of trees are different. Some method is needed to balance the schedule lengths of trees. This in turn should minimize the overall schedule length. The same issue is taken as the problem to be solved for PhD work. In this chapter, problem is formally defined along with motivation, assumptions, objectives and description of the problem. To keep the chapter complete, explanation given in earlier chapters is repeated at some places in current chapter.

3.1 Motivation

As explained earlier, networks are not always homogeneous. Many applications require heterogeneous networks. For example, sensors are deployed on a bridge to monitor several parameters like vibration, tilt, cracks, shocks and deformation etc.. A single type of sensor may not be able to sense all the parameters. So different types of sensors need to be deployed.

Parent selection in heterogeneous networks should be different. A node may have number of candidate parents. But it should send the packet to the node where packet can be aggregated. For example, a temperature sensor should select a temperature sensor as parent so that its temperature reading can be aggregated with temperature reading of parent. As a result, only one packet would come out of parent. But if pressure sensor is selected as parent, pressure and temperature readings can not be aggregated. So, parent sensor has to send out two packets: (i) The first packet would have pressure reading (ii)

The second packet would have temperature reading received from child.

In a single sink network, the size of the tree will grow as the number of nodes increases. All the nodes would join the same tree. Only one sink will receive packets from all the nodes. This will also increase workload on neighbors of sink. Funneling effect [1] may take place. In addition, schedule length will also increase as the size of tree increases because schedule length depends on density and height of the tree.

Often it is desirable to deploy multiple sinks in the network. The benefit is that multiple trees would be formed. Unlike single sink networks, all the nodes need not join the same tree. As a result, workload would be distributed across the sinks. At the same time, schedule length of every tree will remain reasonable because size of the tree would be reduced.

When multiple trees are formed, a node can join either only one tree or more than one trees. Joining multiple trees means having multiple parents and multiple transmission slots. In short, multiple pairs of slots and parents are selected. If a node is member of a single tree, it requires only one pair of slot and parent. Making a node member of more than one trees requires more work at the time of scheduling because multiple slot/parent pairs are assigned. Also the schedule length will increase because more slots are consumed. Thus packets will face more latency. Assigning multiple parents is really useful in case of parent failure because node can automatically switch to new parent for data transfer. But in distributed algorithms, repairing is done locally. Whenever parent dies, node quickly selects new parent on its own. In addition, schedule length remains reasonable if node joins only one tree because extra slots are not assigned. So it is preferable that every node should join exactly one tree.

Node distribution across the entire network may not be uniform. This means that node density is different in different regions of the network. Density is very high in some regions, whereas very low in other regions. In practice when finer observation is required at some places, more sensors are deployed. But at other places less number of sensors are deployed. This results in unequal density.

When multiple sinks are deployed, non-uniform node distribution may result in unbalanced schedule lengths of trees. The tree spanning dense region of network will result in high schedule length whereas the tree spanning sparse region will result in low schedule length. If schedule length of a tree is SH, it means every node will get its turn to transmit after SH time-slots. So larger schedule length means nodes have to wait longer to transmit. Packets need to be buffered for longer time at the sending node. This in turn would increase latency.

In heterogeneous networks, all types of sensors may not be present everywhere. For example, in some region only two types of sensors need to be present. In the other region of the same network, six types of sensors are required. In a region of two types of sensors, aggregation would be better compared to the other region. As a result, less number of packets are transmitted in the region having two types of sensors compared to the region having six types of sensors. Thus tree spanning the region with two types of sensors is likely to result in smaller schedule length than the tree spanning through region of six types of sensors.

Thus in a multi-sink network, the schedule lengths of trees may not always be balanced. The reason is either non-uniform node distribution or different levels of heterogeneity across the trees. It is desirable that schedule lengths are balanced. If schedule lengths are not balanced, nodes of one tree will suffer high latency and the others will have low latency.

Our aim is to design a distributed schedule length balancing algorithm for multi-sink heterogeneous networks.

3.2 Problem Statement

To design a distributed algorithm to balance schedule lengths of sink-rooted trees in multi-sink heterogeneous sensor networks.

3.3 Assumptions

1. One or more sinks are deployed.
2. More than one type of nodes may be deployed in the network.
3. Node distribution is not uniform.
4. Every sink is root of exactly one tree.

5. Every node joins exactly one tree. In other words, trees are disjoint.

6. Every packet requires one time-slot. Every node is sensing the environment for some time and sending its reading as part of a packet. So time-slot size is such that entire packet is sent. Our work is extensible for the case when a node requires multiple time-slots to send a single packet.

3.4 Objectives

Let us assume that total number of sensor nodes in n and number of sinks is N. So N trees are formed : T_1,T_2,....,T_N. Their schedule lengths are denoted as SH_1,SH_2,....,SH_N respectively. The objectives of proposed solution are as follows:

1. Overall schedule length must be minimized. Overall schedule length SH of the network is defined as follows:

$$SH = max(SH_i), i = 1, 2, ..., N \tag{3.1}$$

2. Difference between schedule lengths of trees should be minimal. It is denoted as SH_{diff}. It is defined as follows:

$$SH_{diff} = \frac{max(SH_i) - min(SH_i)}{max(SH_i)} * 100\%, i = 1, 2, ..., N \tag{3.2}$$

   This would balance the schedule lengths of trees. As a result, TDMA cycle would be completed almost at equal time in all the trees. Finally overall schedule length would be minimized.

3. Minimize Control Overhead (CO). Control overhead at a node i is count of control packets sent by that node. The control messages are required for achieving schedule length balancing and actual scheduling & tree formation. For longer lifetime of sensor nodes, energy consumption due to control messages should be kept minimal.

   Let us denote control overhead at node i by CO_i. Total control overhead CO is defined as follows:

$$CO = \sum_{i=1}^{n} CO_i, i = 1, 2, 3, ..., n \tag{3.3}$$

4. Minimize Average Energy Consumption during Control Phase (E^C). The control phase is the time duration from network setup to the end of scheduling and parent selection algorithm. Energy consumption during control interval is directly proportional to control overhead.

   Let initial energy of every node be E^{init}. The residual energy in node i at the end of control phase be denoted as E_i^{cres}. Energy consumption during control phase at node i be denoted as E_i^C. It is defined as follows:

$$E_i^C = E^{init} - E_i^{cres}, i = 1, 2, 3, ..., n \tag{3.4}$$

   Average energy consumption (E^C) is defined as follows:

$$E^C = \frac{\sum_{i=1}^{n} E_i^C}{n}, i = 1, 2, 3, ..., n \tag{3.5}$$

5. Minimize Average Energy Consumption during Data Phase (E^D). The data phase is the time duration during which nodes send data to sink. Data Phase begins after end of Control Phase. Energy consumption during data phase is proportional to number of data packets sent and received by node.

   The residual energy in node i at the end of data phase be denoted as E_i^{dres}. Energy consumption during data phase at node i be denoted as E_i^D. It is defined as follows:

$$E_i^D = E_i^{cres} - E_i^{dres}, i = 1, 2, 3, ..., n \tag{3.6}$$

   Average energy consumption (E^D) is defined as follows:

$$E^D = \frac{\sum_{i=1}^{n} E_i^D}{n}, i = 1, 2, 3, ..., n \tag{3.7}$$

3.5 Description

To solve above problem, we have attempted following two sub-problems. Their solutions are combined to solve entire problem.

1. Joint scheduling & tree formation in a single sink heterogeneous network.

2. Schedule length balancing in multi-sink homogeneous network.

At first, we attempted joint scheduling & tree formation for a single link network with heterogeneous nodes. We have suggested a distributed scheduling & tree formation algorithm for single sink network with heterogeneous nodes. It is named as AAJST (Attribute Aware Joint Scheduling & Tree formation). The objective is to minimize schedule length by maximizing aggregation. The details are given in Chapter 4.

Then second sub-problem is attempted. It is assumed that multiple sinks are deployed and node distribution is not uniform across the entire region. We have suggested a distributed algorithm which allows every node to decide which sink (i.e. tree) to join such that when actual scheduling and tree formation takes place, the schedule lengths are balanced. The algorithm is named as SLBMHM (Schedule Length Balancing for Multi-sink HoMogeneous networks). The details are given in Chapter 5.

Lastly ideas of scheduling in single sink heterogeneous networks and schedule length balancing in multi-sink homogeneous networks are combined. The resulting algorithm is capable to achieve schedule length balancing in multi-sink heterogeneous networks and also maximize aggregation. The algorithm is named as SLBMHT (Schedule Length Balancing for Multi-sink HeTerogeneous networks). The details are given in Chapter 6.

3.6 Summary

In this chapter, main problem is formally defined. Assumptions and Objectives of proposed solution are explained. Description explaining break-up of main problem into sub-problems is also presented. The Scheduling & Tree formation for heterogeneous networks is explained in the next chapter.

Chapter 4

Attribute Aware Joint Scheduling and Tree formation (AAJST) Algorithm

The first sub-problem of our main problem is scheduling & tree formation in single-sink heterogeneous networks. We have suggested an algorithm named as AAJST (Attribute Aware Joist Scheduling and Tree Formation) for slot and parent selection in case of heterogeneous networks. In heterogeneous networks, different types of packets are received by a node. Our approach is to select different parent for every outgoing packet based on the type of the packet. This chapter presents design of proposed algorithm along with simulation results.

4.1 Motivation

Sensor networks can be homogeneous or heterogeneous. When application requirement is such that only a single physical quantity is to be measured, network is homogeneous. For example, temperature of some place is to be observed. Hence temperature sensors are deployed in that place. But often in many applications, multiple physical quantities are to be observed. For example, on a bridge sensors are often deployed to observe pressure, tilt, shocks, deformation and other things. In this case, sensors of different types need to be deployed. Nowadays single device is capable of sensing multiple quantities. But still all quantities can not be sensed by a single device. So heterogeneous network is still a valid assumption.

Scheduling and Tree formation in heterogeneous network is different than in homogeneous network. In Figure 4.1, homogeneous network and heterogeneous networks are shown. Nodes with labels “T” are temperature sensors and label “P” are pressure sensors.

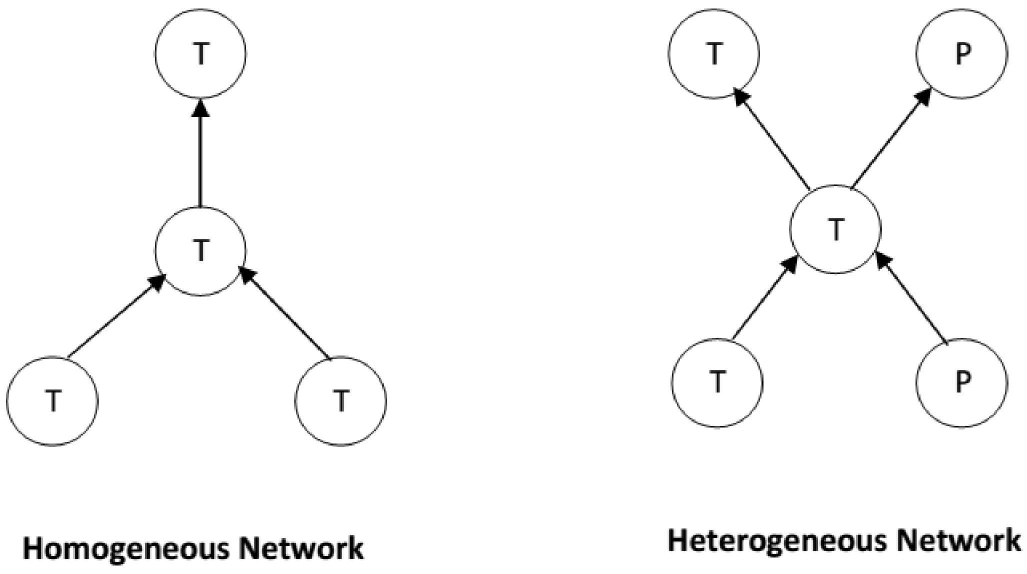


Figure 4.1: Illustration of Difference between Homogeneous and Heterogeneous Networks

In every tree-based network, non-leaf nodes receive one or more packets from children. If network is homogeneous, every node can aggregate all incoming packets with its own packet. As a result, only one packet goes out of the node. In the figure shown above, homogeneous network has all temperature sensors. Every node sends and receives packets of type temperature. At a given node, all temperature packets may be aggregated. Aggregation function may be sum, average, median etc.. Thus only one packet goes out. As a result, every node has to select one parent and one time slot.

If network is heterogeneous, different types of nodes are present in the network. As shown in Figure 4.1, heterogeneous network has temperature and pressure sensors present. It is possible that children of given node are of different types. Thus a node may receive different types of packets. In Figure 4.1, the temperature sensor in the middle has two children: one temperature sensor and one pressure sensor. It receives two packets: one temperature packet and one pressure packet. Temperature packet can be aggregated at temperature sensor. But pressure packet cannot. Thus temperature sensor has to send out two packets: one temperature packet (result of aggregation of two temperature packets) and one pressure packet (received from pressure sensor). To fix the length of slot, we assume that every slot carries one packet. Thus node will require two time-slots to transmit these two packets.

Temperature node can send both the packets to a single parent. If both packets are sent to temperature sensor, temperature packet is aggregated but pressure packet is not. If both the packets are sent to pressure sensor, situation is reverse. One alternate is to send each packets to a different parent, as shown in Figure 4.1. Temperature packet should be sent via temperature sensor and pressure packet should be sent via pressure packet. Thus every packet is aggregated at parent node. If aggregation is improved, average number of packets coming out per node decreases. This results in reduction in number of time slots required and also energy consumption.

Thus it is observed that parent selection in heterogeneous networks is different than homogeneous networks. Nodes need to select multiple parents. Parent selection should be such that packet gets aggregated as early as possible. If aggregation is improved, nodes have to transmit less number of packets. As a result, count of total transmission slots is reduced. Finally schedule length is also reduced. In addition, energy consumption is also reduced. At every node, energy is consumed due to following two reasons: (i) Control packets exchanged with neighbors during slot and parent selection. (ii) Transmission of data packets. Proper parent selection reduces energy consumption during both control phase and data transmission phase. As node has to select less number of slots, less control messages are required. As less number of data packets are to be sent, energy consumption during data phase is reduced.

As explained in Chapter 2, slot and parent selection (i.e. tree formation) should take place jointly. If tree is formed first and then slot assignment is done, tree structure controls the performance of scheduling algorithm. But if schedule and tree are formed together, scheduling algorithm is not dependent on structure of tree. In other words, every node may check if there is any suitable parent in the lowest available time-slot. In stead of selecting parent first and then finding time-slot, it is better to select slot and parent together such that node can transmit in the lowest possible slot. This approach reduces schedule length of the tree.

Joint approach is applicable to heterogeneous networks also. In homogeneous networks, every node finds single pair of slot and parent whereas multiple slot/parent pairs are required for every node in heterogeneous networks.

Problem of scheduling & parent selection in heterogeneous networks is formally defined in next sub-section. Subsequent sub-sections present idea of the proposed algorithm, formal description of the proposed algorithm, correctness proofs and simulation results.

4.2 Problem Statement

To design a distributed parent selection algorithm for single sink heterogeneous wireless sensor networks.

4.2.1 Assumptions

1. Single sink is deployed.
2. Network is heterogeneous.
3. Every packet requires one time slot.
4. Aggregation convergecast is used. Two or more packets of same type can be fully aggregated into one packet. Different packets of different types can not be aggregated. They are transmitted individually.

4.2.2 Objectives

Let us denote total number of nodes in the network by n.

1. Minimize Schedule Length (SH). Schedule length is count of unique slots required to schedule entire tree. In other words, it is the highest slot number.
2. Minimize Average Aggregation Factor. Let the no. of packets received for forwarding by node i be R_i and no. of packets forwarded by node i be F_i. Aggregation factor η_i at a node i is defined as follows:

$$\eta_i = \frac{R_i - F_i}{R_i} \tag{4.1}$$

Aggregation factor at a node i represents the fraction of received packets aggregated at that node. The higher value of n_i is considered better. Average aggregation factor η is defined as follows:

$$\eta = \frac{\sum_{i=1}^{n} \eta_i}{n}, i = 1, 2, 3, ..., n \tag{4.2}$$

3. Minimize Control Overhead. Control Overhead (CO) is defined in equation 3.3.
4. Minimize Average Energy Consumption during Control Phase (E^C). It is defined in equation 3.5.
5. Minimize Average Energy Consumption during Data Phase (E^D). It is defined in equation 3.7.

4.3 Attribute Aware Joint Scheduling and Tree formation (AAJST) for Single Sink Heterogeneous Networks

4.3.1 AAJST Algorithm

As mentioned earlier, a joint distributed scheduling and tree formation for homogeneous networks is presented in DICA[21]. It is modified to work with heterogeneous networks. New proposed protocol is named as Attribute Aware Joint Scheduling & Tree formation (AAJST).

When network is heterogeneous, different types of nodes are present in the network. As mentioned earlier, bottom-up scheduling and parent selection is desirable to maintain aggregation freshness in aggregated convergecast. When a node attempts to decide its slot and parent, it has following pieces of information: (i) number of incoming packets (ii) type of each incoming packet. Of course, node also knows the type of packet generated by itself. Based on these, node can identify the followings: (i) number of outgoing packets (ii) type of each outgoing packet. For each outgoing packet, node may select a different parent such that packet would be aggregated as soon as possible. Parent selection based on count of unscheduled neighbors (as proposed in DICA[21]) is not a suitable approach

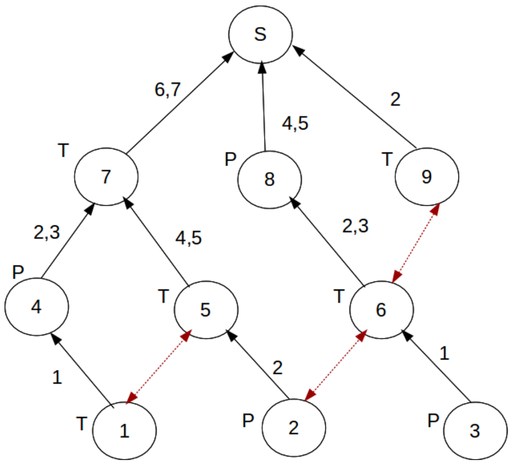


Figure 4.2: Illustration of Scheduling and Tree Formation using DICA_EXTENSION

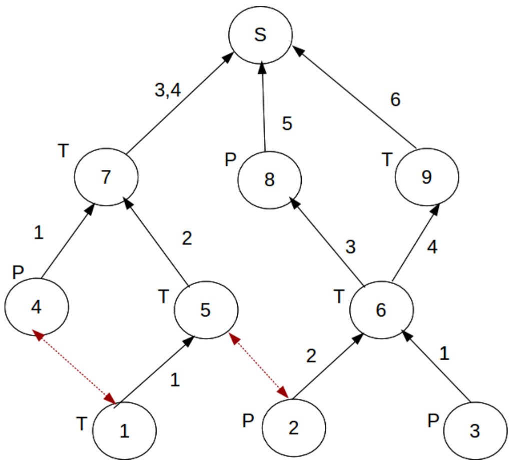


Figure 4.3: Illustration of Scheduling and Tree Formation using AAJST

for heterogeneous networks. Following paragraphs explain the core idea behind parent selection in AAJST.

In Figures 4.2 and 4.3, same network is shown. The network is heterogeneous. Two types of sensors i.e. temperature and pressure sensors are present. Continuous line from one node to the other indicates that former has selected later as parent. Dashed line (in red color) between two nodes indicate that the nodes are in radio range of each other. Figure 4.2 illustrates scheduling and parent selection without considering node heterogeneity. DICA [21] is used for slot/parent selection. Only difference is that now multiple slots/parent pairs are selected. We call this approach DICA_EXTENSION. Figure 4.3 illustrates scheduling and parent selection as done in AAJST.

To maintain aggregation freshness, scheduling would take place from bottom to top (i.e. from leaf nodes to sink). The first step is Leveling. It is present in both AAJST and DICA_EXTENSION. Leveling is done by flooding HELLO messages in the network by sink. Every HELLO message contains a field 'counter'. Sink sets counter to 0 and broadcasts the message. Every node receiving HELLO packet increments the counter and broadcasts the message. The value of counter indicates level of the node broadcasting HELLO message Once leveling is done, scheduling and parent selection algorithm runs. Next few paragraphs explain slot and parent selection as done in DICA_EXTENSION.

Slot/Parent selection begins from leaf nodes. Nodes 1,2 and 3 are leaf nodes. As shown in Figure 4.2, node 1 is a temperature sensor. Node 2 and Node 3 are pressure sensors. Node 1 would try to select slot 1 for transmission. Nodes 4 and 5 are in radio range of 1. So it would select one of them as parent. Node 4 has less number of unscheduled neighbors than node 5. The unscheduled neighbors of node 4 are 1 and 7. But the unscheduled neighbors of node 5 are 1, 2 and 7. Thus node 1 would select node 4 as parent.

It is possible that node 2 would start slot selection with node 1. It would select slot 1 only. Nodes 5 and 6 are in its radio range. Node 5 has three unscheduled neighbors (nodes 1,2 and 7). Whereas node 6 has four unscheduled neighbors (2,3,8,9). Node 2 will select node 5 as parent because it has the least number of unscheduled neighbors. The REQUEST packet broadcasted by node 2 would be heard by node 5 and 6. As node 5 has already received REQUEST from 1, it would send negative reply to 2. The reason is that parent of node 1 (i.e. node 4) has less number of unscheduled neighbors than parent of node 2 (i.e. node 5). Thus node 1 has better parent. It gets preference over node 2. So, node 2 would wait until node 1 completes the process of slot and parent selection. It selects slot 2 to transmit to parent node 5.

Node 3 can perform slot and parent selection in parallel with node 1. Node 3 has no choice but to select node 6 as parent. Because it is the only one in its radio range.

Node 4 is a pressure sensor. It receives temperature packet from node 1. Node 4 selects two transmission slots i.e 2 and 3. In one slot it would send temperature packet. In the other slot, it would send pressure packet. Node 4 has no choice except to select node 7 as parent. Node 5 also selects node 7 as parent because it has no other choice. It has to send out two packets, one pressure and the other temperature. It selects two transmission slots i.e. 4 and 5. Node 6 selects node 8 as parent. As node 6 has to forward two packets i.e. pressure and temperature, it selects two slots (2 and 3). Node 8 and 9 have equal number of unscheduled neighbors i.e. node 6 only. Node 8 is selected (by node 6) as its ID is smaller.

Finally nodes 7, 8 and 9 are scheduled. Node 7 selects slots 6 and 7. Node 8 selects slots 4 and 5. Node 9 selects slot 2. Node 9 has no child but selecting slot 1 would create interference at node 6. So it could not transmit in slot 1. The largest slot in entire schedule is 7. So schedule length is 7. Next few paragraphs explain slot and parent selection as done in AAJST.

In AAJST also, scheduling and tree formation is bottom-up. First leaf nodes are scheduled. Nodes 1, 2 and 3 are leaf nodes. They can start the process in parallel. Node 1 selects slot 1. As node 1 generates temperature packet, it selects node 5 as parent so that its packet could be aggregated with temperature reading generated by node 1. Eventhough node 5 has more number of unscheduled neighbors than node 4, priority is given to node 5.

Node 3 has no choice but to select node 6 as parent. It also selects slot 1. Node 2 selects node 6 as parent eventhough it is a temperature sensor. The other choice is node 5 which is also temperature sensor. Pressure packet sent by node 2 would be aggregated at node 6 with pressure packet sent by node 3. As shown in Figure 4.2, selecting node 5 as parent would require node 5 to use two slots.

Node 4 has no child. So it selects slot 1. It selects node 7 as parent. Node 5 is a temperature sensor. It selects node 7 as parent. Node 5 has no other choice. Node 6 has two pressure packets coming in and itself generates temperature packet. For pressure packet, it selects node 8 as parent because node 8 is a pressure sensor. Node 9 is selected as parent for temperature packet.

Finally, nodes 7, 8 and 9 are scheduled. Node 7 selects slots 3 and 4. Node 8 and 9 select slot 5 and 6 respectively. The highest slot in entire schedule is 6. Thus schedule length is 6.

From Figures 4.2 and 4.3, it is seen that AAJST is likely to result in lower schedule length compared to DICA_EXTENSION . The core idea of AAJST is to select parent considering the type of packet to be forwarded. In contrast, DICA_EXTENSION selects the node with the least number of unscheduled neighbors as parent. It does not consider the type of packet and type of parent.

Parameter	Meaning
P_u	Parent of node u
TS_u	Transmission slot of node u
TM_u	Highest transmission slot of children of node u
OH_u	Neighbors of node u overhearing in TS_u
RV_u	Neighbors of node u receiving in TS_u
TR_u	Neighbors of node u transmitting in TS_u
$candp_u[TS_u]$	Neighbors of node u which do not transmit, receive or overhear in slot TS_u
REQUEST	The message broadcast by a node to inform its choice of slot and parent to its candidate parents
REPLY	Response generated by candidate parents upon receiving REQUEST message
SCHEDULE	The message broadcast by a node to confirm its choice of slot and parent to its candidate parents
FORBIDDEN	Message broadcast by a candidate parent upon reception of SCHEDULE message

Table 4.1: Notations used in AAJST Algorithm (taken from [21])

In summary, every node should perform following steps to find a parent to forward a packet of type t in given time slot:

1. Check if there is any neighbor of type t in neighborhood. If so, packet should be sent to that neighbor. If no such node is found, step 2 should be executed.

2. Check if there is any node in neighborhood which is receiving packets of type t from other nodes. If any such node is found, it should be considered as parent for packet of type t. If no such node is found, step 3 should be executed.

3. Check if there is any node in neighborhood which has one or more nodes of type t in its neighborhood. If any such node is found, it should be considered as parent for packet of type t. If no such node is found, step 4 should be followed.

4. Select as parent the neighbor node with minimum number of unscheduled neighbors.

The AAJST algorithm is formally written as Algorithm 1. Its explanation is given below. The required notations are summarized in Table 4.1.

The AAJST algorithm can be explained as follows. For every outgoing packet, node executes AAJST algorithm. Input to the algorithm is message m. The output of the algorithm is corresponding parent and slot to transmit that message. Initially, node is

Algorithm 1: Attribute Aware Joint Scheduling and Tree Formation

```
procedure NOT_READY
    $TS_u = 0$
    Initialize $OH_u$, $RV_u$,$TR_u$ and $candp_u[TS_u]$ to $\emptyset$
    Wait until all neighbors at level $(l + 1)$ are scheduled
    Call $READY()$
end procedure
procedure READY
    $TS_u = TM_u + 1$
    Increment $TS_u$ until $candp_u[TS_u]$ has at least one member
    $PARENT\_SELECTION()$
    Broadcast REQUEST message to all nodes in $candp_u[TS_u]$
    Wait until REPLY messages come from all nodes in $candp_u[TS_u]$
    if any one REPLY is negative then
        Call $WAIT\_FOR\_SELECTION$
    else
        Broadcast SCHEDULE message to confirm $TS_u$ and $P_u$
end procedure
procedure PARENT_SELECTION( )
    m = message to be scheduled
    for each node n in $candp_u[TS_u]$ do
        if node type is same as message type then
            $P_u = n$
            return
    max=0
    for each node n in $candp_u[TS_u]$ do
        if $max <$ no. of incoming packets of type $m$ then
            max = no.of incoming packets of type $m$
            $P_u = n$
    if max != 0 then
    return
    for each node n in $candp_u[TS_u]$ do
        if $max <$ No. of neighbors of $n$ of same type as $m$ then
            max = No. of neighbors of n of same type as $m$
            $P_u = n$
    if max != 0 then
        return
    else
        $P_u$ = node with min. no. of unscheduled neighbors
        if more than one such nodes are present then
            $P_u$ = node with lowest ID
end procedure
procedure WAIT_FOR_SELECTION
    Wait until FORBIDDEN messages are received from conflicting nodes
    Call READY()
end procedure
```

in NOT READY state. It calls function not_ready(). It waits for its neighbors in level $(l+1)$ to get scheduled. Level of given node is l. When all such nodes are scheduled, given node switches to READY state. it calls function ready().

In ready() function, node selects lowest possible transmission slot TS_u as one more than the highest transmission slot of children. Node increments TS_u until TS_u is found such that no neighbors are receiving in TS_u and set of candidate parents is not empty.

As a next step, parent_selection() function is called to select suitable parent. It works as follows: Given node is forwarding a packet of type t. First it checks if there is a node of type t in candidate parent set. If such a node is present, it is selected as parent and function returns. Otherwise, the node receiving maximum number of packets of type t is selected as parent and function returns. If no node in candidate parent set is receiving any packet of type t, following step is executed. Node checks if there is any candidate parent which has any neighbor of type t. If so, such a node is selected as parent. If there are more than one such nodes, the node with maximum number of neighbors of type t is selected as parent. Then function returns. If still suitable parent is not found, the node with the minimum number of unscheduled neighbors is selected as parent and function returns.

Once parent is selected, node broadcasts REQUEST message to candidate parents. If it receives positive response from all candidate parents, it broadcasts SCHEDULE message. If negative response comes from any one candidate parent, node moves to WAIT_FOR_SELECTION state. It calls wait_for_selection() function. When node overhears FORBIDDEN message, it switches to READY state and calls ready() function.

4.3.2 Correctness of Algorithm

Lemma 4.3.1. *AAJST does not result in cycle.*

Proof. In AAJST, tree formation and scheduling proceeds in bottom-up manner. Every node at level l is scheduled only after its neighbors at level $(l+1)$ are scheduled. The node selects as many slot/parent pairs as the number of outgoing packets. Every node selects parent from its neighborhood. Thus a node at level l may select parent from neighbors at level $(l+1)$, l or $(l-1)$.

Assume that every node at level l selects node of level $(l-1)$ as parent. Suppose path is A$-$ >B$-$ >C$-$ >D. That is parent of A is B, parent of B is C and parent of C is D. Thus Level of A $>$ Level of B $>$ Level of C $>$ Level of D. Now D has to select a parent. For cycle to be formed, D should select either C,B or A as parent. But as every node of level l has to select parent from level $(l-1)$, D can not select any of those three nodes as parent. So cycle can not be formed.

There are chances of cycle formation when node is allowed to select a parent from level l or $(l+1)$. Following points are ensured by algorithm: (i) Parent is always assigned higher time slot than highest time slot of children. (ii) Node can not select a node as parent whose time slot is less than or equal to current time slot of node. Consider path A$-$ >B$-$ >C$-$ >D again. So, Time slot of A $<$ Time slot of B $<$ Time Slot of C $<$ Time slot D. Cycle would be formed if D selects C,B or A as parent. But it can not. Because time slot of D would be highest among A,B and C. So it has to select different node as parent. Thus a node may select parent from level l or $(l+1)$. But cycle is prevented by condition that parent's slot should be higher than child's slot.

$\square$

Lemma 4.3.2. *AAJST results in collision free schedule.*

Proof. When a node p wants to select a slot TS_p, it prepares a list of neighbors which do not transmit, receive or overhear in slot TS_p. If any neighbor transmits in slot TS_p, it can not be selected as parent. Because wireless devices are half-duplex. Node can not transmit and receive at the same time. If any neighbor is receiving or overhearing packets from any other node in slot TS_p, transmission from node p will create interference at that neighbor node. Thus initial slot selection is such that collision does not occur.

It may happen that two nodes p and q together begin slot selection. Both of them select TS_p as their transmission slot. Both would send REQUEST message to their candidate parents. If there is a common candidate parent, it would send positive reply to only one of the two. Positive reply is sent to the one whose selected parent has least number of unscheduled neighbors. If the parents have equal number of unscheduled neighbors, number of unscheduled neighbors of sender nodes are checked. The one with smaller number of unscheduled neighbors gets positive reply and the other gets negative reply. If both the senders have equal number of unscheduled neighbors, tie is broken based on ID of sender. The one with smaller ID is preferred. Thus either p or q (but

Scenario	No. of Attributes (n_A)
1	1
2	2
3	4
4	6
5	8

Table 4.2: Different Simulation Scenarios

not both) will get transmission slot TS_p. The node which gets negative reply has to wait until the other node completes slot/parent selection process.

From above two paragraphs, it is clear that no two nodes will select common transmission slot such that it will create collision at candidate parents. Thus collision free schedule is formed.

□

4.4 Simulation Results

We have used Network Simulator 2 (NS-2.35) to test the proposed algorithm. Simulation is designed as follows:

4.4.1 Simulation Design

A square area of 3000 x 3000 meters is considered for node deployment. Nodes are deployed randomly. The area is divided into grid of 20 x 20 points. Distance between every two horizontal or vertical grid points is 15 meters. Nodes are probabilistically deployed at grid points. Probability p_d of node being deployed at a grid point is 0.5. That is, there are 50% chances that a node is present at given grid point.

Network is heterogeneous. As mentioned earlier, type and attribute refer to the same thing. In future discussions, word attribute will be used to refer to type of packet. To test performance of proposed protocol, different scenarios are created as shown in Table 4.2:

Five different scenarios are created. In first scenario, no. of attributes present in the network is 1. That is, all the nodes are of same type i.e. homogeneous network. In second scenario, no. of attributes is 2. It means that two types of nodes are present in

Parameter	Value
Node deployment	Random
Radio Radius	30m
Transmission Power Consumption	0.660 W
Receive Power Consumption	0.395 W
Sleep Power Consumption	0 W
Data Generation Rate	1 packet every 10 seconds
Simulation Time	2500 Seconds

Table 4.3: Simulation Setup

the network. Other scenarios can be understood in a similar way. As seen from Table 4.2, from scenario 1 to 5, no. of attributes are increased. That is, network becomes more and more heterogeneous.

In every scenario, nodes are randomly deployed on grid points as explained earlier. Every node is randomly assigned attribute. No. of attributes is denoted as n_A. All attributes are equally probable. That is, the probability p^{A_i} that node is assigned attribute A_i is as follows:

$$p^{A_i} = \frac{1}{n_A} \tag{4.3}$$

For example, in second scenario, n_A is 4. Four attributes are A_1, A_2,A_3,A_4. So, probability p^{A_i} is 0.25. That is, a node is assigned any one attribute with probability 0.25. As attribute denotes type of node, we can say that A_1 is temperature, A_2 is pressure, A_3 is solar radiation and A_4 is humidity.

Different performance parameters are used to evaluate the performance of proposed protocol. In every graph, X-axis is number of attributes. We are checking performance by varying heterogeneity level of the network.

For every scenario, five different instances are generated randomly. Simulation result for any scenario is an average of results of instances of that scenario. To measure the confidence in the result, along with average, standard deviation is also calculated. The same is plotted in graphs as error bars.

Network has only one sink. It is placed at bottom-right corner of plane.

4.4.2 Simulation Setup

Table 4.3 presents simulation setup. Deployment of nodes is random as mentioned in previous sub-section. Radio radius of nodes is set to 30 meters. Normally sensor nodes have radio radius in the same range. Every node generates one data packet at every 10 seconds. The duration of simulation is 2500 seconds. Initial 2000 seconds are for control phase. During control phase, nodes perform slot and parent selection. Rest 500 seconds represent data phase. Nodes generate and send data packets during data phase through tree formed during control phase.

4.4.3 Performance Parameters

Following performance parameters are studied through simulation. As mentioned earlier, number of nodes is denoted by n.

1. Schedule length (SH).
2. Average Aggregation Factor (η). It is defined in equation 4.2.
3. Control Overhead (CO). It is defined in equation 3.3.
4. Energy Consumption During Control Phase (E^C). It is defined in equation 3.5.
5. Energy Consumption During Data Phase (E^D). It is defined in equation 3.7.

4.4.4 Discussion of Results

Simulation Results are discussed in this subsection.

Aggregation Factor, Schedule length and Energy Consumed during Data Phase

In Figure 4.4, the graphs of Average Aggregation Factor v/s. No. of Attributes are presented. As mentioned earlier, aggregation factor at a node is fraction of received packets aggregated at that node. The higher value of aggregation factor is considered better. It is seen from the graphs that AAJST results in better aggregation than DICA_EXTENSION. In AAJST, for every packet, a parent is selected considering attribute of the packet. This

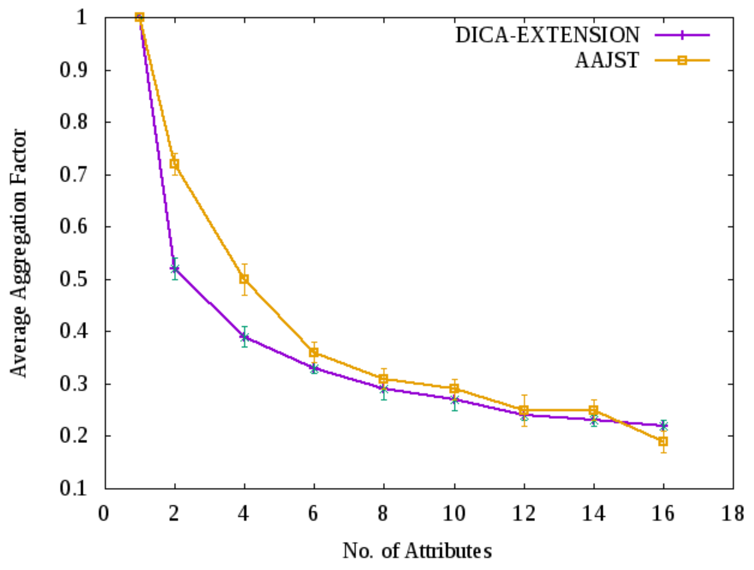


Figure 4.4: Dependency of Aggregation Factor on No. of Attributes

increases the chance that packet would be aggregated earlier in its journey towards sink. Thus every node needs to forward fewer number of packets. As a result, better aggregation factor is achieved.

When network is homogeneous, perfect aggregation occurs in both the algorithms. Every node sends out only one packet, no matter how many packets are received. Thus aggregation factor is 1 (100% aggregation). As number of attributes increases, aggregation factor deteriorates. When more attributes are present, sender of packet may not be able to find suitable next hop such that packet can be aggregated.

It is observed that when two attributes are present, the difference between aggregation factors of AAJST and DICA_EXTENSION is around 20 %. But as heterogeneity increases, the gap between aggregation factors of two methods is reduced. When no. of attributes is 4, the difference is around 10%. At the end, the gap is as small as 3% when no. of attributes is increased to 16. When heterogeneity is high, it is difficult for AAJST also to find a parent where packet could be aggregated. Packet gets aggregated after traveling more number of hops. Thus at high level of heterogeneity, performance of AAJST is close to DICA_EXTENSION.

The graphs for Schedule Length v/s. No. of Attributes are shown in Figure 4.5. It is seen that AAJST results in small schedule length than DICA_EXTENSION. Schedule length is the count of unique slots required to schedule the tree. It is explained earlier

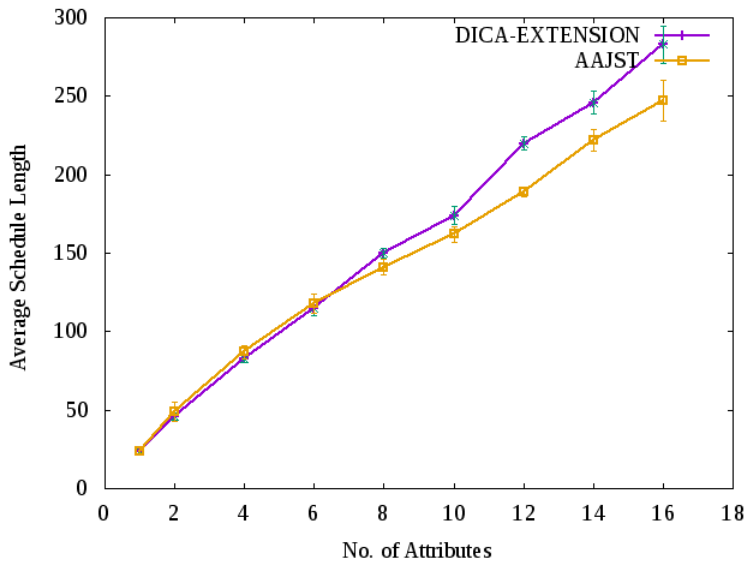


Figure 4.5: Dependency of Schedule Length on No. of Attributes

that AAJST results in better aggregation factor than DICA_EXTENSION. Due to better aggregation, number of packets passing through the tree during a TDMA cycle is reduced. Thus less number of slots are required to schedule the tree.

As it is seen in Figure 4.4 that aggregation factor deteriorates with increase in number of attributes. As schedule length in inversely proportional to aggregation factor, schedule length increases with increase in number of attributes present in the network. This applies to both AAJST and DICA_EXTENSION.

When number of attributes are 2, 4 and 6, schedule lengths in both the algorithms are almost same. But for the remaining cases, AAJST results in smaller schedule length then DICA_EXTENSION. When number attributes are 8 and 10, AAJST gives 5 percent smaller schedule length than DICA_EXTENSION. Gradually the difference increases. For other values the difference is 10 percent or more.

In Figure 4.6, the graphs of Energy Consumption during Data Phase v/s. No. of Attributes are shown. Nodes send and receive data packets during data phase. If less number of packets are sent/received, less energy is consumed. The AAJST algorithm results in better aggregation factor than DICA_EXTENSION. So, nodes need to forward less number of data packets. Thus energy consumed in data phase is less compared to DICA_EXTENSION. As in both the algorithms, aggregation factor deteriorates with increase in number of attributes, energy consumption during data phase increases with

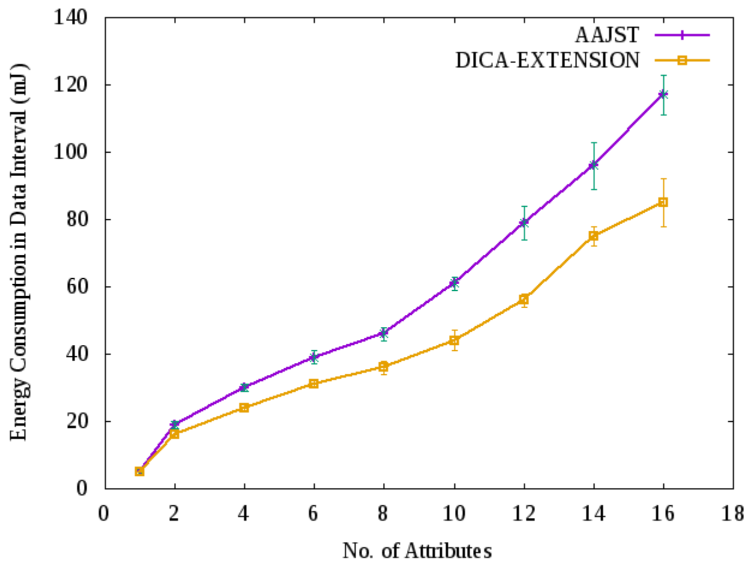


Figure 4.6: Dependency of Energy Consumption during Data Phase on No. of Attributes

increase in number of attributes. The saving in energy consumption is from 15 percent to 30 percent.

Total Number of Required Slots, Control Overhead and Energy Consumption during Control Phase

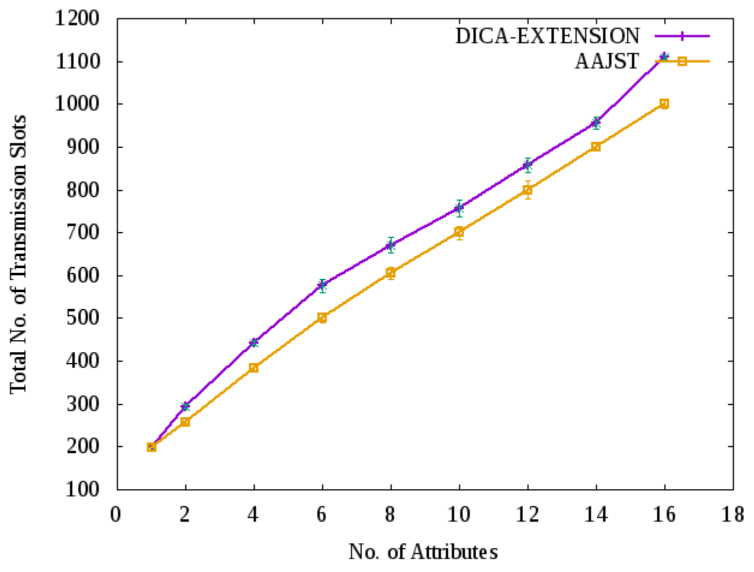


Figure 4.7: Dependency of Total No. of Transmission Slots on No. of Attributes

In Figure 4.7, the graphs of Total Number of Transmission Slots v/s. No. of Attributes are presented. As explained earlier, increase in no. of attributes results in poor aggregation. That is, nodes have to forward more number of packets. It is assumed that every packet requires one transmission slot. As number of packets required to be forwarded increases, number of slots required by a node also increases. Thus the total number of required slots to schedule entire tree increases with no. of attributes. Schedule length is count of unique slots whereas here total of slots required by all the nodes is considered. Due to spatial reuse, schedule length is less than count of total slots.

As explained earlier, AAJST results in better aggregation factor. As less number of packets are forwarded by nodes, number of slots required is also less. So AAJST results in less number of total transmission slots than DICA_EXTENSION. It is seen from the graphs that AAJST results in approximately 10 percent less transmission slots than DICA_EXTENSION.

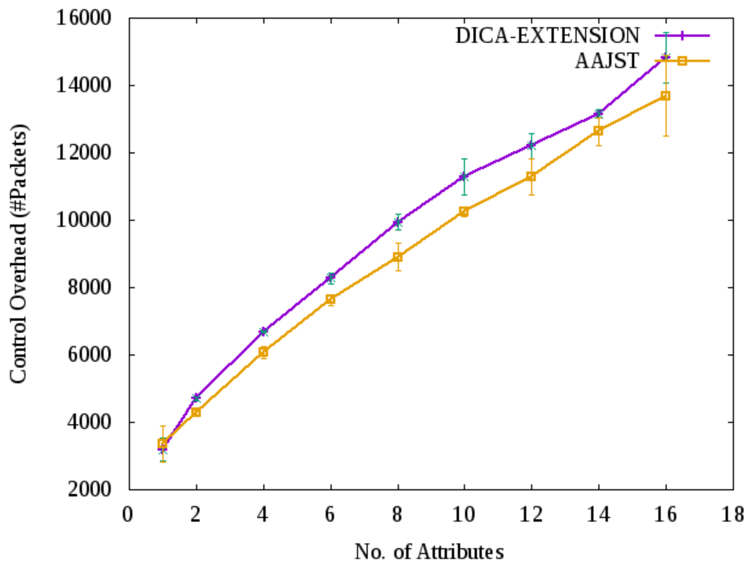


Figure 4.8: Dependency of Control Overhead on No. of Attributes

In Figure 4.8, the graphs of Control Overhead v/s. No. of Attributes are presented. During scheduling & tree formation, every node exchanges some control messages with neighbors so that collision free schedule is formed. These messages constitute control overhead. It is seen from Figure 4.8 that as no. of attributes increases, control overhead also increases. As explained earlier, increase in no. of attributes results increase in total number of required transmission slots. Every node has to select more number of transmission slots. For each slot-selection, a number of control messages are exchanged between given node and its neighbors. Thus control overhead increases with no. of attributes.

As AAJST needs less number of transmission slots, it's control overhead is also less compared to DICA_EXTENSION. It is seen from the graph that AAJST results in 7 percent to 10 percent less control overhead than DICA_EXTENSION.

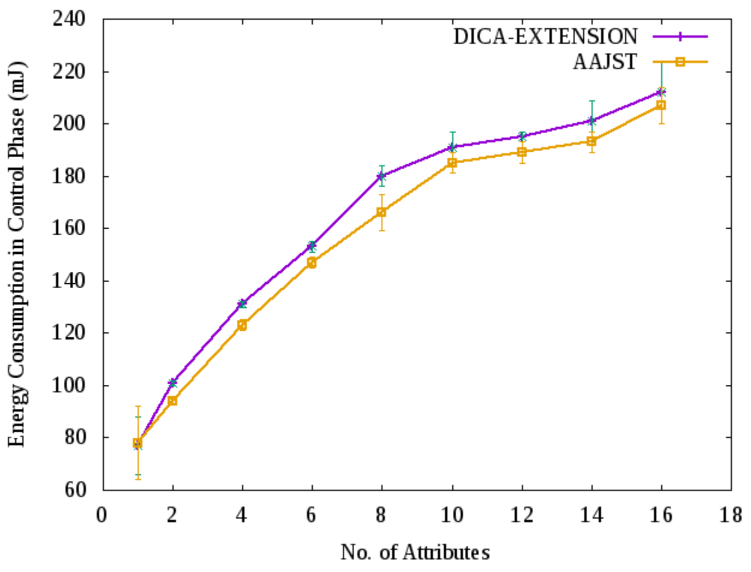


Figure 4.9: Dependency of Energy Consumption during Control Phase on No. of Attributes

Energy consumption during control phase is directly proportional to control overhead. In Figure 4.9, graphs of the same are shown. The nature of graphs is similar to those in Figure 4.8. So, explanation is not repeated. It is seen that AAJST results in approximately 5 percent less energy consumption compared to DICA_EXTENSION.

Thus following conclusion can be derived from graphs shown in Figures 4.4,4.5 and 4.6. In AAJST algorithm, every packet is sent towards a parent such that it can be aggregated as soon as possible. Whereas in DICA-Extension, no such criteria is used. So, AAJST algorithm results in better aggregation compared to DICA-extension. As aggregation factor increases, less packets come out of nodes. Thus energy spent in data transmission is also saved. In addition, as every packet consumes one time-slot, reduction in number of outgoing packets results in reduction in number of slots required to schedule the tree. So, schedule length is also reduced.

4.5 Summary

In this chapter, AAJST (Attribute Aware Joint Scheduling & Tree formation) algorithm for single-sink heterogeneous networks is presented. The idea is to select different parent for every outgoing packet. Parent selection is done based on type of packet. It is observed that AAJST results in better aggregation compared to DICA_EXTENSION. As a result, it results in smaller schedule length, reduction in energy consumption during control phase and data phase.

Chapter 5

Schedule Length Balancing for Multi sink HoMogeneous networks (SLBMHM) Algorithm

When node distribution is not uniform, schedule lengths of trees may be very different. In this chapter, a distributed algorithm is presented which guides every node to join exactly one tree such that schedule lengths of trees remain balanced. The proposed algorithm is named as SLBMHM (Schedule Length Balancing for Multi sink HoMogeneous networks). As a result of balancing, overall schedule length is also minimized. This chapter presents detailed design of proposed algorithm, correctness proofs and simulation results.

5.1 Motivation

Often node deployment is not uniform. Many times application requirement is such that finer observation is required at some places in the region. Thus more sensors need to be deployed at such places. But observation need not be very fine at other places. So sensors may be sparsely deployed. Thus some places in the network have high node density and other places have low density. But every node eventually becomes part of a tree. Thus some trees are very dense and others are very sparse. Schedule length of a dense tree is higher than that of sparse tree. Thus difference in node density results in difference in schedule lengths of trees.

One way to attempt schedule length balancing is as follows: First form the trees and assign slots to the nodes. As a result, exact schedule length of each tree would be available. Now move the nodes from trees with high schedule length to the trees with low schedule length. The movement should be such that the resulting schedule lengths of all the trees remain almost equal.

It is suggested in DICA([21]) that scheduling of an aggregated convergecast tree must should be done in bottom-up fashion. If a node moves from one tree to another, it means that its parent changes. For aggregation freshness, the time slot of parent must be higher than child's slot. If parent's current time slot is not greater than newly arrived child's time slot, parent must be assigned new slot. The process results in a sequence of changes. That is, now slot of parent's parent need to be modified. If a large number of nodes move from one tree to another tree, the other must be rescheduled completely. On the other end, the tree whose nodes leave must also be rescheduled for effective time slot utilization. Otherwise, many slots in the schedule (of the tree) may be wasted.

Thus it seems that if balancing is done after scheduling, scheduling need to be done once again. As distributed scheduling algorithms involve message exchange between every node and its neighbors, scheduling is costly in terms of energy consumption. So rescheduling is not desirable.

So it is better to perform scheduling after assignment of nodes to trees. For this every node has to join a tree such that schedule lengths of trees remain balanced at the end. In this work, we have proposed an algorithm knows as SLBMHM(Schedule Length Balancing for Multi-sink HoMogeneous sensor networks). The algorithm runs before slot & parent selection takes place. The algorithm works in two phases: (i) Schedule Length Estimation (ii) Tree Switching. During schedule length estimation, temporary sink-rooted trees are formed. The schedule length of each tree is estimated by corresponding sink. The schedule lengths are exchanged by the sinks and an average is calculated. In the second phase, nodes from the trees with greater than average schedule length are asked to move to those trees whose schedule length is smaller than average schedule length. Once each and every node decides which tree (i.e. sink) to join, exact slot & parent selection as suggested in DICA([21]) is performed.

5.2 Problem Statement

To design a distributed schedule length balancing algorithm for aggregated convergecast scheduling in multi-sink sensor networks with non-uniform node distribution and homogeneous nodes.

5.2.1 Assumptions

1. One or more sinks are deployed.
2. Network is homogeneous. That is, all the nodes are of the same type.
3. Node distribution is not uniform.
4. Every sink is root of exactly one tree.
5. Every node joins exactly one tree. In other words, trees are disjoint.
6. Every packet requires one time-slot.

Most of the assumptions are discussed in Chapter 3. So detailed explanation is not given here.

5.2.2 Objectives

1. Overall schedule length (SH) must be minimized. It is defined in equation 3.1.
2. Difference between schedule lengths of trees should be minimal. It is denoted as SH_{diff}. It is defined in equation 3.2
3. Control Overhead (CO) should be minimal. It is defined in equation 3.3.
4. Average Energy Consumption during Control Phase should be minimal. It is denoted as E^C. It is defined in 3.5.
5. Minimize Average Energy Consumption during Data Phase. It is denoted as E^D. It is defined in 3.7.

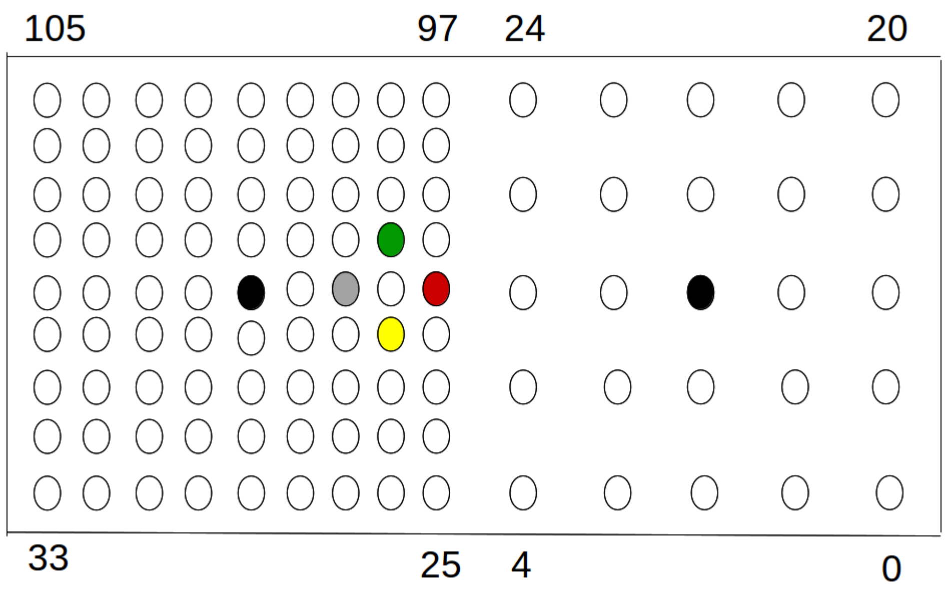


Figure 5.1: Illustration of Node Deployment for SLBMHM Algorithm

5.3 SLBMHM Algorithm

5.3.1 Illustration of the Algorithm with an Example

In Figure 5.1, a sample topology is shown. Nodes 0 to 24 are part of sparse region. Nodes 25 to 105 are in dense region. Assume that there are two sinks: S_1 is node 12 (center node in sparse region), S_2 is node 65 (center node is dense region). At the beginning of algorithm, S_1 and S_2 will turn by turn flood HELLO packet. At the end, every node will know its hop distances from both S_1 and S_2. Every node will select the nearest sink (in terms of hop count) as its home sink. Nodes 0 to 24 will join tree of S_1. Let us call it Group 1. Nodes 25 to 105 will join tree of S_2. Let us call it Group 2.

Every node in Group 1 and Group 2 will select one node as temporary parent. In Figure 5.2, formation of temporary trees is shown. All edges are not shown in figure. Some sample edges are shown to provide better visualization. Now leaf nodes in Group 1 and Group 2 will calculate their neighbor count and height. Neighbor count and height would be sent as part of JOIN messages. Each node in path will wait for its temporary children to send JOIN message. Once a node receives JOIN messages from all its temporary children, it will add neighbor counts received with its own neighbor count. The node will send JOIN message to its parent. It will contain total neighbor count and maximum

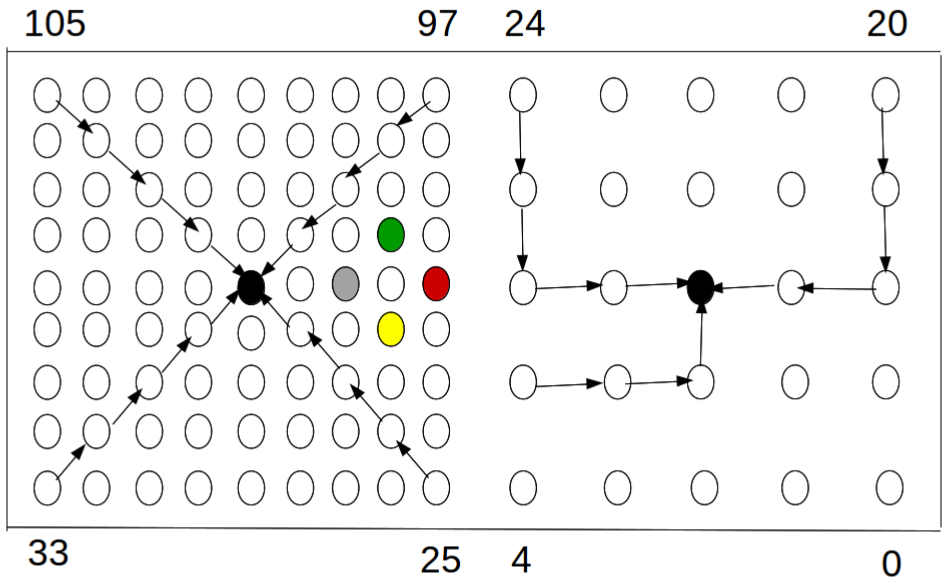


Figure 5.2: Illustration of Formation of Temporary Trees in SLBMHM Algorithm

of height values received from children. Finally S_1 and S_2 both will calculate values of average density and height for respective tentative trees.

Based on average density and depth, sink nodes will estimate the schedule lengths of their tentative trees. In this example, schedule length of tree rooted at S_1 will be less i.e. smaller than average schedule length of two trees. Reverse is true for tree rooted at S_2. Sink S_1 will flood a message BAL_NOT_REQD in its tree. Sink S_2 will flood BAL_REQD message in its tree. S_2 estimates the maximum level (h^{bal}) of its tree to have balanced schedule length. It will mention the same in the BAL_REQD message. Nodes at level $h^{bal}+1$ or higher in tree rooted at S_2 should attempt to shift to tree rooted at S_1. Other nodes would not shift.

Every node in Group 1 will broadcast SINK_CONFIRM message to tell its neighbors that it is stick to the old sink. Nodes at the boundary of two subregions (i.e. node 25,34,43,52,61,70,79,88,97) are the first ones to hear SINK_CONFIRM message from nodes in Group 1. These nodes will try to switch to S_1. Every node will estimate the possible schedule length of S_1 if it joins the sink. If estimated schedule length is less than or equal to balanced schedule length, node would switch to S_1. Else it would stick to the old sink. In both the cases, node broadcasts a SINK_MODIFIED message to inform its home sink to its neighbors. The message contains following fields: flag sink_changed, ID of new home sink and estimated new schedule length of S_1. If node has changed the sink, it will set value of sink_changed flag to 1 and will set ID to 12 (i.e. S_1). Else it will set value of sink_changed flag to 0 and will set ID to 65 (i.e. S_2).

For example, let us assume that node 61 (node filled with red color) switches to S_1. It will write the new estimated schedule length of S_1 considering that itself joins S_1. Consider that original schedule length of S_1 is 10. The node 61 is 3 hops away from S_1. In neighborhood of 61, no node is at distance 3 from S_1. So if node 61 joins the sink S_1, new schedule length of S_1 will be $10 + 0 + 1 = 11$. In SINK_MODIFIED message, node 61 will mention value 11.

Consider node 63 (filled with gray color). It is 4 hops away from S_1. It receives SINK_MODIFIED message from 61. Node 61 is 3 hops away from S_1. It knows that if 61 joins S_1, the resulting schedule length is likely to be 11. Now assume that two neighbors of 63, namely 71 (filled with green color) and 53 (filled with yellow color) have switched to S_1. Both have broadcasted SINK_MODIFIED message. So node 61 is aware that 71 and 53 have joined S_1. Thus, number of nodes at same distance as node 63 from S_1 and have switched to S_1 are 2. Node 63 estimates new schedule length of S_1 as 11 (as received from 61) + 2 (two neighbors at the same level as itself from S_1 have switched to S_1) + 1 (itself will require one slot) = 14. If this estimated value is less than balanced schedule length, node 63 will switch to S_1. Else it will stick to S_2.

In this example, node 63 is at hop distance 4 from S_1. In its neighborhood, there is only one node (node 61) at distance 3 from S_1. There may be multiple nodes at level 3 in neighborhood of node 63. So node 63 may receive multiple SINK_MODIFIED messages. In that case, it will use maximum of received values of schedule estimate in its calculation (i.e. first term in sum) of new schedule of sink1.

Switching process would start from border of two regions. It would progress from border to left i.e. towards S_2. Slowly nodes in left side would switch to S_1. As we move towards left boundary of region, chances of node switching to S_1 gets reduced. Because, as distance from S_1 increases, the estimated schedule length (if node switches to S_1) increases. When estimated schedule length starts approaching balanced schedule length, switching process stops. Thus it is ensured that schedule length of S_2 decreases, but that of S_1 does not cross balanced schedule length. Once switching process is complete, DICA[21] would be executed to perform actual scheduling & tree formation.

5.3.2 Flow Diagram of the Algorithm

Proposed algorithm is explained through flow diagrams in Figures 5.3 and 5.4. The figures are self-explanatory. First the steps of Figure 5.3 are executed. Then steps mentioned in Figure 5.4 are executed. Details are not mentioned in flow diagram. But it helps the reader to understand overall approach.

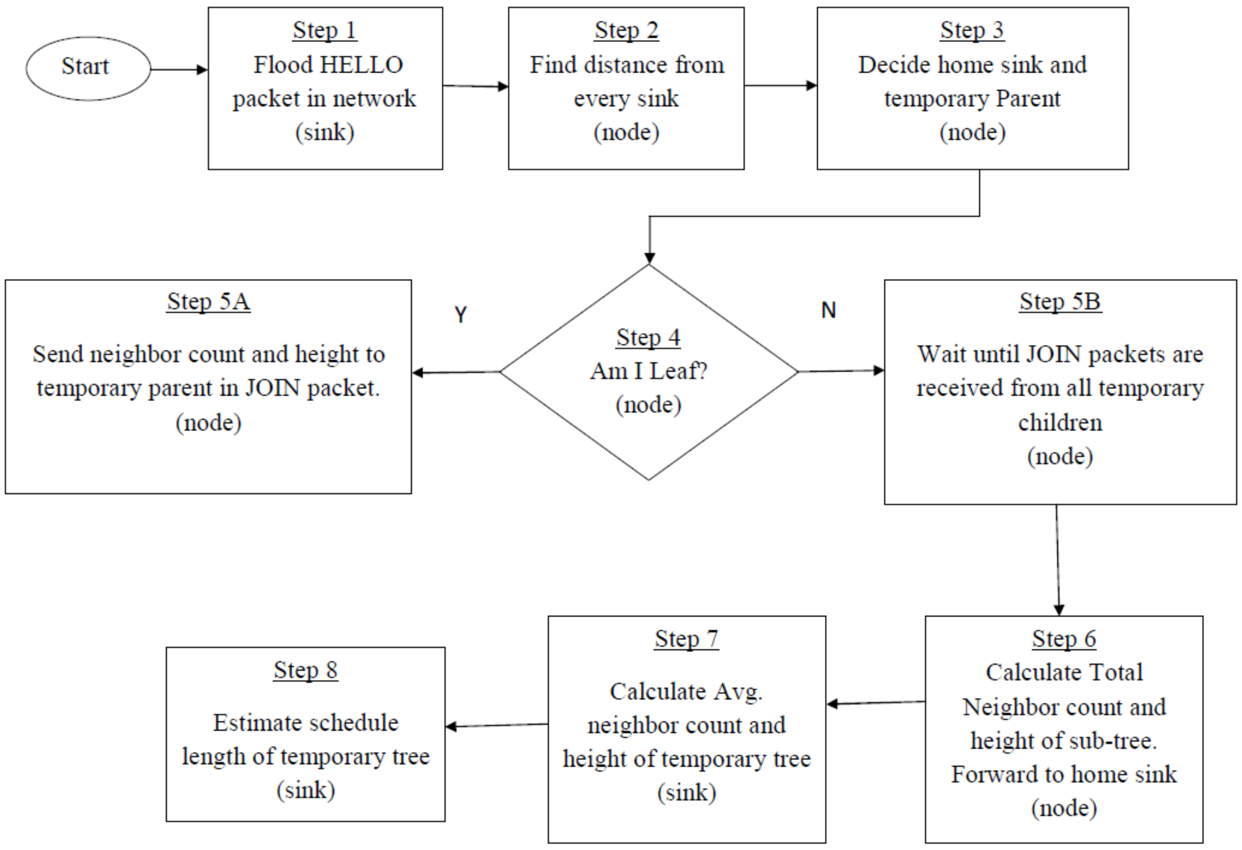

Start
Step 1
Flood HELLO packet in network (sink)
Step 2
Find distance from every sink (node)
Step 3
Decide home sink and temporary Parent (node)
Step 4
Am I Leaf? (node)
Y
N
Step 5A
Send neighbor count and height to temporary parent in JOIN packet. (node)
Step 5B
Wait until JOIN packets are received from all temporary children (node)
Step 6
Calculate Total Neighbor count and height of sub-tree. Forward to home sink (node)
Step 7
Calculate Avg. neighbor count and height of temporary tree (sink)
Step 8
Estimate schedule length of temporary tree (sink)

Figure 5.3: Flow Diagram of SLBMHM Algorithm - Part I

5.3.3 Steps of the Algorithm

In this subsection, steps of SLBMHM algorithm are mentioned. Important notations used throughout the algorithm are summarized in Table 5.1.

The steps are as follows:

1. Flooding of HELLO packets and leveling of nodes:

   (a) Every sink S_i floods HELLO packets in the network turn by turn.

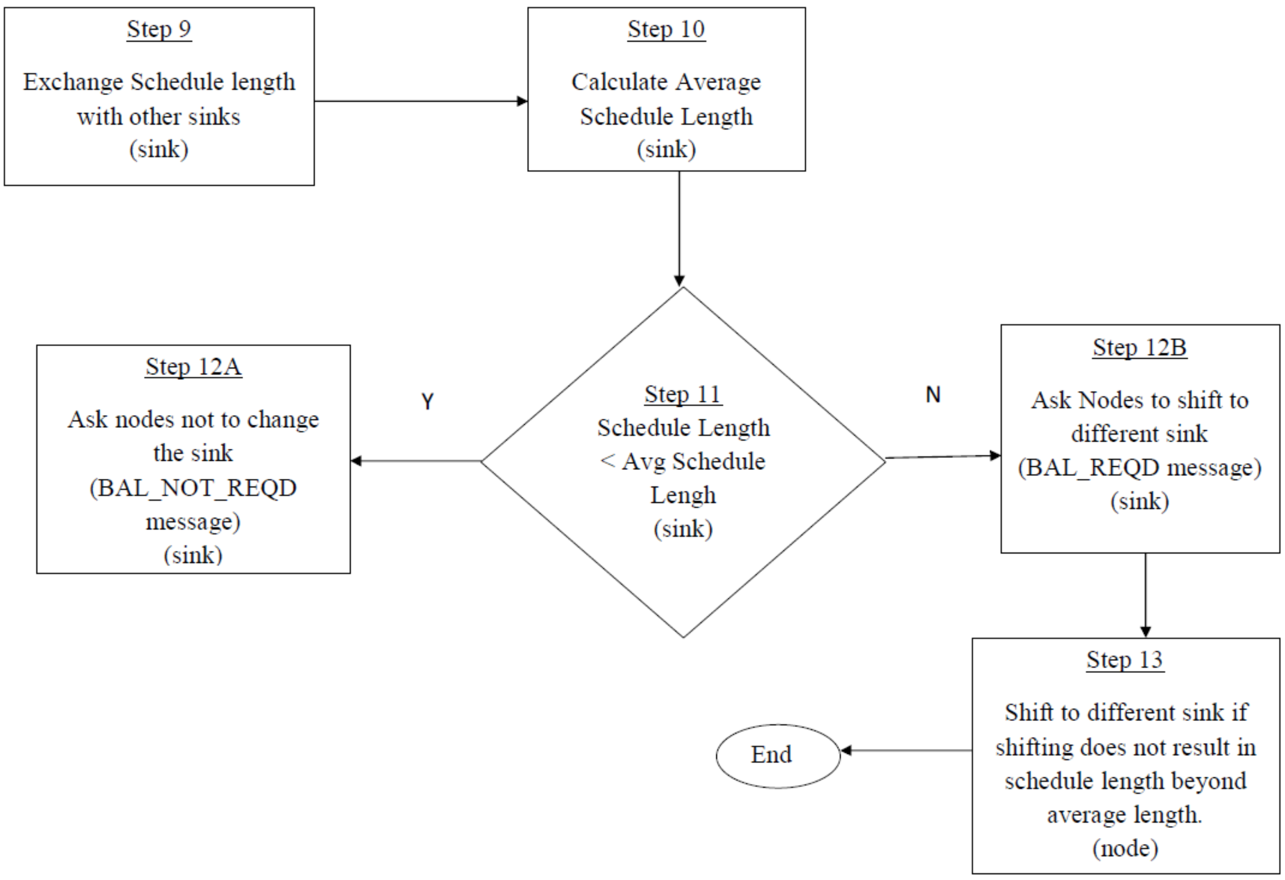

Step 9
Exchange Schedule length with other sinks (sink)
Step 10
Calculate Average Schedule Length (sink)
Step 11
Schedule Length < Avg Schedule Lengh (sink)
Y
N
Step 12A
Ask nodes not to change the sink (BAL_NOT_REQD message) (sink)
Step 12B
Ask Nodes to shift to different sink (BAL_REQD message) (sink)
Step 13
Shift to different sink if shifting does not result in schedule length beyond average length. (node)
End

Figure 5.4: Flow Diagram of SLBMHM Algorithm - Part II

(b) HELLO packet has a field $LEVEL$. It is initialized to 0 by sink as sink is at $LEVEL$ 0.

(c) Every node receiving HELLO packet generated from S_i performs following steps:

i. If $LEVEL$ value in HELLO packet is smaller than current value of d_i, next two steps are executed. Else no action in taken.

ii. Set d_i to $LEVEL + 1$.

iii. Increment $LEVEL$ field of HELLO by 1 and broadcast HELLO so that its neighbors would receive it.

At the end of step 1, every node knows its distance from each of N sinks. That is every node has updated the vector (d_1, d_2, d_3, ...,d_N).

2. Formation of temporary trees: Every node performs following steps to form tem-

Parameter	Meaning
n	No. of nodes
N	No. of sinks
S_i	Sink i
HELLO	Packet initiated by sinks for leveling of network
LEVEL	Field in HELLO packet
d_i	Distance in hop count from sink S_i (initialized to infinity)
h	Height of given node in temporary tree i.e. distance in hop count from temporary home sink
temp_home_sink	Temporary home sink of given node
temp_parent	Temporary parent of given node in temporary tree
JOIN	Message sent by a node to its temporary parent
σ_i	Average density of tree rooted at S_i
h_i	Height of tree rooted at S_i
SH_i^{est}	Estimated schedule length of tree i
SH_{bal}	Average of schedule lengths of all trees i.e. balanced value
h_i^{bal}	Required height of tree of S_i to achieve SH_{bal}
BAL_REQD	Message sent by overloaded sink in its tree
BAL_NOT_REQD	Message sent by underloaded sink in its tree
SINK_MODIFIED	Message broadcasted by a node to inform its neighbors about tree-switching
sink_changed	Flag present in SINK_MODIFIED message. It is 0 if sender of SINK_MODIFIED decides not to change temp_home_sink. Else it is 1.
SH_i^{cest}	Current estimated schedule length of tree rooted at S_i. Used during tree switching.
$nbr\_switched_i$	No. of neighbors of given node which switched to sink S_i and are at same hop distance from S_i as given node.

Table 5.1: Notations used in SLBMHM Algorithm

porary trees.

(a) Temporarily join the sink which is at the minimum distance. Every node joins S_i such that $d_i = \min(d_1, d_2, d_3, ..., d_N)$. Height h of node is set to d_i. The variable temp_home_sink is set to S_i

(b) Select any one node as parent which is at smaller distance from the selected sink. This parent is temporary. The ID of temporary parent is stored in temp_parent. Actual scheduling and parent selection will be performed later.

At the end of step 2, network is divided into N temporary trees. Every tree is rooted at one sink.

3. Estimation of schedule length by every sink: Following steps are executed so that every sink could estimate schedule length of its temporary tree.

   (a) Every leaf node will send a JOIN message to temporary parent so that parent would add the node in its temporary children list. JOIN message sent by leaf nodes contains following information.

      i. Neighbor count. The neighbors are those nodes which are in direct radio range of given node and have same temp_home_sink.

      ii. Height h.

      iii. Node count. This is the number of nodes in sub-tree rooted at sender of JOIN message. This field contains value 1 as leaf node is the only node in its subtree.

   (b) Every non-leaf node at level h will wait to hear JOIN messages from its neighbors at level $(h + 1)$) which belong to same temp_home_sink. Once it hears JOIN messages from all neighbors at level $(h+1)$, it sends a JOIN message to its temporary parent. JOIN message sent by non-leaf node contains following information.

      i. Total neighbor count. It is sum of neighbor counts received from temporary children and node's own neighbor count.

      ii. Maximum of height values received from temporary children.

      iii. Total Node count. It is sum of node counts received from temporary children plus 1. As mentioned earlier, this field indicates number of nodes present in the sub-tree rooted at given node.

   (c) At the end of above step, every sink S_i receives JOIN messages from its temporary children. Every sink is able to calculate following parameters about its temporary tree.

      i. Total neighbor count. Sink S_i finds sum of total neighbor counts received from its temporary children.

      ii. Height (h_i) of the tree.

iii. Total node count. It is sum of node counts received from its temporary children.

iv. Average density (σ_i). It is ratio of total neighbor count and total node count.

v. Estimated Schedule length (SH_i^{est}). It is the estimate of schedule length of temporary tree rooted at S_i. It is function of σ_i and h_i. It is derived by putting values of σ_i and h_i in Equation 5.9. Detailed explanation about schedule length estimation is given later in the chapter.

(d) Sinks exchange estimated schedule lengths and find average schedule length. It is referred as 'balanced schedule length', denoted as SH_{bal}.

4. Schedule length balancing: Following steps are performed for schedule length balancing.

   (a) If sink S_i finds that SH_i^{est} is greater than SH_{bal}, it attempts to remove some nodes from its tree. It sends BAL_REQD message in its tree to inform the nodes that balancing is required. Else it sends BAL_NOT_REQD message in the tree.

   (b) As part of BAL_REQD message, sink S_i sends following parameters:

      i. Estimated schedule lengths i.e. SH_1^{est},SH_2^{est},....,SH_N^{est}.

      ii. Balanced schedule length (SH_{bal}).

      iii. Required height (h_i^{bal}) of the tree rooted at sink S_i to achieve balanced schedule length. The value of h_i^{bal} is calculated by putting SH_{bal} and average density σ_i of the tree in Equation 5.9. Sink indicates that all the nodes in its tree at a level greater than h_i^{bal} should attempt to switch to a different tree.

   (c) If a node whose temp_home_sink is S_i, receives BAL_NOT_REQD message from S_i, it confirms attachment to sink S_i by broadcasting SINK_CONFIRM message.

   (d) If a node whose temp_home_sink is S_i, receives BAL_REQD message from S_i, the node will broadcast SINK_MODIFIED message with sink_changed flag set to 0 if d_i is less than or equal to h_i^{bal}.

(e) In the above case, if d_i is greater than h_i^{bal}, following steps are executed to change home sink.

   i. Wait for neighbors belonging to sinks having schedule length less than that of S_i to finalize their sinks either by deciding to stick with same sink or changing to new sink. Then following steps are performed.

   ii. Create a set of target sinks. A sink S_j is member of the set if following two conditions are satisfied: (i) Schedule length of S_j is less than balanced schedule length. (ii) At least one node is present in neighborhood of given node which belongs to S_j and is nearer to S_j compared to given node. If the set is empty, node keeps waiting. When it overhears a SINK_MODIFIED message (described later), it tries to create the set again. If set is non-empty, following steps are performed.

   iii. Assume that Z sinks are present in the set of target sinks. The set is denoted at *tgt_sinks*. If sink S_j is present in set *tgt_sinks*, its current schedule length is denoted as SH_j^{cest}.

      - SH_j^{cest} is same as SH_j^{est} if the node (i.e. the one trying to switch) is one hop away from a node whose temp_home_sink is S_j.
      - Otherwise SH_j^{cest} is set as follows. Given node is more than one hops away from node(s) whose temp_home_sink is S_j. But still S_j is in set Z. It means that some neighbor has switched to S_j. When a node switches to new sink, it broadcasts SINK_MODIFIED message with sink_changed flag set to 1. In addition, it also writes new estimated value of SH_j^{cest} in the message. Given node may overhear many such messages. It sets SH_j^{cest} to the maximum of received values of SH_j^{cest}.

   iv. Node estimates new schedule length of every sink S_j in set tgt_sinks considering that it would switch to the sink. Number of neighbors who belong or switched to sink S_j be *nbr_switched*$_j$. Then new value of SH_j^{cest} is estimated as follows:

$$SH_j^{cest} = SH_j^{cest} + nbr\_switched_j + 1 \tag{5.1}$$

      The rationale behind above formula is as follows: The value present in SH_j^{est} is the current estimated schedule length of sink S_j as known to given node. At one time multiple nodes are attempting to switch to a different sink. Given node overhears $SINK\_MODIFIED$ messages from neighbors. So it knows which neighbors have switched to which sink. It can not use the same slots as used by its neighbors otherwise collision may occur at other nodes. Second term $nbr\_switched_j$ takes care of the same.

The term $nbr\_switched_j$ is the count of neighbors switched to S_j and at the same hop distance from S_j as given node. The nodes at a lower distance may have already switched. The nodes at a higher distance may be waiting to switch. Actual scheduling algorithm proceeds in bottom-up manner. So a node at height h would be scheduled only after its neighbors at height $h + 1$ are scheduled. At one time competing nodes are those at the same level.

The last term in above equation is '1'. Given node transmits one packet. So it will consume one slot. Thus one additional slot should be added in current schedule length.

v. Node updates SH_j^{cest} for each sink S_j in set tgt_sinks. It decides to shift to sink S_k whose SH_k^{cest} minimum and is less than SH_{bal}. If no such sink is found, node sticks to current sink and broadcasts SINK_MODIFIED message with flag sink_changed set to 0.

vi. When node decides to switch to sink S_k, it broadcasts SINK_MODIFIED message. The message has *sink_changed* flag set to 1. The message notifies the neighbors about node's decision. The message also contains latest value of SH_k^{cest} so that neighbors could update their estimates of SH_k^{cest}.

5. Once every node finalizes the sink, scheduling & tree formation algorithm DICA [21] is executed. Every node selects a slot and parent. Thus at the end, N different trees are formed. Every tree is rooted at one sink.

5.3.4 Correctness of the Algorithm

Lemma 5.3.1. *In homogeneous network, schedule length of tree T depends on it's average density (neighbor count, σ) and height (h).*

Proof. For collision free schedule formation it is required that transmission slot selected by given node should be such that it does not create interference at neighboring nodes. Suppose that number of neighbors of node n is n_c. Each one of n_c neighbors is receiving in certain slot. Thus node n can not transmit in those n_c slots. The slot selected by n should be different than those n_c slots. If network is dense, value of n_c would be high. If network is sparse, n_c would be low. Thus in a dense network, number of slots consumed are more compared to sparse network.

Slot assignment should be bottom-up in aggregated convergecast. That is, slot assignment should progress from leaf to root. Leaf node(s) should be assigned lowest timeslot. Time slots increase from leaf to root. This criteria is essential for aggregation freshness i.e. parent could aggregate children's packets in the same TDMA cycle and transmit further. Otherwise parent would receive packets from children in one TDMA cycle and could forward in next TDMA cycle. If tree height is more, leaf is far from root. So more slots are required to reach the root. This in turn means that if height increases, schedule length increases.

In raw convergecast, scheduling need not be bottom to top. But every node has to forward all the packets coming from children node. If leaf node is far from root, its packet has to travel longer path to root. At each hop, packet needs one time-slot. So as distance from root increases, number of required time slots also increases.

Thus whether it is raw or aggregated convergecast, schedule length depends on tree height and average density.

□

Lemma 5.3.2. *SLBMHM algorithm ensures that every node has at least one path to the sink.*

Proof. In SLBMHM algorithm, first temporary trees are formed. Every node joins the sink which is at least distance compared to other sinks. Node finds its distance from sinks based on $LEVEL$ field in HELLO packets received from sinks. If HELLO packet from sink comes to a node, it means that there is at least one path from node to sink. Node selects any one node as temporary parent. Thus unique path is guaranteed. Through that path values of neighbor count and height reach the sink.

During schedule length balancing phase, a node belonging to an overloaded sink tries to switch to a tree of underloaded sink. As mentioned in algorithm, a node selects a sink as new home-sink subject to following conditions: (i) Current schedule length of that sink is less than average schedule length. (ii) There is at least one node present in neighborhood which belongs to new home-sink and is nearer to new sink compared to current node. The second condition ensures that node has at least one path to sink. During actual scheduling and tree formation phase, node selects a node as parent which belongs to the same home-sink as itself. Due to condition (ii) above, there will be at-least one such node in neighborhood of sink. So, it will be able to select such a node as parent. □

Lemma 5.3.3. *SLBMHM algorithm reduces schedule length of overloaded sink(s) without increasing schedule length of underloaded sink(s) beyond average.*

Proof. In SLBMHM algorithm, every sink estimates its schedule length. Sinks exchange schedule lengths and calculate average (or balanced) schedule lengths.

- Consider that number of sinks is two i.e. S_1 and S_2.

  Assume that schedule length of S_1 is more than average schedule length. Schedule length of S_2 is less than average schedule length. S_1 will ask its nodes at a level higher than certain threshold level h^{bal} to switch to S_2, if possible. Every node i decides whether to switch to S_2 or not based on following criteria: If resulting schedule length of S_2 is less than balanced schedule length, then only node would switch. When a node switches to new tree, it broadcasts estimated new schedule length of that tree. Thus other nodes would also know about new schedule length of that tree. Thus when a node takes decision of whether to switch or not, it has latest information about schedule length of the tree to which it wants to switch. It would not switch if resulting schedule length is more than balanced value.

- Consider that number of sinks is more than two, i.e. S_1,S_2,S_3,....,S_N

Consider that schedule length of S_1 is more than average schedule length. It would ask its nodes at a level higher than h^{bal} to shift to a different tree. Here node may have multiple choices of sinks to switch. Node would prepare a list of tentative target sinks. Every sink S_j present in this set would meet following two conditions: (i) Schedule length of S_j is less than balanced schedule length. (ii) At least one node is present in neighborhood of given node which belongs to S_j and is nearer to S_j compared to given node. For each S_j present in the set, node estimates new schedule length as if it switches to S_j. Node would switch to the sink whose estimated schedule length is less than balanced schedule length and least compared to other sinks in the set. If there is no sink found whose estimated schedule length is less than balanced schedule length, node would stick to old sink.

Thus it is attempted to reduce schedule length of one sink, but without increasing schedule length of the other sink beyond balanced value.

□

Lemma 5.3.4. *If schedule lengths of trees are balanced, maximum schedule length of the network is also reduced. Thus average number of slots before a node gets its turn to transmit also gets balanced.*

Proof. Assume that there are N sinks in the network. The corresponding trees are T_1,T_2,...,T_N. Their schedule lengths are SH_1,SH_2,....,SH_N respectively. Let average schedule length be SH_{bal}. The maximum schedule length (or overall schedule length of network) SH_{max} is max (SH_1,SH_2,....,SH_N).

All nodes and sinks are part of same network. But different trees can be scheduled independently i.e. on different frequency channels. If schedule lengths are balanced, the difference between SH_i and SH_{bal} would be small for every tree T_i. After balancing, revised maximum schedule length SH'_{max} is likely to be around SH_{bal}. Thus $SH'_{max} < SH_{max}$. So it is proved that maximum schedule length is reduced.

If schedule lengths are not balanced, number of slots a node has to wait to get its turn to transmit depends on which tree it belongs to. If node belongs to tree with small schedule length, it would get its turn to transmit very quickly. But if it belongs to a tree with large schedule length, it has to wait for longer period of time before it gets its turn.

When schedule lengths are balanced, schedule lengths of all trees are near to SH_{bal}. So no matter which tree a node belongs to, it has to wait for approximately SH_{bal} slots to get transmission turn. Thus all nodes have to wait for almost equal number of slots.

Thus balancing schedule length not only reduces overall schedule length of the network but also reduces latency.

$\square$

5.3.5 Schedule length (SH) as function of Average Density (σ) and Height (h) of Tree

It is mentioned in proposed algorithm that every sink estimates schedule length (SH) of its temporary tree based on following two parameters: average density (σ) and height (h) of tree. To establish relationship between SH, σ and h, a simulation based study is carried out as follows.

Single sink network is considered. For a given value of σ, values of h are gradually changed. For each h, DICA[21] as explained in Chapter 2 is executed and SH is calculated.

Following values of densities are used: $\sigma_1 = 4$, $\sigma_2 = 8$, $\sigma_3 = 12$ and $\sigma_4 = 20$. Meaning of $\sigma = x$ is that there are average x neighbors of every node. In other words, x nodes are present in radio range of every node. For each σ_i, depth is varied from 2 to 20. Thus there are four graphs of schedule length (SH) v/s. height (h). These equations are mentioned below:

$$SH_1 = 1.07 * h + 4.57, (\sigma = 4) \tag{5.2}$$

$$SH_2 = 1.88 * h + 9.07, (\sigma = 8) \tag{5.3}$$

$$SH_3 = 1.22 * h + 18.84, (\sigma = 12) \tag{5.4}$$

$$SH_4 = 4.90 * h + 35.81, (\sigma = 20) \tag{5.5}$$

It is observed that schedule length varies linearly with height. Thus,

$$SH = m * h + c \tag{5.6}$$

Here m is the slope of graph and c is y-axis intercept. It can be observed that slope values (1.07,1.88,1.22 and 4.90) vary with σ. Also value of intercept c (4.57, 9.07,18.84 and 35.81) vary with σ. Thus we can plot graphs of slope v/s. density and intercept v/s. density.

Following equation shows relation of slope and density.

$$m = 0.2 * \sigma + 0.24 \tag{5.7}$$

Following equation shows relation of intercept and density.

$$c = 2 * \sigma - 5 \tag{5.8}$$

Putting values of m and c in equation 5.6, final SH can be presented as function of σ and h as follows.

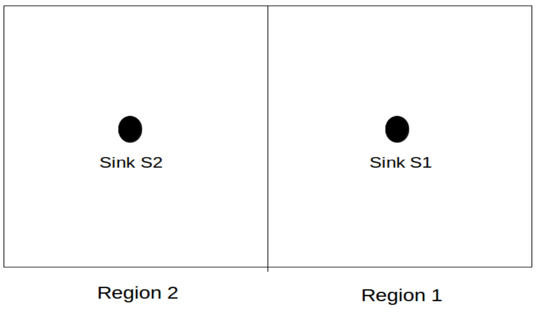
$$SH = p\sigma h + qh + r\sigma - v \tag{5.9}$$

In equation 5.9 above, values of p, q,r and v are 0.2,0.3,2 and 5 respectively. The values of σ used are from 4 to 20 and height h is between 2 to 20.

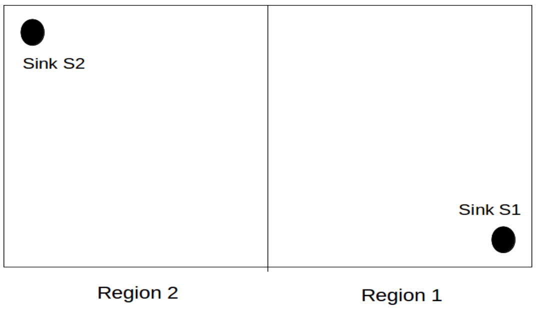
Every sink knows average density (σ) and height (h) for its tentative tree. So above equation can be used by a sink to estimate schedule length of its tree.

5.4 Simulation Results

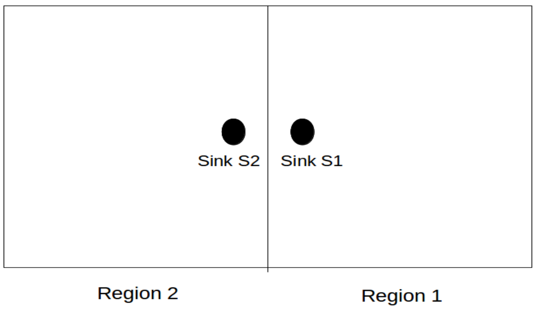
In this section, simulation results are discussed. Details of simulation design, simulation setup, other protocols used for comparison and different performance parameters are also discussed.



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner



(c) Sinks are nearby

Figure 5.5: Different Sink Deployments

5.4.1 Simulation Design

In Figures 5.5a, 5.5b and 5.5c, different sink deployments are illustrated. The same is listed below. There are two sinks: S_1 and S_2.

1. Sinks are in center of each sub-region (Figure 5.5a).

2. Sinks are at opposite diagonal corners (Figure 5.5b).

3. sinks are near to each other (Figure 5.5c).

The three cases differ only in sink placements. Else they are same. Common details are presented next. A square region of 200 meters x 200 meters is used for simulation. It is divided into a grid of 20 x 20 points. Distance between any two horizontal or vertical grid

Scenario	Probability p_1^d	Probability p_2^d	Density Deviation (σ_{dev})
1	0.3	0.3	3
2	0.3	0.5	4
3	0.3	0.7	5
4	0.3	0.9	6

Table 5.2: Node Distribution Scenarios

points is 10 meters. The whole region is divided into two sub-regions. Each sub-region is of size 100 meters x 200 meters. Each region contains one sink.

Nodes are deployed randomly in the network such that region 1 has low node density and region 2 has high node density. In region 1, probability of node being present at a grid point be p_1^d. Similarly let p_2^d be the probability of node being deployed at a grid point in region 2. Four different scenarios are generated. Value of p_1^d is fixed to 0.3 in all scenarios. But from scenario 1 to scenario 4, values of p_2^d is varied as 0.3,0.5,0.7 and 0.9. That is from scenario 1 to scenario 4, node density of region 2 is incremented.

Assume that there are n nodes in the network. Let σ_i be the neighbor count of node i, then average density σ is defined as follows:

$$\sigma = \frac{\sum_{i=1}^{n} \sigma_i}{n} \tag{5.10}$$

Density deviation (σ_{dev}) for the entire network is calculated as follows:

$$\sigma_{dev} = \sqrt{\frac{\sum_{i=1}^{n} (\sigma - \sigma i) * (\sigma - \sigma i)}{n}} \tag{5.11}$$

The four scenarios are summarized in Table 5.2.

As seen from Table 5.2, density deviation increases as region 2 becomes denser. This is natural because as density of region 2 increases, average number of neighbors per node in region 2 increases. But average number of neighbors per node in region 1 is fixed. As a result, deviation in density increases.

The SLBMHM algorithm is evaluated with respect to different performance parameters. These parameters are discussed in next sub-section. But for every parameter, performance is evaluated by varying density deviation. In other words, X-axis is density deviation in all graphs of simulation results.

There are three different sink deployments. For each deployment, there are four dif-

Parameter	Value
Node deployment	Random
Area	200m x 200m
Radio Range	30m
Transmission Power Consumption	0.660 W
Receive Power Consumption	0.395 W
Sleep Power Consumption	0 W
Data Generation Rate	1 packet every 10 seconds
Simulation Time	5500 Seconds

Table 5.3: Simulation Setup

ferent node distribution scenarios. Performance of SLBMHM is evaluated for each combination of sink deployment and node distribution scenario. For a specific sink deployment, four different instances of each node distribution scenario are randomly generated. So, each point in graph is an average of four different simulation runs. Corresponding standard deviation is also calculated and plotted as error bar.

5.4.2 Simulation Setup

Table 5.3 summarizes different parameters used for simulation. Radio range of nodes is set to 30 meters. The same is used in simulation setup in Chapter 4. Simulation time is 5500 seconds. Out of which, 5000 seconds are used for execution of schedule length balancing and scheduling & tree formation algorithm. Rest 500 seconds are for data transfer. During these 500 seconds, nodes generate data and send to sink. As mentioned in table, every node generates 1 data packet every 10 seconds.

5.4.3 Discussion of Results

In literature, there are many existing algorithms for multi-sink networks. To evaluate our proposed algorithm, we have compared the same with following different approaches.

1. Hop count based approach: Every node selects nearest sink as home sink. Here nearest means the sink reachable in the least number of hops. Graphs for this approach are labeled as 'hop-count'.

2. LBR (Load Balanced Routing)[34]: It deals with tree formation in multi-sink networks. Every node selects a sink based on ratio of number of neighbors of sink and

distance from sink in hop count. The sink with highest ratio is selected as home sink. Corresponding graphs are labeled as 'LBR' (Load Balanced Routing).

3. SMTLB (Spanning Multi Tree Load Balanced routing) [38]: It also deals with tree formation in multi-sink networks. Initially every one hop neighbor of a sink becomes root of a subtree. Thus multiple subtrees are initiated. Subtrees grow in parallel. At every iteration, the least loaded tree is selected for expansion.

Performance Measures

Following performance parameters are studied through simulation. Number of nodes is denoted by n.

1. Density Difference (σ_{frac}) : Let Average Density of Tree T_i be σ_i. Difference between densities of two trees (σ_{frac}) is defined as follows:

$$\sigma_{frac} = \frac{\mid \sigma_1 - \sigma_2 \mid}{max(\sigma_1, \sigma_2)} * 100\% \tag{5.12}$$

2. Maximum Schedule Length (SH) : It is overall schedule length of the network. Let SH_i be schedule length of Tree T_i. Maximum schedule length of network (SH) is defined as follows:

$$SH = max(SH_1, SH_2) \tag{5.13}$$

3. Difference in Schedule Length (SH_{frac}) : It indicates how much the schedule lengths of two trees differ. Difference in schedule length SH_{frac} is defined as follows:

$$SH_{frac} = \frac{\mid SH_1 - SH_2 \mid}{max(SH_1, SH_2)} * 100\% \tag{5.14}$$

4. Control Overhead (CO): It is as defined in equation 3.3.

5. Energy Consumption During Control Phase (E_C) : It is as defined in equation 3.5.

6. Energy Consumption During Data Phase (E_D) :It is as defined in equation 3.7.

In next subsection, simulation results for all above parameters are presented. The SMTLB[38] algorithm is centralized in nature. It is implemented as an application program. It is not implemented in NS-2. For SMTLB, results of fractional density difference, fractional schedule length difference and maximum schedule length are generated. But results for other parameters are not generated.

Fractional Density Difference, Fractional Schedule Length Difference and Maximum Schedule Length

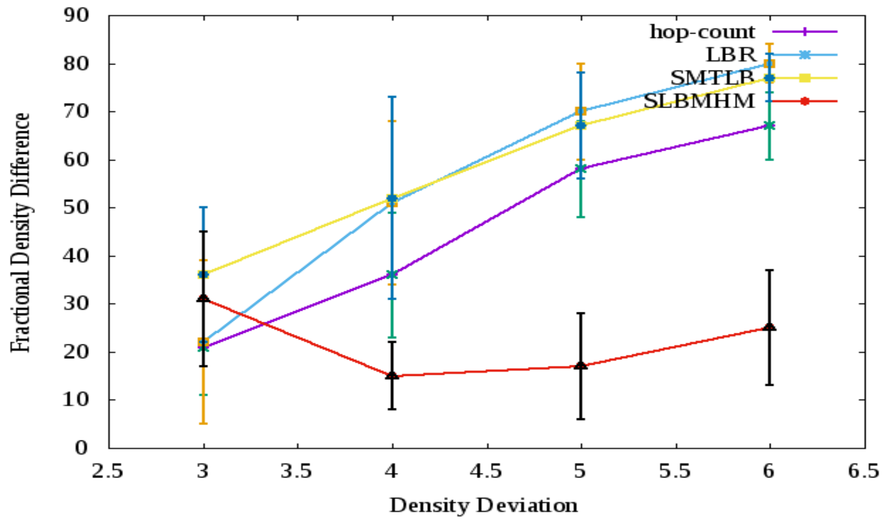
In Figures 5.6a,5.6b and 5.6c the graphs of Fractional Density Difference v/s. Density Deviation for three cases namely sinks in center, sinks at diagonal corner and sinks near to each other are shown.

It is seen from the graphs that SLBMHM algorithm minimizes density difference between two trees compared to other three approaches. In SLBMHM algorithm, when a sink finds that its schedule length is higher than the balanced schedule length, it asks nodes belonging to its tree to shift to a different tree. Thus nodes shift from dense region to sparse region (or dense tree to sparse tree). As a result, density of dense region is reduced and that of sparse region increases. So, difference in density is reduced.

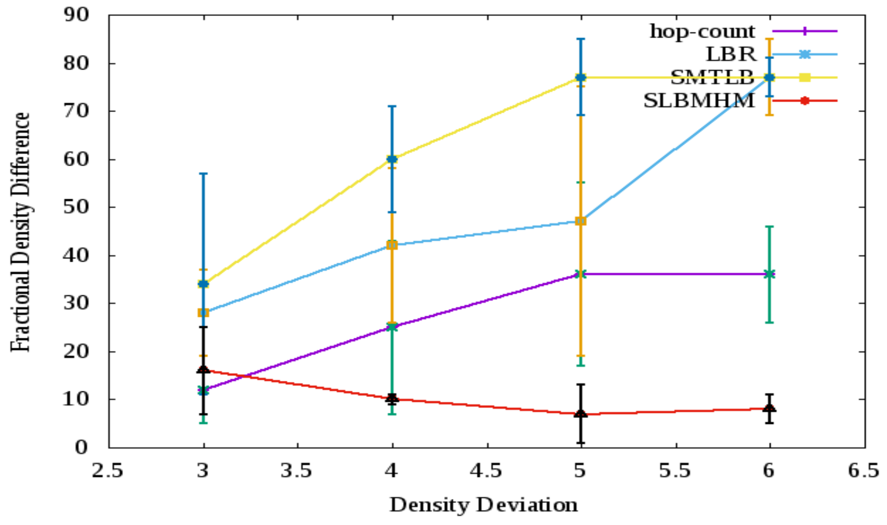
In SMTLB algorithm, subtrees grow gradually by expanding the least loaded subtree. This does not guarantee that the densities of trees remain balanced. The same is reflected in simulation results.

In LBR algorithm, parent selection is done based on ratio of neighbor count of sink and hop distance from sink. A node joins the sink with the highest ratio. The idea is to prefer the sink with more number of neighbors so that funneling effect [1] can be avoided. When two regions are different in density, nodes would prefer to join the sink present in dense region. Only those nodes which are far away from that sink would prefer to join the other sink. Thus LBR does not attempt to balance density of trees.

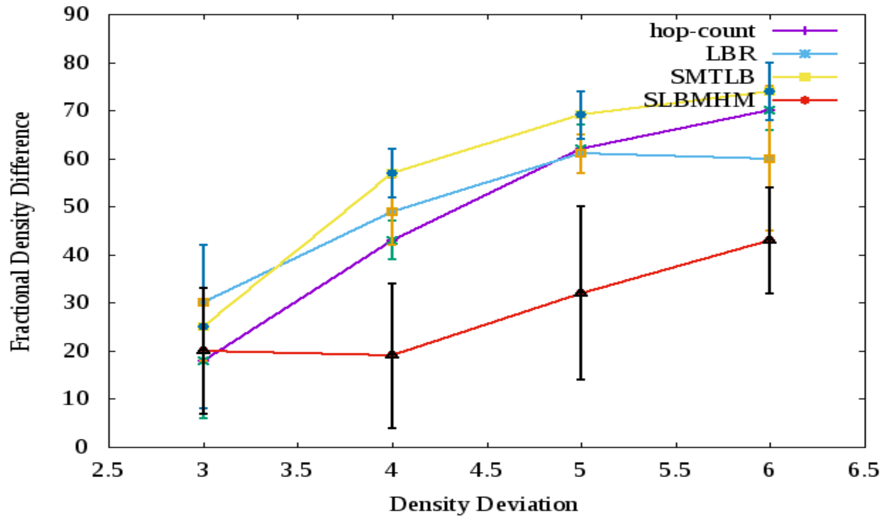
In hop count based approach, every node joins the sink which is at the least hop distance. When network is dense, more number of nodes are present in a given radius compared to a sparse network. As hop distance from sink is the only criteria to join the tree, nodes in dense part of network join the nearest sink and the resulting tree has large number nodes. Nodes in sparse region of network join the nearest sink resulting in a small tree. Thus densities of trees remain unbalanced.



(a) Sinks are at center in each sub-region

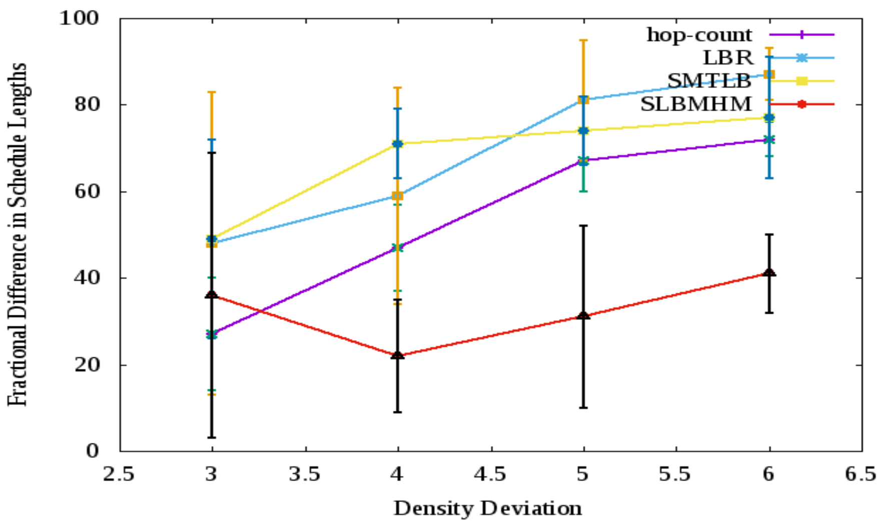


(b) Sinks are at diagonal corner in each sub-region

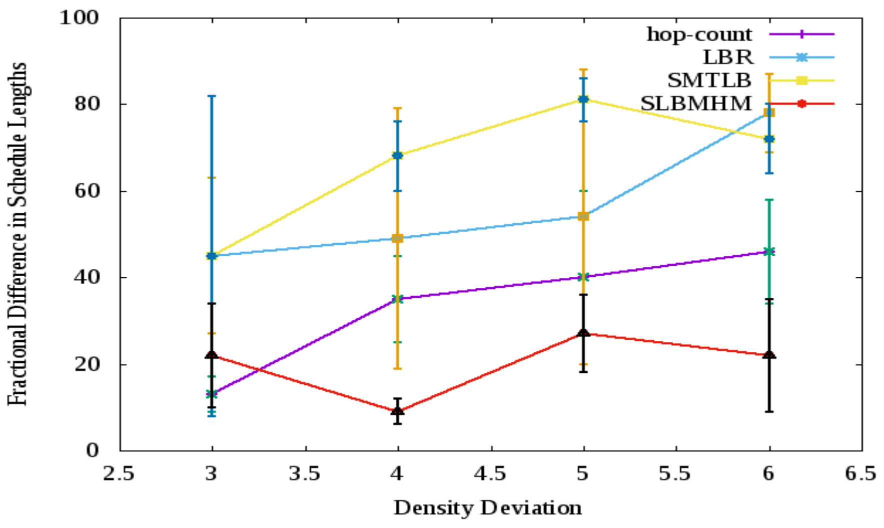


(c) Sinks are nearby

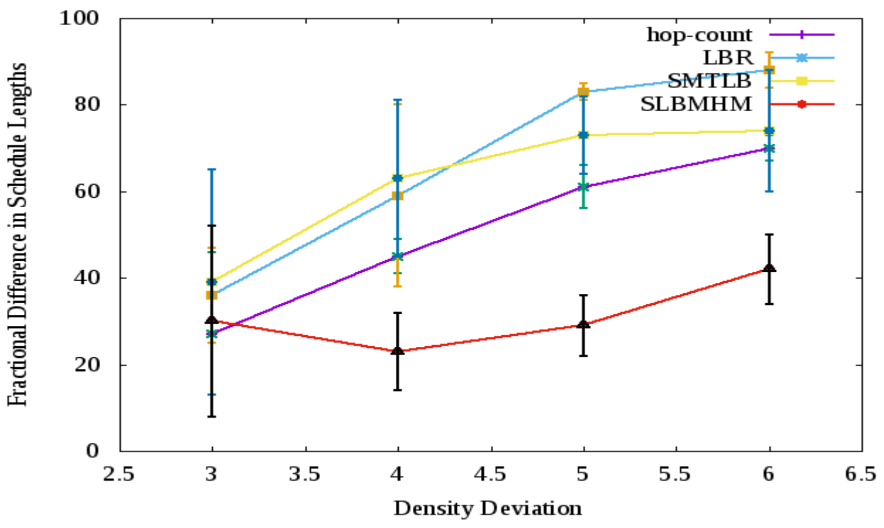
Figure 5.6: Dependency of Fractional Density Difference on Density Deviation



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner in each sub-region



(c) Sinks are nearby

Figure 5.7: Dependency of Fractional Difference in Schedule Lengths on Density Deviation

As SLBMHM algorithm is able to balance densities of trees, the resulting schedule lengths are also balanced. As other algorithms do not attempt to balance the densities of trees, they result in larger difference in schedule lengths of trees. The same is explained through Figures 5.7a,5.7b and 5.7c. It is also seen from the figures that as density deviation increases, the difference in schedule lengths increases. This is because increase in density deviation results in the tree of region 2 becoming denser and so it results in much higher schedule length than tree of region 1.

In Figures 5.8a,5.8b and 5.8c the graphs of Maximum Schedule Length v/s. Density Deviation are shown. Here two trees T_1 and T_2 are formed with schedule lengths SH_1 and SH_2, respectively. As T_2 spans over dense region, SH_2 is higher than SH_1. The max. schedule length of the network (SH) is max(SH_1,SH_2). The SLBMHM algorithm attempts to reduce SH_2 and increase SH_1 by performing density balancing i.e. shifting nodes from T_2 to T_1. The SH_1 is not increased above average schedule length SH_{bal}. As a result, max(SH_1,SH_2) goes down and remains around SH_{bal}. Thus, SH is reduced.

As mentioned earlier, other approaches result in high difference in schedule lengths of the two trees. As a result, maximum schedule length also remains high compared to SLBMHM algorithm.

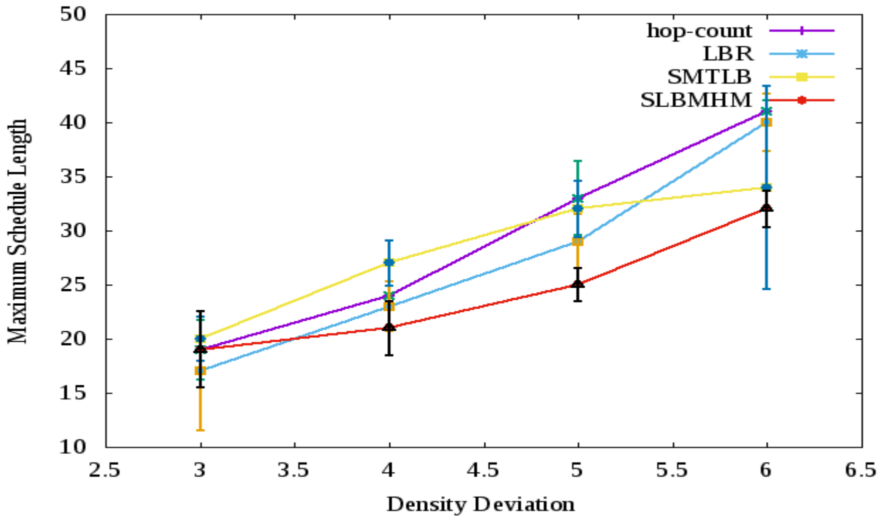
For some cases, the performance of SLBMHM is not the best. But still it is almost near to the other algorithms. Detailed quantitative analysis of results is presented in the last sub-section of current section.

Control Overhead and Energy Consumption during Control Phase

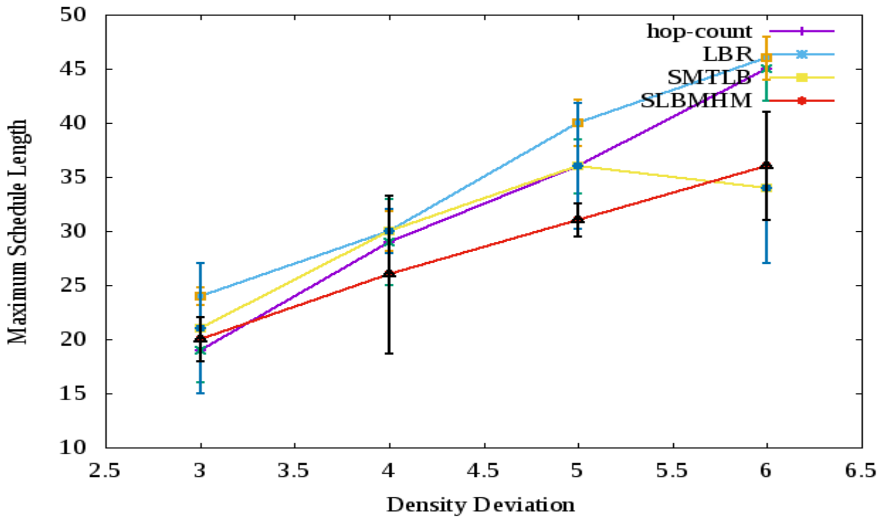
In Figures 5.9a,5.9b and 5.9c the graphs of Control Overhead v/s. Density Deviation are shown. It is observed from the graphs that as deviation increases, control overhead also increases. This is natural because increase in deviation means increase in average density of entire network. As number of nodes increases, count of control messages also increases.

When parent selection is done based on hop count, control overhead consists of following components:

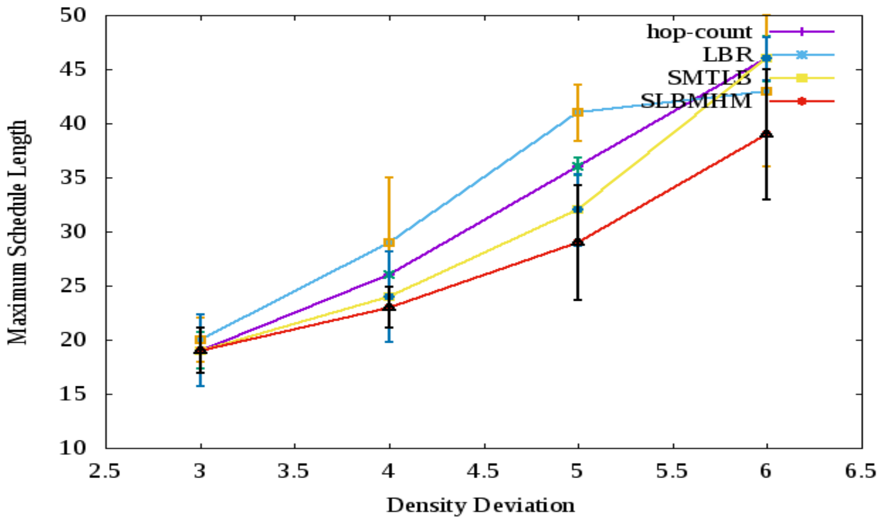
- HELLO messages flooded by sinks for leveling.
- Control messages generated due to execution of DICA[21] for scheduling & tree formation. The messages involved are Request, Response, Schedule and Forbid-



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner in each sub-region



(c) Sinks are nearby

Figure 5.8: Dependency of Max. Schedule Length on Density Deviation

den. These messages are exchanged between a node doing slot/parent selection and candidate parents of the same node.

In LBR algorithm, control overhead consists following components.

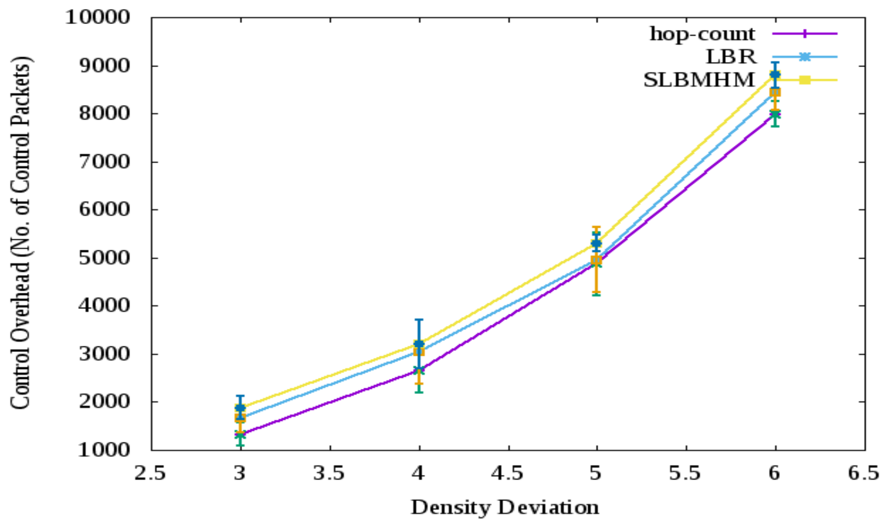
- HELLO messages flooded by sinks for leveling.
- Every sink floods its neighbor count in the network.
- Control messages generated due to execution of DICA[21] for scheduling & tree formation.

In SLBMHM algorithm, control overhead consists of following components.

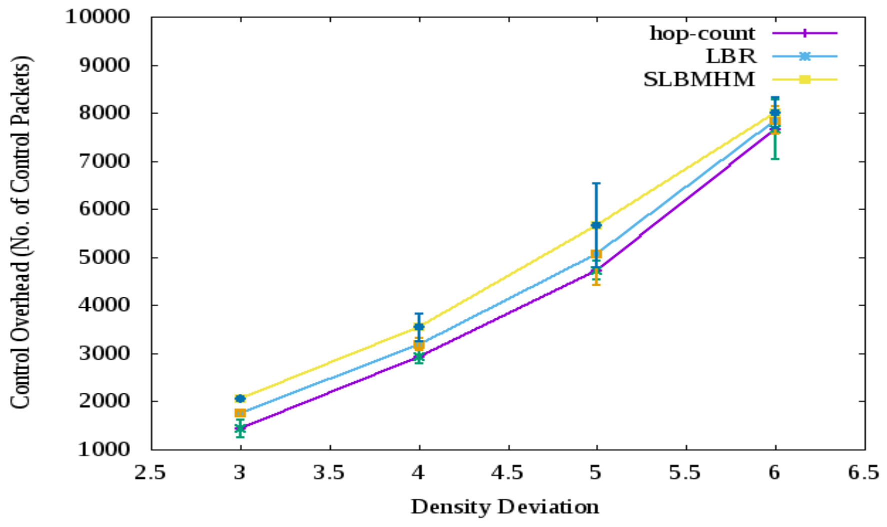
- HELLO messages flooded by sinks for leveling.
- Transmission of JOIN message by every node towards sink.
- Flooding of BAL_NOT_REQD message from sink with small schedule length in its tree.
- Flooding of BAL_REQD message from sink with large schedule length in its tree.
- Broadcasting of SINK_CONFIRM and SINK_MODIFIED messages by nodes.
- Control messages due to execution of DICA[21] for scheduling & tree formation.

The hop count based method should result in the least control overhead as it involves the least number of control messages. The SLBMHM algorithm involves maximum number of control messages so it should result in highest control overhead. The performance of LBR should remain between the two. The same is observed from Figures 5.9a,5.9b and 5.9c except in 5.9c for Figure $\sigma = 6$.

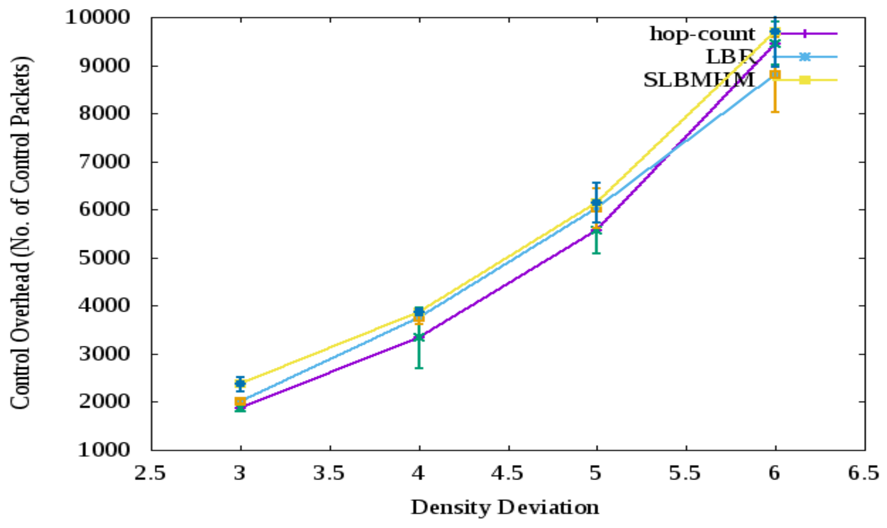
Energy consumption during control phase is directly proportional to control overhead. In Figures 5.10a,5.10b and 5.10c graphs for the same are shown. The nature of the graphs is same as in Figures Figures 5.9a,5.9b and 5.9c. So, more explanation is not given here. Detailed quantitative analysis of results is presented in the last sub-section of current section.



(a) Sinks are at center in each sub-region

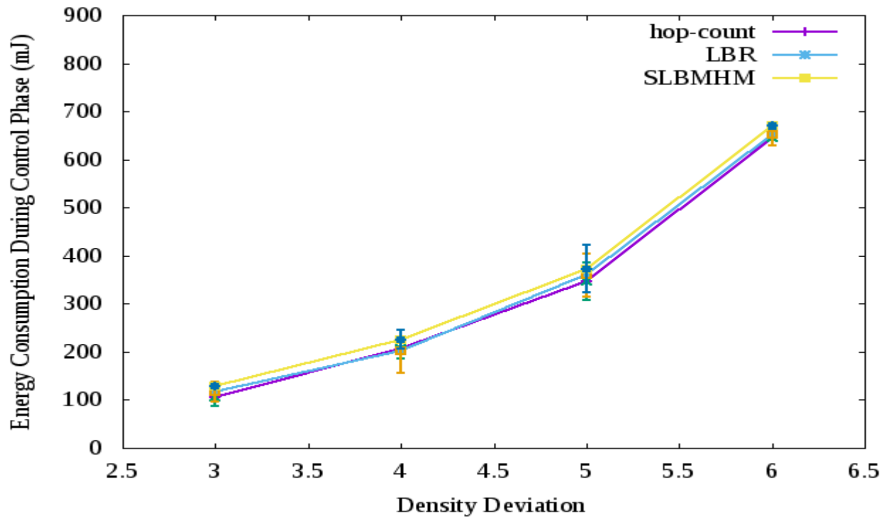


(b) Sinks are at diagonal corner in each sub-region

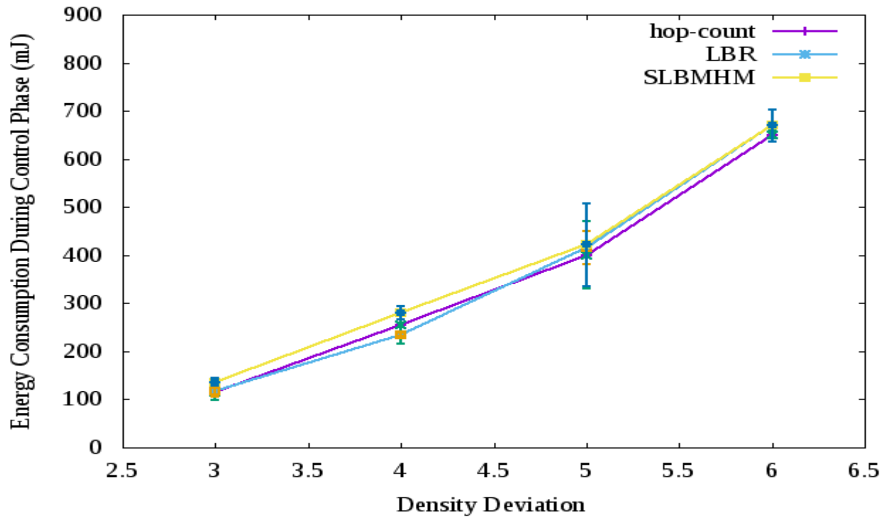


(c) Sinks are nearby

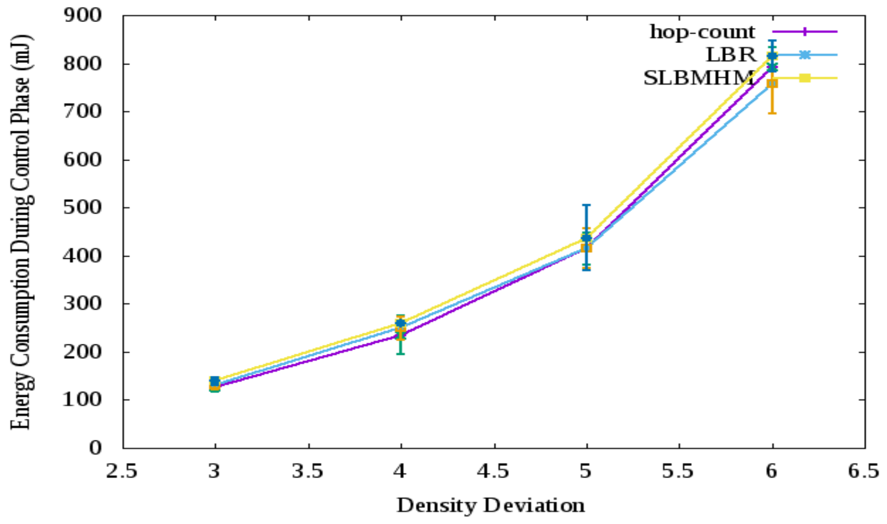
Figure 5.9: Dependency of Control Overhead on Density Deviation



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner in each sub-region



(c) Sinks are nearby

Figure 5.10: Dependency of Energy Consumption during Control Phase on Density Deviation

Sinks in Center		
Density Deviation	Avg improvement (%)	Std. deviation (%)
4	55%	35%
5	35%	17%
6	36%	6%
Sinks at Diagonal Corner		
Density Deviation	Avg improvement (%)	Std. deviation (%)
4	74%	28%
5	13%	1%
6	39%	11%
Sinks Nearby		
Density Deviation	Avg improvement (%)	Std. deviation (%)
4	47%	24%
5	52%	10%
6	40%	10%

Table 5.4: Percentage Improvement in Schedule Length Difference

Energy Consumption During Data Phase

The energy consumption during data phase in all three algorithms namely hop-count, LBR and SLBMHM remains almost same. The reason is that every node is configured to generate one packet at every 10 seconds. That is, all the nodes in the network are generating packets at the same rate. The neighborhood of every node remains same in all the three algorithms. So, number of children per node are also not likely to change. Because, all three algorithms use DICA[21] for scheduling & tree formation.

Thus all the nodes generate packets at the same interval. Moreover, in all the three algorithms, number of packets received by a node does not change. So, energy consumed during data phase does not vary across the algorithms.

As density deviation increases, the energy consumption is likely to increase in all the three algorithms.

5.4.4 Quantitative Comparison of Results

In the Tables 5.4 and 5.5 percentage improvement in schedule length difference and maximum schedule length respectively are summarized. To calculate improvement for a particular case, result of SLBMHM algorithm is compared with the second best performing algorithm for the given case.

It is observed from the Table 5.4 that the average improvement achieved by the

Sinks in Center		
Density Deviation	Avg improvement (%)	Std. deviation (%)
4	9%	5%
5	18%	10%
6	20%	1%
Sinks at Diagonal Corner		
Density Deviation	Avg improvement (%)	Std. deviation (%)
4	23%	7%
5	13%	6%
6	20%	7%
Sinks Nearby		
Density Deviation	Avg improvement (%)	Std. deviation (%)
4	14%	2%
5	14%	1%
6	24%	5%

Table 5.5: Percentage Improvement in Maximum Schedule Length

Sinks in Center	
Density Deviation	Avg increase (%)
3	20%
4	9%
5	7%
6	3%
Sinks at Diagonal Corner	
Density Deviation	Avg increase (%)
3	19%
4	10%
5	6%
6	4%
Sinks Nearby	
Density Deviation	Avg increase (%)
3	10%
4	11%
5	5%
6	3%

Table 5.6: Percentage Increase in Energy Consumption during Control Phase

SLBMHM algorithm ranges from minimum 13% to maximum 74%. In some cases, standard deviation is high. But still subtracting standard deviation from average results in a meaningful value. As seen from Table 5.5 that improvement in overall schedule length ranges from 9% to 24%. It means that in every TDMA cycle, schedule length is reduced by 9% to 24%.

It is seen from the Table 5.6 that maximum increase in control energy consumption is 20%. To calculate percentage increase in energy consumption for a particular case, result of SLBMHM algorithm is compared with the best performing algorithm for the given case.

The control phase does not take place very often. Tree formation and scheduling takes place first time after initial node deployment. Then after, whenever significant number of new nodes are added or existing nodes die, tree repairing and rescheduling is done. So, this additional energy consumption does not affect the network life time much. The reduction in schedule length and schedule length difference is achieved in each TDMA cycle. So, improvement in packet latency and fairness in transmission opportunity is ensured during each cycle. In short, this a trade-off between schedule length and energy consumption.

The SLBMHM algorithm does not achieve improvement in schedule length difference and maximum schedule length when density deviation is 3. But it results in 20% higher control energy consumption. When both the regions are equal in average density, the trees formed through SLBMHM algorithm are different in terms of density. So, their schedule lengths are very different and overall schedule length also remains higher. As the SLBMHM algorithm does not result in improvement when density deviation is 3, corresponding raws are not shown in Tables 5.4 and 5.5.

5.5 Summary

In this chapter, schedule length balancing for multi-sink homogeneous networks is presented. Many times distribution of nodes is not uniform in the entire region. So some trees are highly dense and others are sparse. The dense trees result in larger schedule length than the sparse ones. To reduce overall schedule length, it is required that schedule lengths should be balanced. Here, a heuristic approach of shifting the nodes from dense tree to sparse tree is suggested. The proposed approach is evaluated through simulations. It is observed from simulation results that proposed approach results in better schedule length balancing than existing algorithms in most of the cases. As a result, overall schedule length also decreases. Thus all the nodes are likely to wait for almost equal number of time slots to get turn to transmit.

Chapter 6

Schedule Length Balancing for Multi-sink HeTerogeneous networks (SLBMHT) Algorithm

In the previous chapter, schedule length balancing for multi-sink homogeneous networks is presented. It was assumed that all nodes are of the same type. But in practice, network may be heterogeneous. That is, more than one types of nodes may be present in the network. Earlier it is mentioned that the term ‘Attribute’ has the same meaning as type. Thus network may have multiple sinks and multiple attributes. In this chapter, Schedule Length Balancing algorithm for Multi sink HeTerogeneous networks (SLBMHT) is proposed. The chapter contains assumptions, objectives, steps of algorithm, correctness proofs and simulation results.

6.1 Motivation

Schedule length balancing for multi-sink homogeneous networks was presented in the previous chapter. The motivation was that when node distribution is not uniform, schedule lengths of sink-rooted trees are not balanced. Some method is required to balance the schedule lengths. This would also reduce the overall schedule length of network.

But it is also possible that network has more than one types of nodes present. So schedule length depends not only on average density and depth of the tree, but also on types of nodes present. It is shown in Chapter 4 that poor aggregation occurs when network is highly heterogeneous. Aggregation has direct impact on schedule length. If aggregation is poor, more packets come out of nodes. So more time-slots are required. Consider that two trees have same average density and depth. First tree has two types of nodes (temperature and pressure sensors) and second has four types of nodes (temperature, pressure, solar radiation and moisture sensors). It is likely that second tree would have larger schedule length than the first tree.

One of the steps in balancing algorithm is estimation of schedule length by sinks. Now schedule length is function of following three parameters.

1. Average density (σ)
2. Height of tree (h)
3. No. of attributes (γ)

Schedule length imbalance may be caused by non-uniform node distribution or network heterogeneity or both. We need an algorithm which attempts to balance the schedule lengths no matter what is the cause of imbalance. In other words, schedule length balancing algorithm should consider network heterogeneity also.

6.2 Problem Definition

In Chapter 3, problem definition is presented along with explanation of assumptions and objectives. It was mentioned that main problem is divided into two sub-problems : (i) Scheduling & tree formation in single sink heterogeneous networks. (ii) Schedule length balancing in multi-sink homogeneous networks. The sub-problems are addressed in Chapters 4 and 5 respectively. In this chapter, we attempt to solve the main problem i.e. one defined in Chapter 3.

Since problem is already defined in detail in Chapter 3, the explanation is not repeated here entirely. But problem statement, assumptions and objectives are stated for the sake of completeness.

6.3 Problem Statement

To design a distributed algorithm to balance schedule lengths of sink-rooted trees in multi-sink heterogeneous sensor networks.

6.3.1 Assumptions

1. One or more sinks are deployed.
2. More than one attributes may be present in the network.
3. Node distribution is not uniform.
4. Every sink is root of exactly one tree.
5. Every node joins exactly one tree. In other words, trees are disjoint.
6. Every packet requires one time-slot.

6.3.2 Objectives

1. Overall schedule length (SH) must be minimized. It is defined in equation 3.1.
2. Difference between schedule lengths of trees should be minimal. It is denoted as SH_{diff}. It is defined in equation 3.2
3. Control Overhead (CO) should be minimal. It is defined in equation 3.3.
4. Average Energy Consumption during Control Phase should be minimal. It is denoted as E^C. It is defined in 3.5.
5. Minimize Average Energy Consumption during Data Phase. It is denoted as E^D. It is defined in 3.7.

6.4 SLBMHT Algorithm

6.4.1 Illustration of the Algorithm with an Example

In Figure 6.1, a sample topology is shown. Nodes 0 to 49 are in region 1. Nodes 50 to 98 are in region 2. Assume that there are two sinks: S_1 is node 24 (center node in region

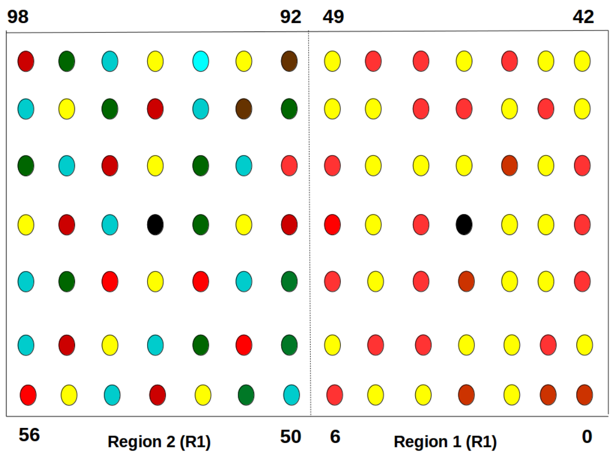


Figure 6.1: Illustration of Node Deployment for SLBMHT Algorithm)

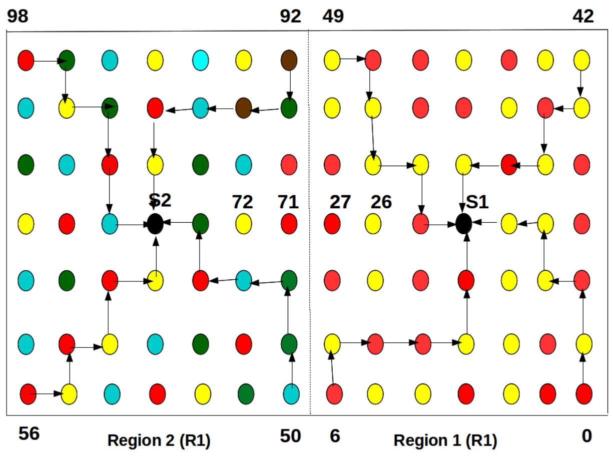


Figure 6.2: Illustration of Formation of Temporary Trees in SLBMHT Algorithm

1, filled with black color), S_2 is node 74 (center node in region 2, filled with black color). Two types of nodes are present in region 1. Nodes of type 1 and type 2 are filled with red and yellow color respectively. Four types of nodes are present in region 2. Nodes of type 1, 2, 3 and 4 are filled with red, yellow, cyan and green colors respectively.

At the beginning of algorithm, S_1 and S_2 will turn by turn flood HELLO packets in the network. At the end, every node will know its hop distances from both S_1 and S_2. Every node will select the nearest sink (in terms of hop count) as its home sink. Nodes 0 to 49 will join S_1. Let us call it Group 1. Nodes 50 to 98 will join S_2. Let us call it Group 2.

Every node in Group 1 and Group 2 will select one node as temporary parent. In Figure 6.2, formation of temporary trees is shown. All edges are not shown in figure. Some sample edges are shown to provide better visualization. Now every leaf node in Group 1 and Group 2 will calculate its neighbor count, depth and also identify its type. Neighbor count, depth and type are sent as part of JOIN message to temporary parent.

Each node in path will wait for its temporary children to send JOIN message. Once a node receives JOIN messages from all its temporary children, it will add neighbor counts received with its own neighbor count. Every non-leaf node maintains a list '*types*'. If sub-tree rooted at given node has at least one node of type i, node will set $types[i]$ to 1. Else $types[i]$ is 0. Every non-leaf node will send JOIN message to its parent. The message will contain total neighbor count, maximum of height values received from children and list *types*. Finally S_1 and S_2 both receive JOIN messages from their temporary children. Each sink will calculate values of average density, height and type count for respective tentative trees. Here type count means how many different types of nodes are present in the tree.

Sink nodes will estimate the schedule lengths of their tentative trees based on average density, height and type count of the tree. As region 1 has two types of nodes and region 2 has four types of nodes, schedule length of tree rooted at S_1 is likely to be less than that of tree rooted at S_2. In scenario of Figure 5.2, schedule lengths were unbalanced due to difference in density. Here, schedule lengths are unbalanced due to difference in heterogeneity. Following paragraphs explain balancing method with respect to Figure 6.2.

S_1 will flood a message BAL_NOT_REQD in its tree. S_2 will flood BAL_REQD message in its tree. S_2 estimates the maximum level h^{bal} of its tree to have balanced schedule length. It will mention the same in the BAL_REQD message. Nodes at level $h^{bal}+1$ or higher in tree rooted at S_2 should attempt to shift to tree rooted at S_1. Other nodes would not shift.

Every node in Group 1 will broadcast SINK_CONFIRM message to tell its neighbors that it is stick to the old sink. Nodes at the boundary of two subregions (i.e. node 50,57,64,71,78,85,92) are the first ones to hear SINK_CONFIRM message from nodes in Group 1. These nodes will try to switch to S_1. Each one of them will estimate the possible schedule length of S_1 if it joins the sink. If estimated schedule length is less than or equal to balanced schedule length, node would switch to S_1. Else it would stick to the old sink. If a node decides to change the sink, it broadcasts a SINK_MODIFIED message to inform its new home sink to its neighbors. The message contains three fields: ID of new home sink (i.e. S_1), estimated new schedule length of S_1 and flag *sink_changed.* If node has changed the sink, it will set ID of new home sink to 24 (i.e. S_1) and flag *sink_changed* would be set to 1.

For example, let us assume that node 71 switches to S_1. It will estimate the new schedule length of S_1 considering that itself joins S_1. Consider that original schedule length of S_1 is 10. Node 71 is four hops away from S_1. In neighborhood of 71, no node is at distance four from S_1. Node 71 is of type 1. Towards S_1, the nearest node of type 1 is node 27. Node 27 is one hop away from node 71. So packet generated by 71 will be aggregated at node 27. So the packet will not consume any additional slots during its journey to S_1. So if node 71 joins the tree of S_1, new schedule length of S_1 will be 10 (original schedule length of tree rooted at S_1) + 0 (no. of neighbors of node 71 switched to S_1 and at the same hop distance as node 71 from S_1) + 1 (distance from node 71 to nearest node of type 1 towards S_1) = 11. If estimated schedule length (i.e. 11) is less than balanced schedule length, node 71 would decide to switch to S_1. It would broadcast In SINK_MODIFIED message with *sink_changed* set to 1. The message will contain new estimated schedule length of S_1 i.e. 11.

Consider node 72. It is five hops away from S_1. It receives SINK_MODIFIED message from 71. Node 71 is four hops away from S_1. Node 72 knows that if 71 joins S_1, the resulting schedule length is likely to be 11. Assume that there are two nodes in neighborhood of node 72 which are at five hop distance from S_1 and have switched to S_1

(these nodes are not shown in figure). Node 72 is of type 2. The nearest node of type 2 towards sink 1 is node 26. It is three hops away from node 72. Node 72 estimates new schedule length of S_1 as 11 (as received from 71) + 2 (two neighbors at the same level as itself from S_1 who have switched to S_1) + 3 (distance from node 12) = 16. If this estimated value is less than balanced schedule length, node 72 will switch to S_1. Else it will stick to S_2. The last term in sum indicates that packet generated by node 72 will travel three hops before it gets aggregated. Thus three extra slots should be added in the schedule length of S_1.

In this example, node 72 is at hop distance five from S_1. In its neighborhood, there is only one node (node 71) at distance four from S_1. There may be multiple nodes at level four in neighborhood of node 72. So node 72 may receive multiple SINK_MODIFIED messages. In that case, it will use maximum of received values of schedule estimate in its calculation (i.e. first term in sum) of new schedule of S_1.

Switching process would start from border of two regions. It would progress from border to left i.e. towards S_2. Slowly nodes in left side would switch to S_1. As we move towards left boundary of region 2, chances of node switching to S_1 gets reduced. Because, as distance from S_1 increases, the estimated schedule length (if node switches to S_1) increases. When estimated schedule length starts approaching balanced schedule length, switching process stops. Thus it is ensured that schedule length of S_2 decreases, but that of S_1 does not cross balanced schedule length. Once switching process is complete, scheduling & tree formation proposed in AAJST would be executed.

6.4.2 Flow Diagram of the Algorithm

The proposed algorithm is explained through flow diagrams in Figures 6.3 and 6.4. The figures are self-explanatory. First the steps of Figure 6.3 are executed. Then steps mentioned in Figure 6.4 are executed. Details are not mentioned in flow diagram. But it helps the reader to understand overall approach.

6.4.3 Steps of the Algorithm

In this subsection, steps of SLBMHT algorithm are explained. Working of SLBMHT algorithm is similar to SLBMHM algorithm discussed in Chapter 5. Only the steps

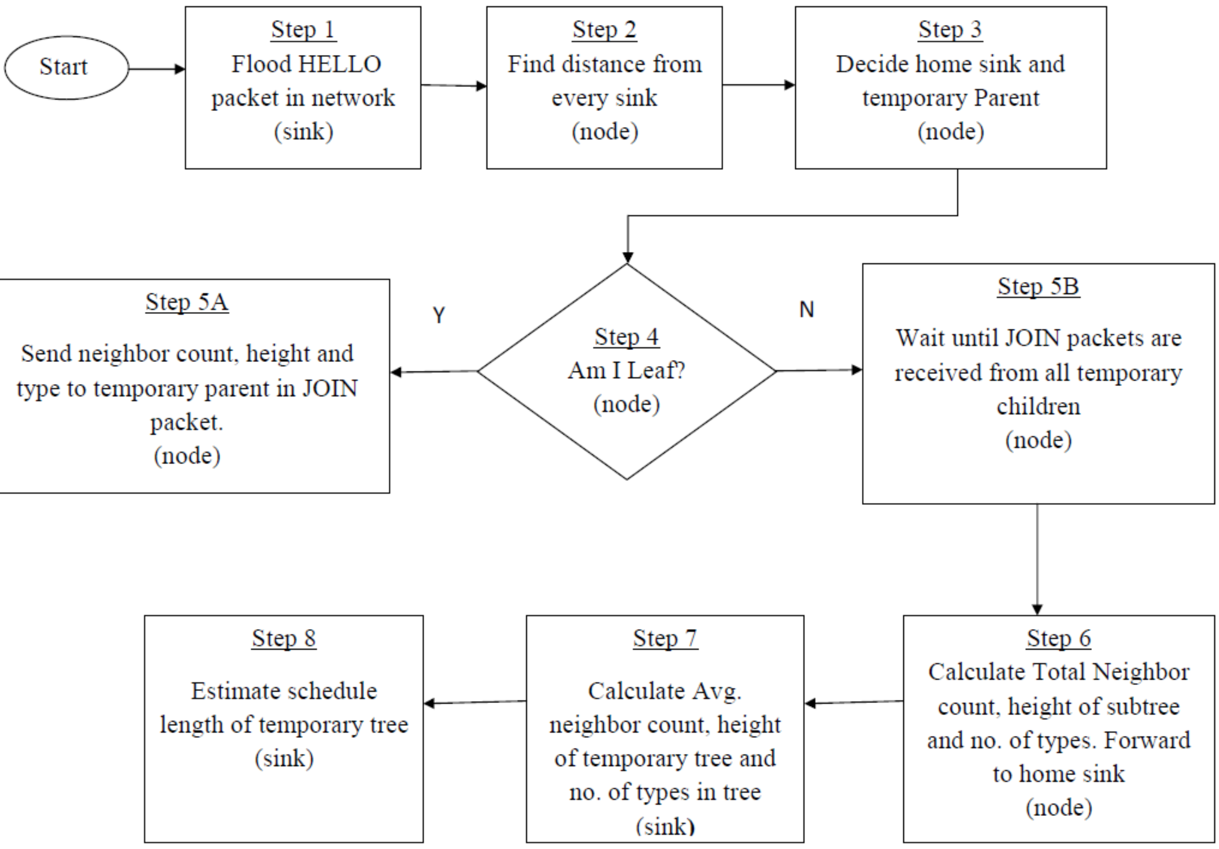

Start
Step 1
Flood HELLO packet in network (sink)
Step 2
Find distance from every sink (node)
Step 3
Decide home sink and temporary Parent (node)
Step 4
Am I Leaf? (node)
Y
N
Step 5A
Send neighbor count, height and type to temporary parent in JOIN packet. (node)
Step 5B
Wait until JOIN packets are received from all temporary children (node)
Step 6
Calculate Total Neighbor count, height of subtree and no. of types. Forward to home sink (node)
Step 7
Calculate Avg. neighbor count, height of temporary tree and no. of types in tree (sink)
Step 8
Estimate schedule length of temporary tree (sink)

Figure 6.3: Flow Diagram of SLBMHT Algorithm - Part I

Parameter	Meaning
nbc	No. of neighbors
t_i^{nbr}	Type of neighbor node i
$< t^{nbr} >$	Vector storing types of neighbors
$types$	It is a vector present in JOIN message.
γ_i	No. of attributes present in tree rooted at sink S_i

Table 6.1: Additional Notations used in SLBMHT algorithm

different than SLBMHM are explained in detail. Notations used in explanation are same as mentioned in Table 5.1. In Table 6.1, additional notations introduced in SLBMHT are explained.

Steps of proposed algorithm are explained below.

1. Flooding of HELLO packets and leveling of nodes:

   The step is same as in SLBMHM except that now HELLO packet also contains a field ‘type’. Types are numbered. For example, temperature is numbered 1, pressure is numbered 2, and so on. The node which broadcasts HELLO packet sets

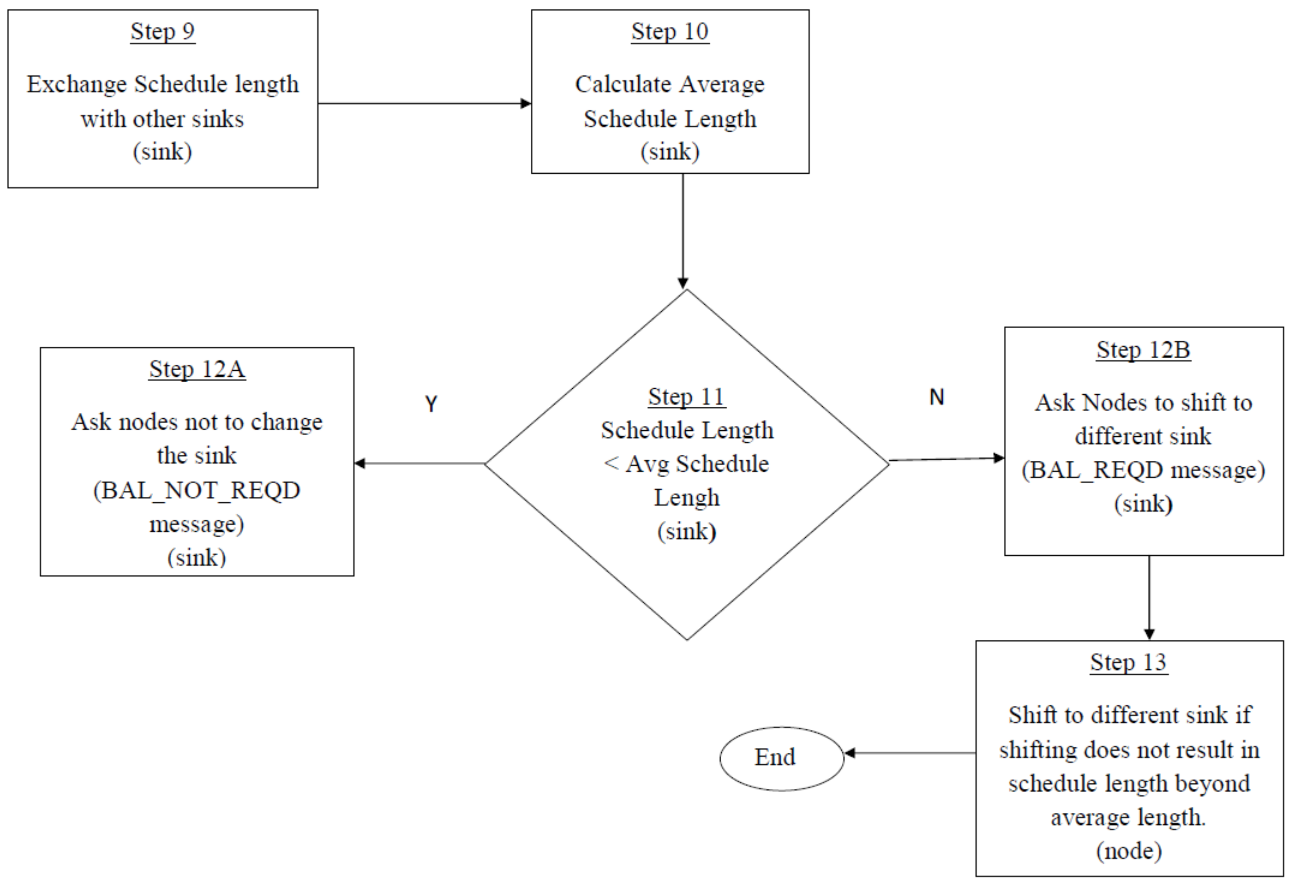

Step 9
Exchange Schedule length with other sinks (sink)
Step 10
Calculate Average Schedule Length (sink)
Step 11
Schedule Length < Avg Schedule Lengh (sink)
Y
N
Step 12A
Ask nodes not to change the sink (BAL_NOT_REQD message) (sink)
Step 12B
Ask Nodes to shift to different sink (BAL_REQD message) (sink)
Step 13
Shift to different sink if shifting does not result in schedule length beyond average length. (node)
End

Figure 6.4: Flow Diagram of SLBMHT Algorithm - Part II

type to appropriate number representing its type. This helps every node to know type for each neighbor.

Now every node maintains an additional vector $< t^{nbr} > = t_1^{nbr}, t_2^{nbr}, \ldots\ldots, t_{nbc}^{nbr}$.

2. Formation of temporary trees:

   The step is same as in SLBMHM.

3. Estimation of schedule length by every sink: Following steps are executed so that every sink could estimate schedule length of its temporary tree.

   (a) Every leaf node will send a JOIN message to temporary parent. Format of JOIN message is same as SLBMHM except that now it contains ‘type’ field. Every leaf node writes number corresponding to its type in JOIN.

   (b) Every non-leaf node at level h will wait to hear JOIN messages from its neighbors at level $(h + 1)$. Once it hears JOIN messages from all neighbors at level

$(h+1)$, it sends a JOIN message to its temporary parent.

Information present in JOIN is same as that in SLBMHM except addition of new vector '*types*'. JOIN message sent by non-leaf node contains following information. If a node of type j is present in tentative sub-tree rooted at given node, $types[j]$ is set to 1. Else it is 0.

(c) At the end of above step, every sink S_i receives JOIN messages from its temporary children. Every sink is able to calculate the following parameters about its temporary tree.

   i. Height (h_i) of the tree.

   ii. Average density (σ_i). It is ratio of total neighbor count and total node count.

   iii. No. of attributes present (γ_i). If region has four types of nodes present, $\gamma_i = 4$.

   iv. Estimated Schedule length (SH_i^{est}). It is the estimate of schedule length of temporary tree rooted at S_i. It is function of σ_i, h_i and γ_i. It is derived by putting values of σ_i, h_i and γ_i in Equation 6.6. Detailed explanation about schedule length estimation is given later in the chapter.

(d) Sinks exchange estimated schedule lengths and find average schedule length. It is referred as 'balanced schedule length', denoted as SH_{bal}.

4. Schedule length balancing: Following steps are performed for schedule length balancing.

   (a) If sink S_i finds that SH_i^{est} is greater than SH_{bal}, it attempts to remove some nodes from its tree. It sends BAL_REQD message in its tree to inform the nodes in its tree that balancing is required. Else it sends BAL_NOT_REQD message in the tree.

   (b) As part of BAL_REQD message, sink S_i sends following parameters:

      i. Estimated schedule lengths SH_1^{est},SH_2^{est},....,SH_N^{est}.

      ii. Balanced schedule length (SH_{bal}).

iii. Required height (h_i^{bal}) of the tree rooted at sink S_i to achieve balanced schedule length. The value of h_i^{bal} is calculated by putting SH_{bal}, average density σ_i and no. of attributes γ_i of the tree in Equation 6.6. Sink indicates that all the nodes in its tree at a level greater than h_i^{bal} should attempt to switch to a different tree.

(c) If a node whose temp_home_sink is S_i receives BAL_NOT_REQD message from S_i, it confirms attachment to sink S_i by broadcasting SINK_CONFIRM message.

(d) If a node whose temp_home_sink is S_i receives BAL_REQD message from S_i, it will broadcast SINK_MODIFIED message with sink_changed flag set to 0 if d_i is less than or equal to h_i^{bal}.

(e) In the above case, if d_i is greater than h_i^{bal}, following steps are executed to change home sink.

i. Wait for neighbors belonging to sinks having schedule length less than that of S_i to finalize their sinks either by deciding to stick with same sink or changing to new sink. Then following steps are performed.

ii. Create a set of target sinks. A sink S_j is member of the set if following two conditions are satisfied: (i) Schedule length of S_j is less than balanced schedule length. (ii) At least one node is present in neighborhood of given node which belongs to S_j and is nearer to S_j compared to given node. If the set is empty, node keeps waiting. When it overhears a SINK_MODIFIED message (described later), it tries to create the set again. If set is non-empty, following steps are performed.

iii. Assume that Z sinks are present in the set of target sinks. The set is denoted at tgt_sinks. If sink S_j is present in set tgt_sinks, its current schedule length is denoted SH_j^{cest}.

- SH_j^{cest} is same as SH_j^{est} if node is one hop away from a node whose temp_home_sink is S_j.
- Otherwise SH_j^{cest} is set as follows. Given node is more than one hops away from node(s) whose temp_home_sink is S_j. But still S_j is in set Z. It means some neighbor has switched to S_j. When a node

switches to new sink, it broadcasts SINK_MODIFIED message with sink_changed flag set to 1. In addition, it also writes new estimated value of SH_j^{cest} in the message. Given node may overhear many such messages. It sets SH_j^{cest} to the maximum of received values of SH_j^{cest}.

iv. Node estimates new schedule length of every sink S_j in set tgt_sinks considering that it would switch to the sink. Number of neighbors who belong or switched to sink S_j be $nbr\_switched_j$. Then new value of SH_j^{est} is estimated as follows:

$$SH_j^{cest} = SH_j^{cest} + nbr\_switched_j + aggr\_dist \tag{6.1}$$

Above equation is similar to equation 5.1 except that here last term is aggr_dist, not '1'. Network is heterogeneous. So given node should also consider the distance at which its packet will get aggregated. Suppose *aggr_dist* is 4. It means that the nearest node of same type as given node is at hop distance 4. So packet generated by given node can not be aggregated before 4 hops. At each hop, packet would consume one slot. Thus 4 slots are added in the schedule length if given node switches to tree of S_j. In homogeneous network, packet would be aggregated at next hop itself. So, *aggr_dist* was 1 in equation 5.1.

v. Node updates SH_j^{cest} for each sink S_j in set *tgt_sinks*. It decides to shift to sink S_k whose SH_k^{cest} minimum and is less than SH_{bal}. If no such sink is found, node sticks to current sink and broadcasts SINK_MODIFIED message with flag sink_changed set to 0.

vi. When node decides to switch to sink S_j, it broadcasts SINK_MODIFIED message. The message has *sink_changed* flag set to 1. The message notifies the neighbors about node's decision. The message also contains latest value of SH_j^{cest} so that neighbors could update their estimates of SH_j^{cest}.

5. Once every node finalizes the sink, scheduling and tree formation algorithm AAJST as proposed in Chapter 4 is executed. Every node selects a slot and parent. Thus at the end, N different trees are formed. Every tree is rooted at one sink.

6.4.4 Correctness of the Algorithm

Lemma 6.4.1. *In heterogeneous network, schedule length (SH) of a tree depends on it's avg. density (neighbor count, σ), height (h) and count of types of nodes (γ) .*

Proof. This lemma is similar to Lemma 5.3.1. So most of the explanation is repeated. Explanation related to average density (σ) and height (h) is same as given in Lemma 5.3.1. But it is presented here for the sake of completeness.

For collision-free schedule formation, it is required that transmission slot selected by given node should be such that it does not create interference at neighboring nodes. Suppose number of neighbors of node n is n_c. Each one of n_c neighbors is receiving in certain slot. Thus node n can not transmit in those n_c slots. The slot selected by n should be different than those n_c slots. If network is dense, value of n_c would be high. If network is sparse, n_c would be low. Thus in a dense network, number of slots consumed are more compared to sparse network.

Slot assignment should be bottom-up in aggregated convergecast. That is, slot assignment should progress from leaf to root. Leaf node(s) should be assigned lowest timeslot. Time slots increase from leaf to root. This criteria is essential for aggregation freshness i.e. parent could aggregate children's packets in the same TDMA cycle and transmit further. Otherwise parent would receive packets from children in one TDMA cycle and could forward in next TDMA cycle. If tree height is more, leaf is far from root. So more slots are required to reach the root. This in turn means that if height increases, schedule length increases.

In raw convergecast, scheduling need not be bottom to top. But every node has to forward all the packets coming from children node. If leaf node is far from root, its packet has to travel longer path to root. At each hop, packet needs one time-slot. So as distance from root increases, number of required time slots also increases.

In heterogeneous networks, more than one types of nodes are present. It is explained in Chapter 4 that quality of aggregation deteriorates with increase in heterogeneity. If aggregation becomes poor, number of packets forwarded by nodes increases. Thus number of slots required to transmit (and also to receive) those packets increases. In short, schedule length increases. In other words, two trees may have same density and depth. But if their degree of heterogeneity is different, their schedule lengths would be different.

In case of raw convergecast, every node sends out all packets coming from children nodes. So change in degree of heterogeneity has no special effect on schedule length in raw convergecast.

Thus from above discussion it is clear that schedule length in aggregated convergecast depends on average density, tree height and degree of heterogeneity. But in raw convergecast, important factors are average density and height of the tree.

□

Lemma 6.4.2. *SLBMHT algorithm ensures that every node has at least one path to the sink.*

Proof. Proof is same as Lemma 5.3.2. □

Lemma 6.4.3. *SLBMHT algorithm reduces schedule length of overloaded sink(s) without increasing schedule length of underloaded sink(s) beyond average.*

Proof. Proof is same as Lemma 5.3.3. □

Lemma 6.4.4. *If schedule lengths of trees are balanced, maximum schedule length of the network is also reduced. Thus average number of slots before a node gets its turn to transmit also gets balanced.*

Proof. Proof is same as Lemma 5.3.4. □

6.4.5 Schedule length (SH) as function of Average Density (σ), height (h) and No. of Attributes (γ)

It is mentioned in proposed algorithm that every sink estimates schedule length (SH) of its temporary tree based on following parameters: average density (σ) of the tree, height (h) of the tree and number of attributes (γ) present in the tree. To establish relationship between SH, σ, h and γ, a simulation based study is carried out as follows.

Single sink network is considered. Different combinations of density, depth and heterogeneity are tried. Following values of densities are used: $\sigma_1 = 4$, $\sigma_2 = 8$, $\sigma_3 = 12$ and $\sigma_4 = 20$. For each σ_i, height (h) is varied from 2 to 20. For each combination of σ and h, γ is varied between 1,2,4 and 6. If γ is 4, it means that four different types of nodes are present.

Graphs of Schedule length v/s. Depth are plotted for fixed σ and γ = 1,2,4 and 6. Thus on a single XY plane, there are four graphs (one for each value of γ). For given σ and γ, schedule length linearly increases with h. It can be expressed as follows:

$$SH = m * h \tag{6.2}$$

In above equation, m is slope of line. As SH is function of σ,γ and h, m should be replaced by some formula containing σ and γ. It is achieved through following steps.

As the first step, graphs of Slope (m) v/s. No. of Attributes (γ) are plotted for different values of σ. It is found that m linearly increases with γ. Following is the general equation for m:

$$m = m_1 * \gamma + c \tag{6.3}$$

Here m_1 is slope of line. Value of c represents the point where line crosses Y axis. Both m_1 and c are linear functions of σ. They are as follows:

$$m_1 = 0.2 * \sigma - 0.13 \tag{6.4}$$

$$c = 0.45 * \sigma - 0.8 \tag{6.5}$$

Using equations 6.4 and 6.5 in equation 6.3, equation for m would be found. Replacing m in equation 6.2, an equation of following form is expected.

$$SH = p * \sigma * \gamma * h - q * \gamma * h + r * \sigma * h - v * h \tag{6.6}$$

In equation 6.6 above, values of p,q,r and v are 0.2,0.13,0.45 and 0.8 respectively. The values of σ used are from 4 to 20,height h is between 2 to 20 and γ is between 1 to 6.

Every sink knows average density (σ), height (h) and number of attributes (γ) for its tentative tree. So above equation can be used by a sink to estimate schedule length of its tree.

6.5 Simulation Results

In this section, simulation results are discussed. Details of simulation setup, other protocols used for comparison and different performance parameters are also discussed.

6.5.1 Simulation Design

Different sink deployments are illustrated in Figures 5.5a, 5.5b and 5.5c. The same deployments are used here also. There are two sinks: S_1 and S_2. The sink deployments are listed below:

1. Sinks are in center of each sub-region (Figure 5.5a).
2. Sinks are at opposite diagonal corners (Figure 5.5b).
3. Sinks are near to each other (Figure 5.5c).

The three cases differ only in sink placements. Else they are same. Common details are presented next. A square region of 400 meters x 200 meters is used for simulation. It is divided into a grid of 40 x 20 points. Distance between any two horizontal or vertical grid points is 10 meters. The whole region is divided into two sub-regions. Each sub-region is of size 200 meters x 200 meters. Every region contains one sink.

In region 1, probability of node being deployed at a grid point is 0.3. The probability of being deployed at a grid point in region 2 is 0.5. Let us denote number of different attributes present in region i by N_i^A. Here N_1^A is 2. That is, two different types of nodes are deployed in region 1. From one simulation run to other, N_2^A is varied between 2,4,6 and 8.

Probability (p_k^j) that a node j is present in region i is assigned type k is defined as follows:

$$p_k^j = \frac{1}{N_i^A}, k = 1, 2, 3, ..., N_i^A \tag{6.7}$$

Table 6.2 summarizes attribute assignment in both the regions. The four rows of Table 6.2 present four different scenarios. In the first scenario both the regions have same number of attributes (i.e. 2). But gradually degree of heterogeneity in region 2 is increased from 2 to 8. As heterogeneity increases, schedule length of region 2 will increase.

Scenario	N_1^A	N_1^A	H_d
1	2	2	1
2	2	4	2
3	2	6	3
4	2	8	4

Table 6.2: Heterogeneity in two regions

Node distribution remains same across all scenarios. The last column is Heterogeneity Difference (H_d). It is defined as follows:

$$H_d = \frac{N_2^A}{N_1^A} \tag{6.8}$$

As mentioned in above equation, Heterogeneity Difference is the ratio of no. of attributes in region 2 and region 1. It is natural that as degree of heterogeneity of region 2 increases, heterogeneity difference increases.

The SLBMHT algorithm is evaluated with respect to different performance parameters. These parameters are mentioned in next sub-section. But for every parameter, performance is evaluated against variation in H_d. So, X-axis is labeled as Heterogeneity Difference (H_d) in all graphs of simulation results.

There are three different sink deployments. For each deployment, there are four different attribute distribution scenarios as mentioned in Table 6.2. Node distribution is same in all scenarios as mentioned earlier. Performance of SLBMHT is evaluated for each combination of sink deployment and attribute distribution scenario. For a specific sink deployment, four different instances of each attribute distribution scenario are randomly generated. So, each point in graph is an average of four different simulation runs. Corresponding standard deviation is also calculated and plotted as error bar.

6.5.2 Simulation Setup

Simulation setup is same as mentioned in Table 5.3. So details are not repeated here.

6.5.3 Discussion of Results

We have compared proposed algorithm with different existing algorithms. They are same as those used for evaluating SLBMHM. For the sake of completeness, list of algorithms used for comparison is given below.

1. Hop count based approach.

2. LBR (Load Balanced Routing)[34].

3. SMTLB (Spanning Multi Tree Load Balanced routing) [38].

The hop count based approach and LBR use DICA_EXTENSION (mentioned in Chapter 4) for slot and parent selection. In SMTLB algorithm, single parent is selected. But as given node may have multiple outgoing packets, multiple slots are selected to transmit to the same parent. The SLBMHT use AAJST (as proposed in Chapter 4) for scheduling & tree formation.

Performance Measures

The performance measures are same as used for multi-sink homogeneous networks. So, they are not explained in detail but listed below for the sake of completeness. Number of nodes is denoted by n.

1. Maximum Schedule Length (SH) : It is defined in equation 5.13.

2. Difference in Schedule Length (SH_{frac}) : It is defined in equation 5.14.

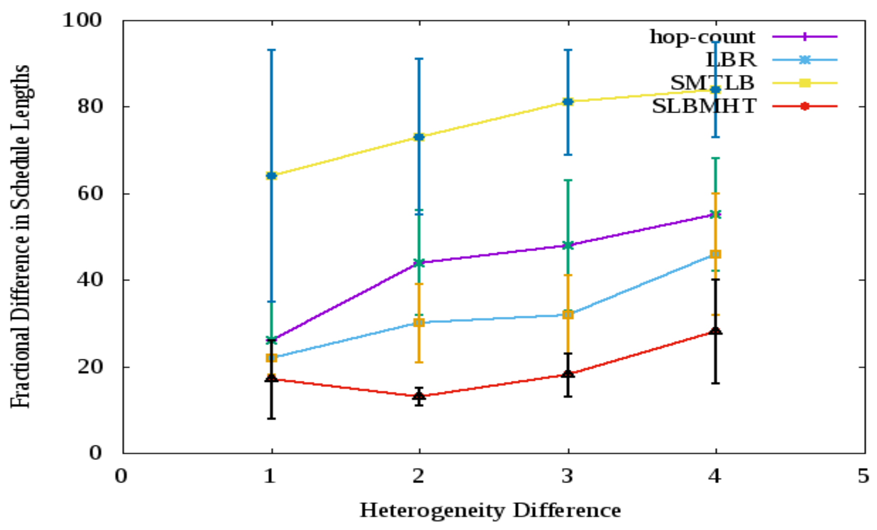
3. Control Overhead (CO): It is as defined in equation 3.3.

4. Energy Consumption During Control Phase (E_C) : It is as defined in equation 3.5.

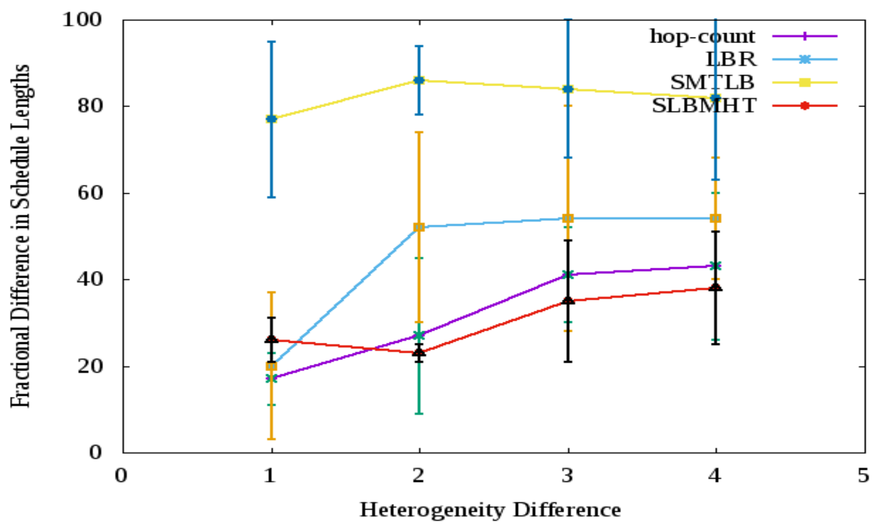
5. Energy Consumption During Data Phase (E_D) :It is as defined in equation 3.7.

As explained in simulation design, the network has two regions. Each region has one sink. Let us denote the tree rooted at sink S_1 as T_1. Tree rooted at sink S_2 be denoted as T_2. The schedule lengths of T_1 and T_2 be SH_1 and SH_2. Maximum schedule length of the entire network be SH. As defined earlier also, $SH = \max(SH_1,SH_2)$.

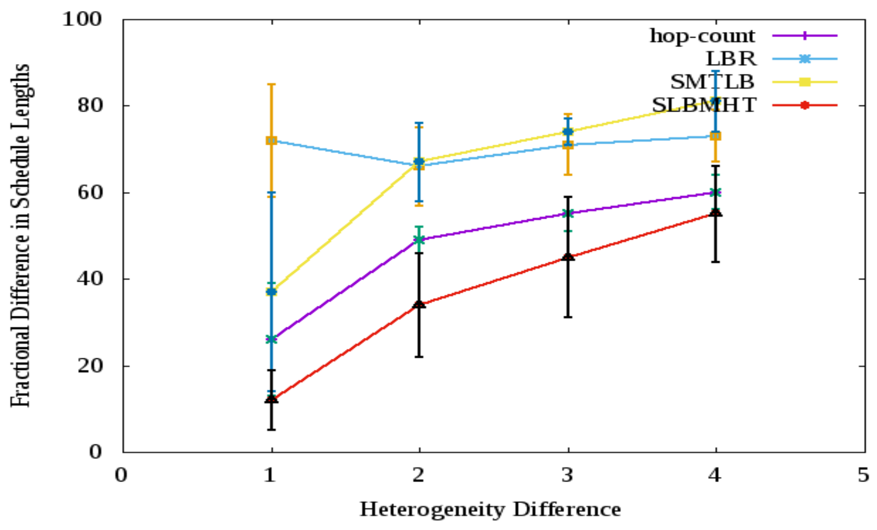
Like in Chapter 5, SMTLB algorithm is implemented as an application program. So, results for total transmission slots, control overhead, energy consumption during control phase and data phase are not generated for SMTLB.



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner in each sub-region



(c) Sinks are nearby

Figure 6.5: Dependency of Fractional difference in Schedule Lengths on Heterogeneity Difference

Fractional Schedule Length Difference and Maximum Schedule Length

In Figures 6.5a,6.5b and 6.5c, the graphs of Fractional difference in Schedule lengths v/s. Heterogeneity Difference for three cases namely sinks in center, sinks at diagonal corner and sinks near to each other are shown. It is seen that as Heterogeneity Difference increases, difference between SH_1 and SH_2 increases. As explained earlier, region 1 has two different types of nodes but types of nodes in region 2 is varied between 2,4,6 and 8. Thus increase in Heterogeneity Difference means degree of heterogeneity increases in region 2. It is shown in Chapter 4 that as heterogeneity increases, aggregation becomes poor. Thus nodes require more transmission slots. As a result, schedule length increases. As heterogeneity of region 2 is increases, SH_2 increases. The degree of heterogeneity of region 1 is not varied. So, SH_1 does not vary with heterogeneity difference. Thus difference between SH_1 and SH_2 increases.

It is also seen that SLBMHT algorithm results in least schedule length difference compared to other approaches. The SLBMHT algorithm is aimed at balancing schedule lengths of both the trees. In this case, SH_2 is higher than SH_1. So nodes belonging to tree T_2 shift to the tree T_1. So schedule length of tree T_1 goes up and that of T_2 goes down. But the resulting values are near to average of original schedule lengths.

As explained earlier, in SMTLB algorithm, subtrees grow gradually by expanding the least loaded subtree. Number of sinks are two. But number of sub-trees are likely to be more than two because every sink may have number of neighbors. Each such neighbor is root of one sub-tree. SMTLB works in top-down manner. That is, parent selection proceeds from sink to leaf nodes. SMTLB is not designed for heterogeneous networks. Unlike in heterogeneous networks, every node just selects a single parent and multiple slots instead of multiple slot/parent pairs. All packets are sent via the same parent. As a result, aggregation remains poor. So, more time slots are required. Thus SMTLB is not taking heterogeneity into account. So trees T_1 and T_2 both result in higher schedule lengths and their schedule lengths are also not balanced.

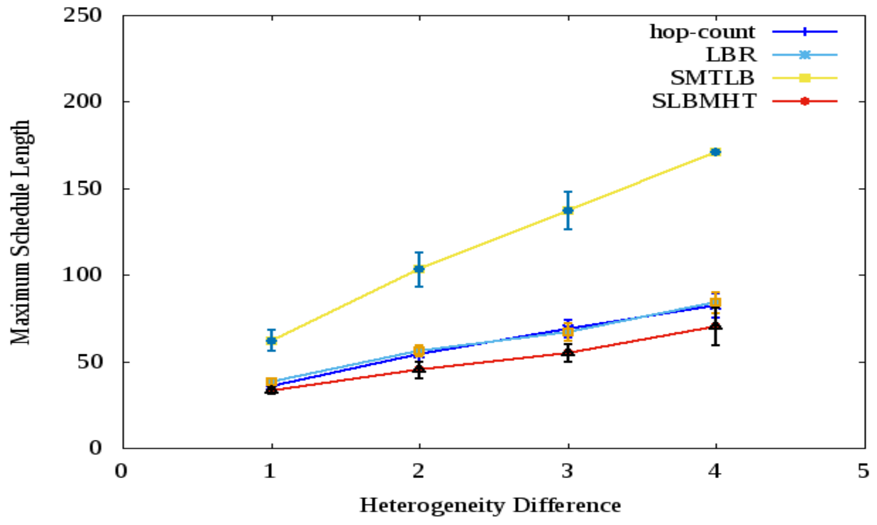
In LBR algorithm, parent selection is done based on ratio of neighbor count of sink and hop distance from sink. A node joins the sink with the highest ratio. The idea is to prefer the sink with more number of neighbors so that funneling effect [1] can be avoided. When two regions are different in density, nodes would prefer to join the sink present in dense region. Only those nodes which are far away from that sink would prefer to join the other sink.

In our simulation setup, region 2 has slightly higher density compared to region 1. So nodes of region 2 would join the tree rooted at S_2. Even the nodes of region 1 for whom ratio of neighbor count of sink and hop distance from sink remains higher for sink 2 will also join tree rooted at sink 2. Thus more nodes would join T_2. In addition, region 2 has four different types of nodes present. So, SH_2 will remain higher than SH_1. Thus LBR algorithm is not able to balance the schedule lengths.

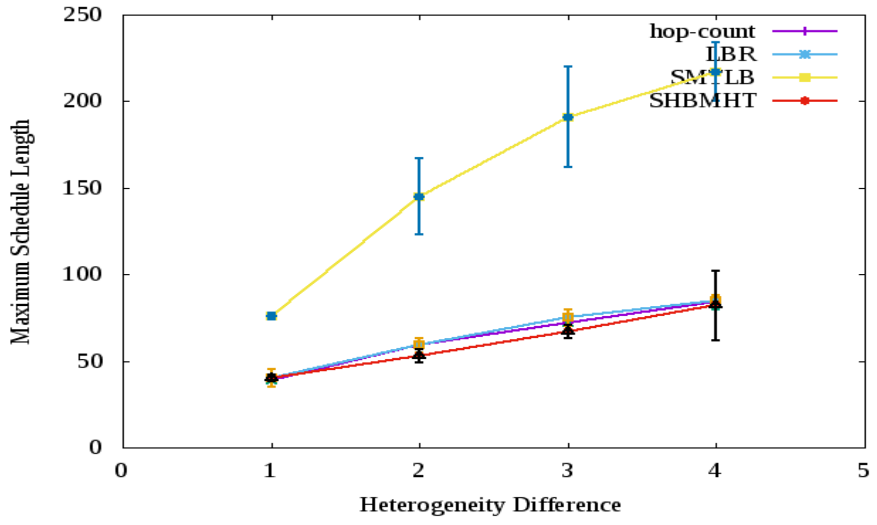
In hop count based approach, every node joins the sink which is at the least hop distance. When network is dense, more number of nodes are present in a given radius compared to a sparse network. As hop distance from sink is the only criteria to select the tree, nodes in dense part of network join the nearest sink S_2 and the resulting tree has large number nodes. Nodes in sparse region of network join the nearest sink S_1 resulting in a small tree. Here not only region 2 has more number of nodes compared to region 1 but it has more types of nodes. So, again SH_2 remains higher than SH_1. So, hop count based approach also is not able to balance the schedule lengths.

In Figures 6.6a,6.6b and 6.6c, the graphs of Max. Schedule Length v/s. Heterogeneity Difference for three cases namely sinks in center, sinks at diagonal corner and sinks near to each other are shown. As mentioned earlier, max. schedule length SH is max(SH_1, SH_2). SLBMHT algorithm is able to balance SH_1 and SH_2 by reducing SH_2 and increasing SH_1 such that SH_1 does not go beyond SH_{bal}. As a result, max(SH_1, SH_2) goes down and so SH. As other approaches are not able to balance SH_1 and SH_2, they are expected to result in higher max. schedule length compared to SLBMHT algorithm. As difference in schedule lengths increases with increase in heterogeneity difference, max. schedule length also increases with heterogeneity difference.

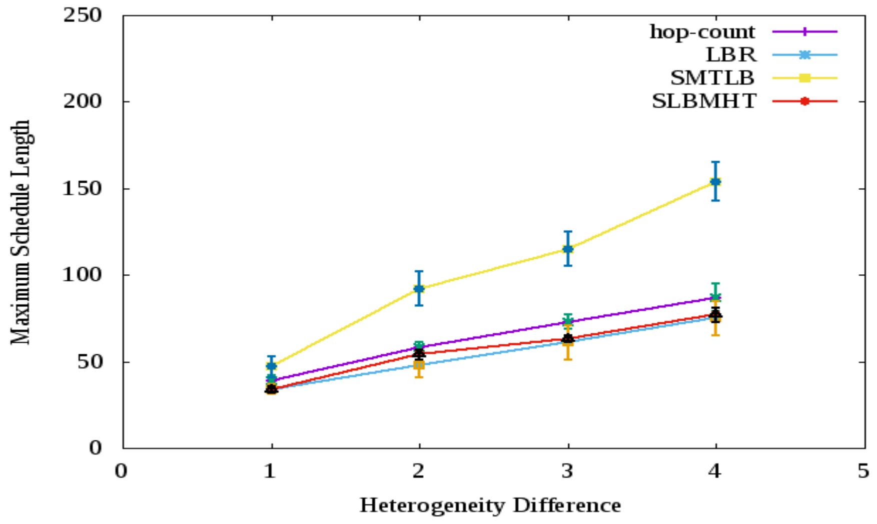
As shown in Figures 6.6a and 6.6b, the SLBMHT algorithm results in smallest schedule length compared to other approaches. It is seen from Figure 6.6c that performance of LBR is better than SLBMHT. The SLBMHT performs better than the other two algorithms. Detailed discussion about quantitative comparison of the results is given in the last subsection.



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner in each sub-region



(c) Sinks are nearby

Figure 6.6: Dependency of Max. Schedule Length on Heterogeneity Difference

Control Overhead and Energy Consumption during Control Phase

In Figures 6.7a,6.7b and 6.7c, the graphs of Control Overhead v/s. Heterogeneity Difference are shown. It is observed from the graphs that as heterogeneity difference increases, control overhead also increases. This is natural because increase in heterogeneity difference means increase in heterogeneity of region 2. As heterogeneity increases, aggregation becomes poor. The nodes require more time slots. Thus the control messages required for selection of slot and parent also increase. As a result, control overhead increases.

When parent selection is done based on hop count, control overhead consists of following components:

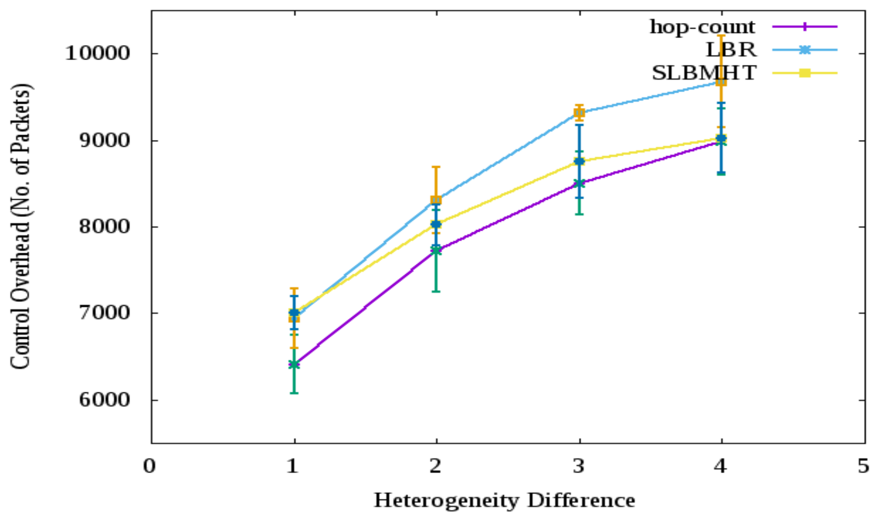
- HELLO messages flooded by sinks for leveling.
- Control messages generated due to execution of DICA[21] for scheduling & tree formation. The messages involved are Request, Response, Schedule and Forbidden. These messages are exchanged between a node doing slot/parent selection and candidate parents of the same node.

In LBR algorithm, control overhead consists following components.

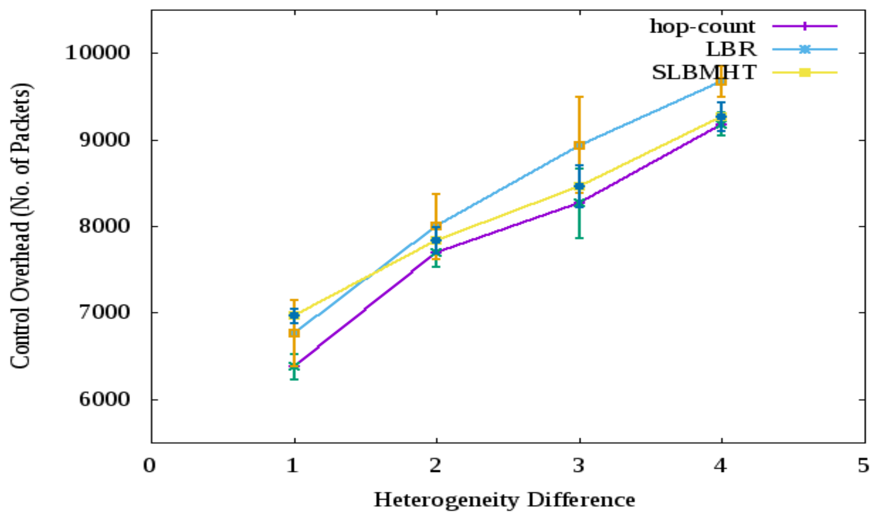
- HELLO messages flooded by sinks for leveling.
- Every sink floods its neighbor count in the network.
- Control messages generated due to execution of DICA[21] for scheduling & tree formation.

In SLBMHT algorithm, control overhead consists of following components.

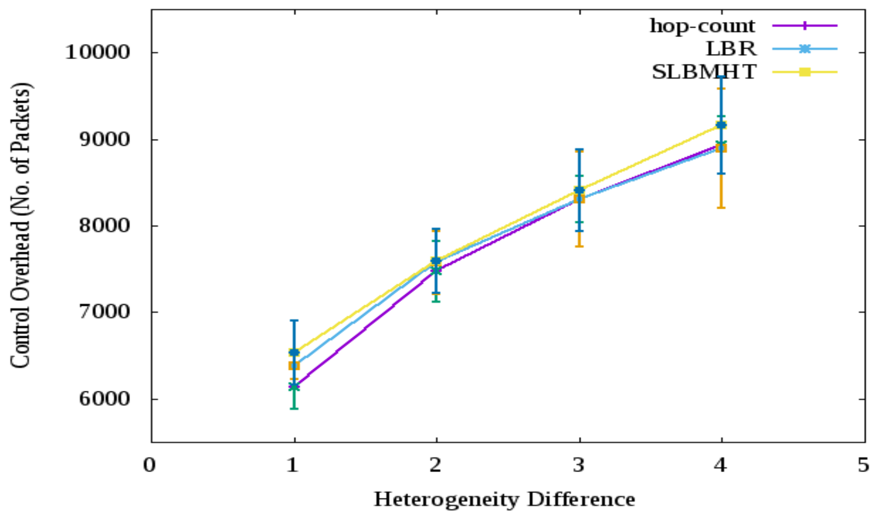
- HELLO messages flooded by sinks for leveling.
- Every node broadcasts a message containing its type and types of its one hop neighbors.
- Transmission of JOIN message by every node towards sink.
- Flooding of BAL_NOT_REQD message from sink with small schedule length in its tree.



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner in each sub-region



(c) Sinks are nearby

Figure 6.7: Dependency of Control Overhead on Heterogeneity Difference

- Flooding of BAL_REQD message from sink with large schedule length in its tree.
- Broadcasting of SINK_CONFIRM and SINK_MODIFIED messages by nodes.
- Control messages due to execution of AAJST for scheduling & tree formation.

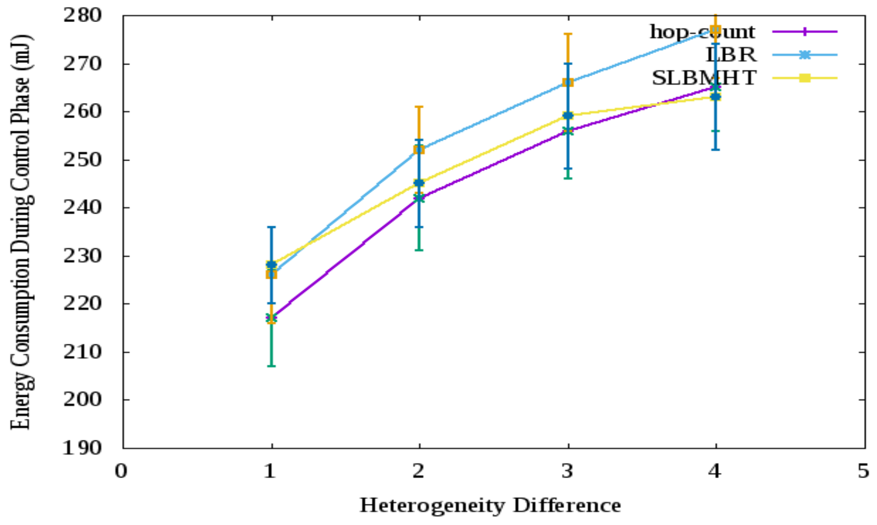
The hop count based method should result in the least control overhead as it involves the least number of control messages. The SLBMHT algorithm involves maximum number of control messages so its should result in highest control overhead. The performance of LBR should remain between the two. It seen from Figure 6.7c that control overhead of SLBMHT is slightly more compared to other two algorithms. In Figures 6.7a and 6.7b, control overhead of SLBMHT remains less than that of LBR. The explanation is given below.

The control overhead consists of following two components: (i) Messages required for schedule length balancing (i.e JOIN, BAL_REQD, BAL_NOT_REQD) (ii) Messages used during scheduling & parent selection (i.e. REQUEST, REPLY, SCHEDULE, FORBIDDEN). The SLBMHT algorithm not only shifts nodes from tree T_2 to T_1, but also uses AAJST algorithm during slot and parent selection. As AAJST algorithm does parent selection based on type of the outgoing packet, it schedules the tree using less number of slots compared to DICA_EXTENSION. In contract, LBR and hop-count based approaches use DICA_EXTENSION for scheduling and tree formation. As a result, they need more number of transmission slots compared to SLBMHT.

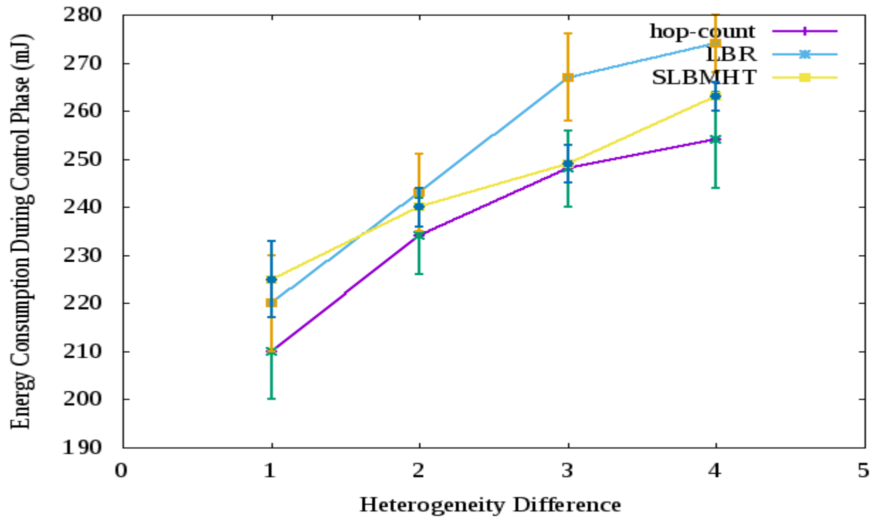
Thus, SLBMHT may require additional messages for balancing, but number of required slots are less. The corresponding control messages are also reduced. As a result, the overall control overhead remains less than that of LBR.

Energy consumption during control phase is directly proportional to control overhead. In Figures 6.8a,6.8b and 6.8c, the graphs for the same are shown. The nature of the graphs is same as in Figures 6.7a,6.7b and 6.7c. So, more explanation is not given here.

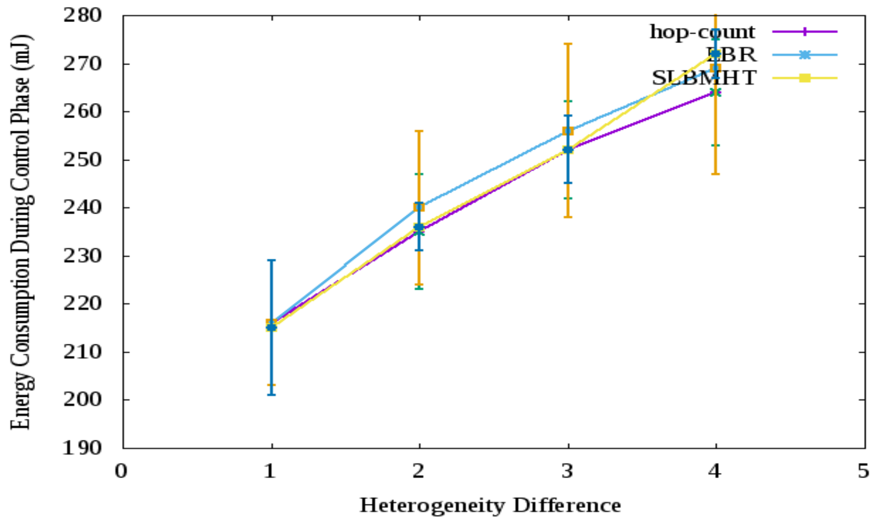
It is seen from Figure 6.8c that SLBMHT algorithm results in energy consumption near to that of hop-count based approach. For the other two figures, energy consumption of SLBMHT is higher. Detailed discussion about quantitative comparison of the results is given in the last sub-section.



(a) Sinks are at center in each sub-region

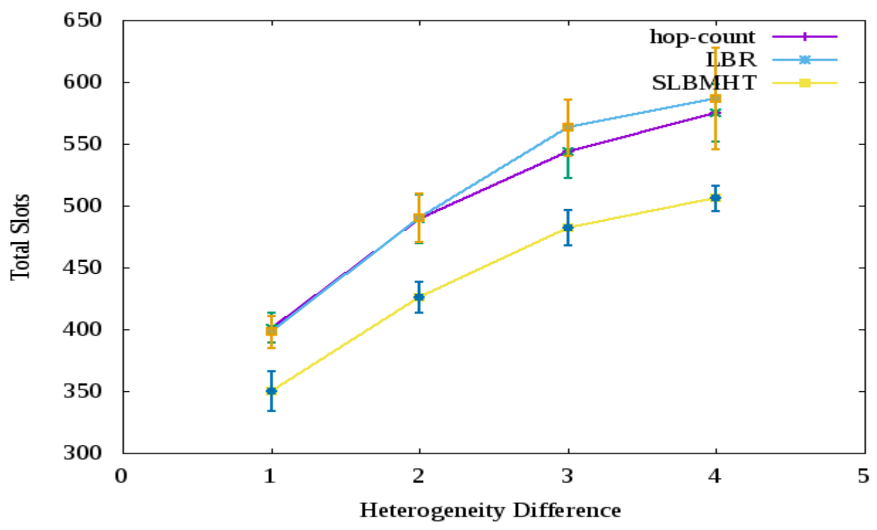


(b) Sinks are at diagonal corner in each sub-region

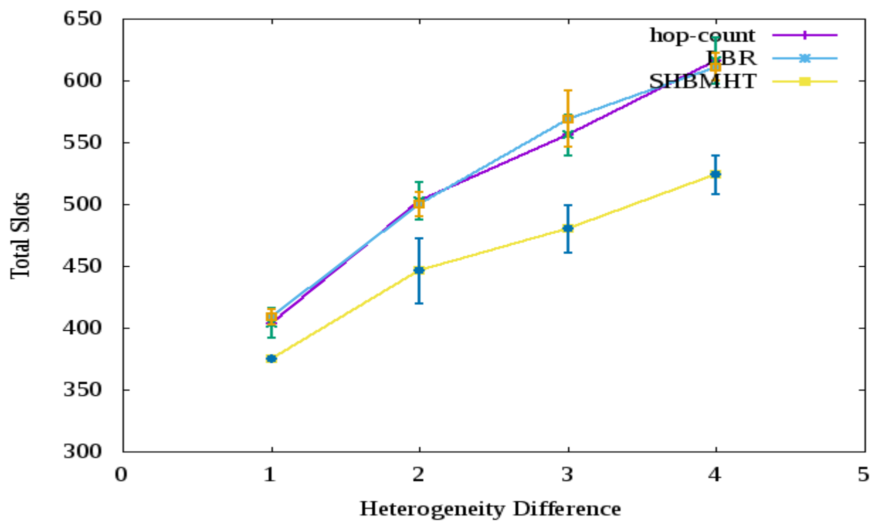


(c) Sinks are nearby

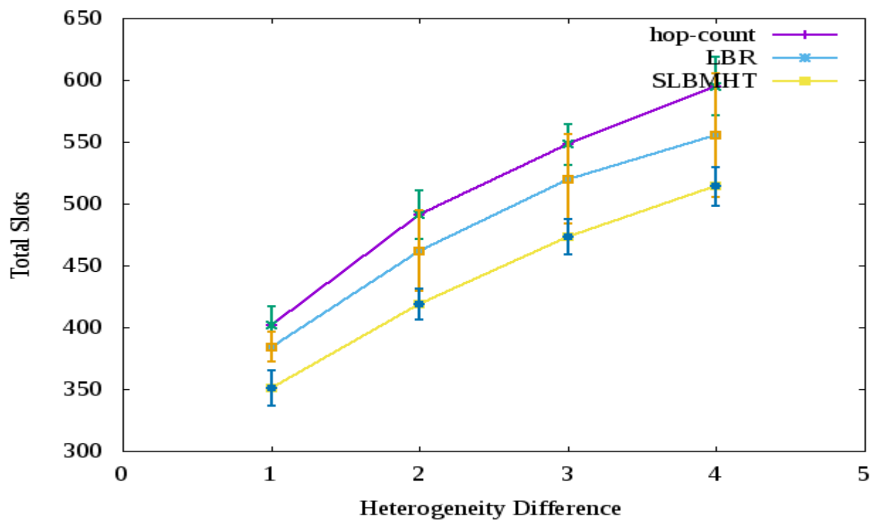
Figure 6.8: Dependency of Energy Consumption during Control Phase on Heterogeneity Difference



(a) Sinks are at center in each sub-region

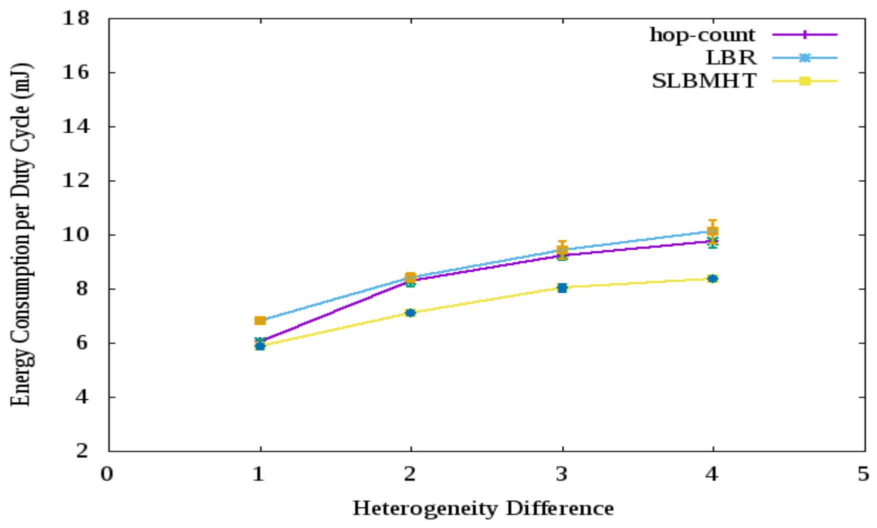


(b) Sinks are at diagonal corner in each sub-region

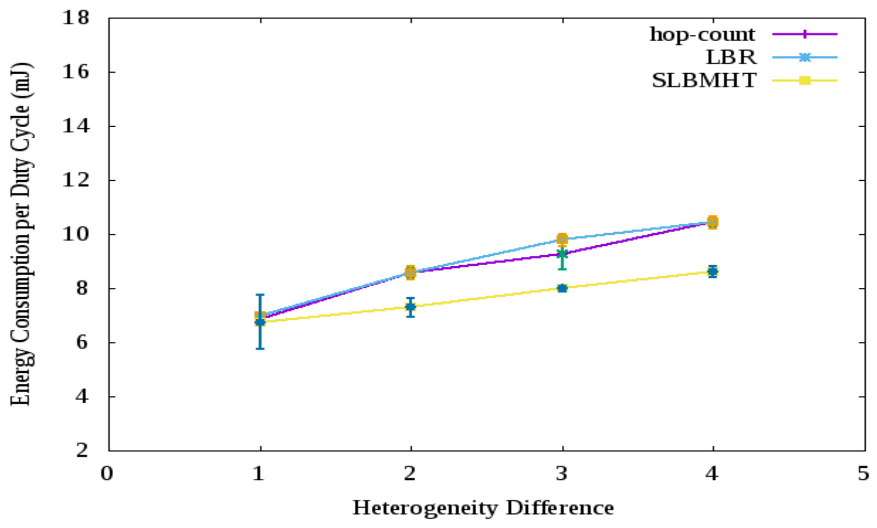


(c) Sinks are nearby

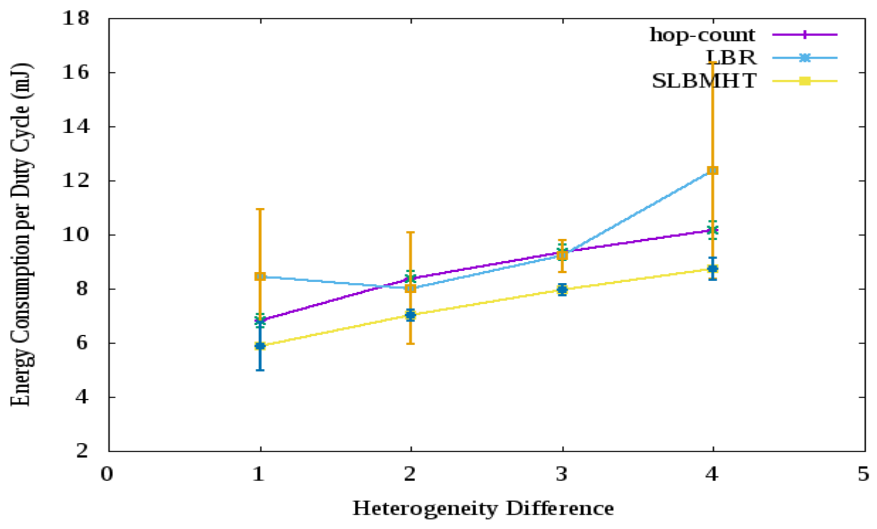
Figure 6.9: Dependency of Total Slots on Heterogeneity Difference



(a) Sinks are at center in each sub-region



(b) Sinks are at diagonal corner in each sub-region



(c) Sinks are nearby

Figure 6.10: Dependency of Energy Consumption per Duty Cycle on Heterogeneity Difference

Total Slots and Energy Consumption During Data Phase

In Figure 6.9a, 6.9b and 6.9c, the graphs for Total slots v/s. Heterogeneity Difference are shown. It is seen that as heterogeneity difference increases, total slots increases. Here total slots means count of all the slots used to schedule the tree. It is different than schedule length. Due to spatial slots reuse, schedule length is less than total number of slots.

As mentioned earlier, increase in heterogeneity difference means increase in heterogeneity level of region 2. As region 2 becomes more and more heterogeneous, aggregation becomes poor. More slots are required to schedule the nodes. Thus the count of total slots increases with increase in heterogeneity difference.

The SLBMHT algorithm performs schedule length balancing as well as attribute aware scheduling & parent selection. Whereas other two algorithms use DICA_EXTENSION for scheduling. So, SLBMHT schedules the nodes in smaller number of slots compared to the other two algorithms.

In Figure 6.10a, 6.10b and 6.10c, the graphs for Energy Consumption per Duty Cycle v/s. Heterogeneity Difference are shown. The duty cycle means one TDMA frame. Energy consumption during data phase is sum of total energy consumed during all the frames.

Energy consumed per duty cycle is directly proportional to number of slots used to schedule the network. If a node is assigned more number of slots, it will result in more energy consumption. As heterogeneity difference increases, aggregation becomes poor. So, more number of slots are required per node. Thus energy consumption also increases.

It is explained through Figures 6.9a,6.9b and 6.9c that the SLBMHT algorithm results in less number of slots than other algorithms. As a result, energy consumption per duty cycle is also less compared to the other three algorithms. Detailed discussion about quantitative comparison of the results is given in the last sub-section

6.5.4 Quantitative Comparison of Results

In the Tables 6.3 and 6.4 percentage improvement in schedule length difference and maximum schedule length respectively are summarized. To calculate improvement for a particular case, result of SLBMHT algorithm is compared with the second best performing

Sinks in Center		
Hetero. Deviation	Avg improvement (%)	Std. deviation (%)
3	22%	14%
4	56%	8%
5	43%	2%
6	39%	1%
Sinks at Diagonal Corner		
Hetero. Deviation	Avg improvement (%)	Std. deviation (%)
3	-52%	-8%
4	14%	5%
5	14%	4%
6	11%	11%
Sinks Nearby		
Hetero. Deviation	Avg improvement (%)	Std. deviation (%)
3	53%	40%
4	30%	23%
5	18%	17%
6	34%	12%

Table 6.3: Percentage Improvement in Schedule Length Difference

Sinks in Center		
Hetero. Deviation	Avg improvement (%)	Std. deviation (%)
3	8%	3%
4	16%	7%
5	20%	2%
6	14%	7%
Sinks at Diagonal Corner		
Hetero. Deviation	Avg improvement (%)	Std. deviation (%)
3	-3%	9%
4	10%	7%
5	6%	19%
6	2%	26%
Sinks Nearby		
Hetero. Deviation	Avg improvement (%)	Std. deviation (%)
3	0%	4%
4	-13%	11%
5	-3%	14%
6	-5%	10%

Table 6.4: Percentage Improvement in Maximum Schedule Length

algorithm for the given case.

It is observed from the Table 6.3 that the SLBMHT algorithm performs quite well when sinks are in center of sub-regions. The average improvement achieved by the

Sinks in Center	
Hetero. Deviation	Avg increase (%)
3	2%
4	14%
5	13%
6	14%
Sinks at Diagonal Corner	
Hetero. Deviation	Avg increase (%)
3	2%
4	15%
5	14%
6	17%
Sinks Nearby	
Hetero. Deviation	Avg increase (%)
3	14%
4	16%
5	15%
6	15%

Table 6.5: Percentage Reduction in Energy Consumption during Data Phase

Sinks in Center	
Hetero. Deviation	Avg increase (%)
3	5%
4	1%
5	1%
6	1%
Sinks at Diagonal Corner	
Hetero. Deviation	Avg increase (%)
3	7%
4	2%
5	1%
6	4%
Sinks Nearby	
Hetero. Deviation	Avg increase (%)
3	1%
4	1%
5	1%
6	1%

Table 6.6: Percentage Increase in Energy Consumption during Control Phase

SLBMHT algorithm ranges from minimum 22% to maximum 56%. When sinks are at diagonal corner or nearby, standard deviation in percentage improvement is high in some cases such that the difference between average value and standard deviation results in a small value. So, the confidence in improvement is low in such cases.

When sinks are at diagonal corner, the distance of nodes belonging to tree T_2 from sink S_1 is more compared to the case when sinks are in center. Moreover, as per equation 6.1, new estimated schedule length of T_1 would also depend on *aggr_dist* i.e. the shortest distance at which the packet generated by given node would be aggregated in its journey towards S_1. The term may be assuming large values for some nodes. Due to these reasons, for many nodes of T_2, new estimated schedule length must be remaining greater than balanced schedule length. So, such nodes do not switch to tree T_1. As a result, the SLBMHT algorithm is not able to reduce the schedule length difference when sinks are at diagonal corner.

When sinks are nearby, it is likely that there would very less nodes of tree T_2 at a distance greater than h_2^{bal} such that they are not far from sink S_1. In addition, the term *aggr_dist* as mentioned above may be a reason of new estimated schedule length being greater than balanced schedule length. So, enough nodes do not switch to tree T_1. As a result, the SLBMHT algorithm is not able to reduce the schedule length difference when sinks are nearby.

It is seen from Table 6.4 that improvement in overall schedule length ranges from 8% to 20% when sinks are in center. When sinks are at diagonal corner, high standard deviation implies little confidence in the average value. When sinks are nearby, average values are negative. As seen from graphs of Figure 6.6, the LBR performs better than the SLBMHT algorithm when sinks are nearby. The negative numbers indicate the improvement achieved by LBR compared to the SLBMHT algorithm.

The reason of no or little improvement in schedule length when sinks are at diagonal corner or nearby is that in such cases the SLBMHT algorithm is not able to balance the schedule lengths. In addition, many times due to racing conditions, node has to try for higher slot to form collision-free schedule. As a result, the highest slot number may be a large value. So, schedule length remains high. Still the results for total transmission slots as in Figure 6.9 indicates that the SLBMHT results in improvements total slots used to schedule the entire network. The result is seen in terms energy savings during data transfer phase.

Thus from Tables 6.3 and 6.4, it can be concluded that the SLBMHT algorithm results in considerable improvement compared to other three algorithms when sinks are in center of each sub-region. But for the other two sink placements, the performance improvement is not guaranteed.

In Table 6.5, percentage improvement in energy consumption during data phase as achieved by the SLBMHT algorithm is summarized. It results in 2% to 17% improvement across all three sink placements. The main reason is that due to usage of AAJST for scheduling & parent selection, aggregation is improved. As a result, less number of transmission slots are needed. So, less energy is consumed.

In Table 6.5, percentage increase in energy consumption during control phase as achieved by the SLBMHT algorithm is summarized. The maximum increase is 7%. As already mentioned, the improvement in energy consumption during data phase is between 2% to 17%. The control phase does not take place too often. It takes place after many cycles of data phase. Thus the increase in energy consumption during control phase in balanced by savings in energy consumption during data phase. As a result, network lifetime is going to increase.

6.6 Summary

In this chapter, schedule length balancing for multi-sink heterogeneous sensor networks is presented. The proposed algorithm is able to balance the schedule lengths of trees when nodes are not uniformly distributed or when different regions have different levels of heterogeneity. It is shown through simulation results, that proposed algorithm results in smaller overall schedule length compared to other algorithms. As less number of slots are required to schedule the network, energy consumption during data transmission phase is also reduced. The proposed algorithm results in more control overhead as additional messages are required for balancing. But the control phase takes place after long time-intervals. So, energy consumption during control phase may be balanced by energy savings during data phase.

Chapter 7

Conclusion & Future Work

In this chapter, concluding comments about the work done is given. Some future research directions are also mentioned.

7.1 Conclusion

We have proposed following algorithms: (i) AAJST (Attribute Aware Joint Scheduling & Tree Formation) for single-sink heterogeneous networks. (ii) SLBMHM (Schedule Length Balancing for Multi-sink HoMogeneous networks) (iii) SLBMHT (Schedule Length Balancing for Multi-sink HeTerogeneous networks).

The AAJST algorithm is based on the idea of selecting parent considering the type of packet to be forwarded. It results in better aggregation compared to DICA_EXTENSION. As aggregation is improved, less number of packets flow in the network. As a result, nodes need less number of transmission slots. Finally, schedule length goes down. As nodes need less number of time-slots, energy spent for exchanging control messages for slot/parent selection is also reduced. In addition, due to better aggregation, nodes transmit and receive less number of packets. So, energy consumed during data transmission phase is also reduced.

From simulation results, it is seen that AAJST results in in 5% to 10% reduction in schedule length compared to DICA_EXTENSION. It results in approximately 5% less energy consumption during control phase. Energy savings during data phase range from 15% to 30%. Thus AAJST seems better choice for scheduling & tree formation in heterogeneous networks.

The SLBMHM algorithm attempts to balance schedule lengths of trees in homogeneous networks when node distribution is not uniform. The algorithm guides every node to join a tree such that when actual scheduling & parent selection algorithm runs, resulting trees have balanced schedule lengths. If schedule length balancing is attempted after scheduling is done, it is likely to result in more control overhead. Because when nodes shift from one tree to other, both the trees need be rescheduled.

In proposed algorithm, every sink estimates schedule length of its tree based on average density (σ) and height(h) of the tree. Sinks exchange schedule lengths and calculate balanced schedule length (SH_{bal}). Nodes belonging to the tree having schedule length larger than SH_{bal} move to another tree having smaller schedule length. It is ensured that schedule length of one tree is reduced but that of the other tree is not increased beyond SH_{bal}.

The performance of SLBMHM is evaluated using simulations. It is considered that two sinks are present. The simulation area is divided into two sub-regions. One sink is present in each sub-region. Three different sink placements namely sinks in center, sinks at diagonal corners and sinks near to each other are used.

Through simulations, it is found that SLBMHM algorithm results in 13% to 74% reduction in schedule length difference and 9% to 24% reduction in overall schedule length. As control messages are required to achieve balancing, energy consumption during control phase is increased. The increase is between 3% to 20%. The control phase does not occur frequently. So, this does not put much burden on sensor nodes. At the same time, improvement in schedule length is seen at every TDMA frame. The balancing algorithm does not change neighborhood of most of the nodes. So, number of packets received and transmitted by a node does not change much. As a result, energy consumption during data transmission phase remains almost same in all the three algorithms i.e. hop-count based approach, LBR and SLBMHM.

The other reason of unbalanced schedule lengths is difference in heterogeneity of regions. As explained earlier, the tree spanning through less heterogeneous region has smaller schedule length compared to the tree spanning through more heterogeneous region. The SLBMHT algorithm attempts to balance schedule lengths in such heterogeneous networks. As such, SLBMHT is an extension of SLBMHM. It can balance schedule lengths in both the cases i.e. uneven node distribution and uneven heterogeneity distribution.

Like SLBMHM, simulation based evaluation of SLBMHT algorithm uses three different sink placement scenarios. When sinks are in center of each sub-region, SLBMHT results in 22% to 56% reduction in schedule length difference and 8% to 20% reduction in overall schedule length. But in the other two scenarios, the algorithm does not always outperform existing algorithms.

The SLBMHT algorithm results in 2% to 17% less energy consumption during data phase. The reason is that the SLBMHT algorithm uses AAJST for scheduling and parent selection. The other approaches use DICA_EXTENSION. Due to better aggregation, the SLBMHT algorithm results in better energy consumption than the existing algorithms. The improvement in data energy consumption is seen in all the three sink placement scenarios. The energy consumption during control phase is increased maximum by 7%.

The SLBMHT algorithm results in reduction in overall schedule length and energy consumption during data phase. It consumes more energy during control phase. But as control phase does not take place very frequently, energy loss during control phase is balanced by energy savings during data phase. The data phase is repeated many times before control phase takes place. Thus network lifetime is likely to increase.

7.2 Future Work

In simulation design of both the algorithms i.e. SLBMHM and SLBMHT, three different placements of sinks are considered: (i) sinks in centre of each sub-region (ii) sinks at diagonal corner (iii) sinks nearby. The proposed algorithms would be better evaluated if simulations are carried out for random placements of sinks. Even number of sinks are set to 2 in the simulation. Again, for better evaluation, more than 2 sinks can be used. In short, simulations may be carried out for more than two randomly placed sinks.

In the proposed algorithms, first schedule lengths of tentative trees are estimated by the sinks and then tree switching takes place. One other way of handling the problem of schedule length balancing is as follows. First, form actual trees and schedule them. Once actual schedule lengths are known, let the tree switching take place. When a node calculates new estimated schedule length of the tree assuming that it would switch to that tree, one parameter used in calculation is current schedule length of that tree. In this work, the current schedule length is an estimated value. But if actual trees and schedules are formed initially, the current schedule length would be exact value. It would be interesting to compare the performance of both the approaches: (i) forming tentative trees followed by switching (ii) forming actual trees followed by switching. Again, the performance parameters would be same i.e. schedule length difference, overall schedule length, energy consumption during control phase and data phase.

Chapter 8

List of Publications

8.1 Papers Published

1. T. Vasavada and S. Srivastava, “Review of Fairness and Graph Colouring Methods for Data Collection in Wireless Sensor Networks”, in *IEEE INDICON*, December, 2013.

2. T. Vasavada and S. Srivastava, “TDMA Scheduling of Group Aware Tree in Wireless Sensor Networks”, in *IEEE INDICON*, December, 2014.

3. T. Vasavada and S. Srivastava, “Distributed Scheduling and Tree Formation for Heterogeneous Wireless Sensor Networks”, in *IEEE International Conference on Advanced Networking and Telecommunication Systems (ANTS)*, November, 2016.

4. T. Vasavada and S. Srivastava, “Schedule Length Balancing for Aggregated Convergecast in Multiple Sinks Wireless Sensor Networks”, in *IEEE Regions 10 Symposium (TENSYMP)*, July, 2017.

8.2 Paper Under Preparation

1. T. Vasavada and S. Srivastava, “Schedule Length Balancing for Aggregated Convergecast in Multi-sink Heterogeneous Wireless Sensor Networks”, to be submitted to Wireless Networks-The Journal of Mobile Communication, Computation and Information, Springer Publications.

Bibliography

[1] F.Wang et. al, "Networked Wireless Data Collection: Issues, Challenges and Approaches", in *IEEE Communication Surveys & Tutorials*, Vol. 13, No. 4, 2011.

[2] G. Tolle et. al, "A Macroscope in the Redwoods", in *ACM SenSys*, 2005.

[3] L. Selavo et. al, "LUSTER: Wireless Sensor Network for Environmental Research", in *ACM SenSys*, 2007.

[4] G. Barrenetxea et. al, "SensorScope: Out-of-the-Box Environmental Monitoring", in *ACM/IEEE IPSN*, 2008.

[5] G. WernerAllen et. al,"Fidelity and Yield in a Volcano Monitoring Sensor Network", in *USENIX OSDI*, 2006.

[6] W.Z.Song et. al, "Air-dropped Sensor Network for Real-time High-fidelity Volcano Mon- itoring", in *ACM MobiSys*, 2009.

[7] Y. Kim et. al, "NAWMS: Nonintrusive Autonomous Water Monitoring System", *ACM SenSys*, 2008.

[8] S. Kim et. al, "Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks", in *ACM/IEEE IPSN*, 2007.

[9] M. Ceriotti et. al, "Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment", in *ACM/IEEE IPSN*, 2009.

[10] C. Hartung et. al, "FireWxNet: A Multi-Tiered Portable Wireless System for Monitoring Weather Conditions in Wildland Fire Environments", in *ACM MobiSys*, 2006.

[11] M.D.Francesco et. al, “Data collection in Wireless Sensor Networks with Mobile Elements: A survey”, in *ACM Transactions on Sensor Networks*, Vol. 8, No. 1, August 2011.

[12] O.D.Incel et. al, “Fast Data Collection in Tree based Wireless Sensor Networks”, in *IEEE Transactions on Mobile Computing*, Vol. 11, No. 1, 2012.

[13] R. Soua et. al, “MUSIKA: A Multi-Channel Multiple sinks Data Gathering Algorithm for Wireless Sensor Networks”, in *IEEE International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2013.

[14] M. Pan et. al, “Quick convergecast in ZigBee Beacon Enabled Wireless Sensor Networks”, in *ACM Journal of Computer Communications*, 2008.

[15] B.Malhotra et. al, “Aggregation Convergecast Scheduling in Wireless Sensor Networks”, in *Springer Journal of Wireless Networks*, Vol 17, Issue 2, 2011.

[16] A.Ghosh et. al, “Bounded Degree Minimum Radius Spanning Trees for Fast Data Collection in Wireless Sensor Networks”, in *IEEE INFOCOM*, 2010.

[17] Mixg Hia et. al, “W-MAC: A Workload-aware MAC Protocol for Heterogeneous Convergecast in Wireless Sensor Networks”, in *Sensors Journal*, MDPI Publications, Vol 17, Issue 2, 2011.

[18] Ichrak Amdouni et. al, “Joint Routing and STDMA-based Scheduling to Minimize Delays in Grid Wireless Sensor Networks”, *A Research Report*, September 2014.

[19] Fang-Jing Wu et. al, “Distributed Wake Up Scheduling for Data Collection in Tree based Wireless Sensor Networks”, in *IEEE Communication Letters*, Vol. 13, Issue 3, 2009.

[20] Chansu Yu et. al, “Many to One Communication Protocol for Wireless Sensor Networks”, in *International Journal of Sensor Networks*, Inderscience Publications, Vol. 12, Issue 3, 2012.

[21] M.Bagga et. al, “Distributed Low Latency Data Aggregation Scheduling in Wireless Sensor Networks”, in *ACM Transactions on Sensor Networks*, Vol. 11, No. 3, April 2015.

[22] M.Bagga et. al, “Efficient Multi-path Data Aggregation Scheduling in Wireless Sensor Networks”, in *IEEE International Conference on Communications*, 2013.

[23] M.Bagga et. al, “Multi-path Multi-Channel Data Aggregation Scheduling in Wireless Sensor Networks”, in *IEEE Wireless Days International Conference*, 2013.

[24] R.Hwang et. al, “A Distributed Scheduling Algorithm for IEEE 802.15.4e Wireless Sensor Networks”, in *International Journal of Computer Standards & Interfaces*, Elsevier Publications, Vol. 52, Issue C, 2017.

[25] W. Lee et.al, “FlexiTP: A Flexible Schedule based TDMA Protocol for Fault Tolerant and Energy-Efficient Wireless Sensor Networks”, in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 6, 2008.

[26] B.Zeng et. al, “A Collaboration based Distributed TDMA Scheduling Algorithm for Data Collection in Wireless Sensor Network”, in *Journal of Networks*, Academy Publishers, Vol. 9, No.9, 2014.

[27] R.Soua et. al, “A Distributed Joint Channel and Time Slot Assignment for Convergecast in Wireless Sensor Networks”, in *6th International Conference on New Technology, Mobility and Security*, Dubai, 2014.

[28] I. Rhee et. al, “DRAND: Distributed Randomized TDMA Scheduling for Wireless Ad hoc Networks”, in *IEEE Transactions on Mobile Computing*, Vol. 8, Issue 10, 2006.

[29] C.Lin et. al, “A Distributed and Scalable Time Slot Allocation Protocol for Wireless Sensor Networks”, in *IEEE Transactions on Mobile Computing*, Vol. 10, No. 4, 2011.

[30] A. Saifullah et. al, “Distributed Channel Allocation Protocols for Wireless Sensor Networks”, in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, Issue 9, 2014.

[31] Y. Wang et. al, “A Deterministic Distributed TDMA Scheduling Algorithm for Wireless Sensor Networks”, in *International Conference on Wireless Communications*, Networking and Mobile Computing, Shanghai, China, 2007.

[32] S.Isik et. al, “Multi-sink load balanced forwarding with a multi criteria fuzzy sink selection for video sensor networks”, in *Journal of Computer Networks*, ElseVier Publications, Vol. 56, Issue 2, 2012.

[33] C.Wang et. al, “A Load Balanced Routing Algorithm for Multi Sink Wireless Sensor Network”, in *IEEE International Conference on Communication Software and Networks (ICCSN)*, 2009.

[34] C. Zhang et. al, “Load-balancing Routing for Wireless Sensor Networks with Multiple Sinks”, in *12th IEEE International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2015.

[35] A.N.Eghbali et. al, “An Energy Efficient Load-balanced Multi-Sink Routing Protocol for Wireless Sensor Networks”, in *10th IEEE International Conference on Telecommunications*, 2009.

[36] H. Jiang et. al, “Energy optimized routing algorithm in Multi sink wireless sensor networks”, in *International Journal of Applied Mathematics and Information Sciences*, Natural Sciences Publications, Vol. 8, No. 1, 2014.

[37] C. Wu et. al, “A Novel Load Balanced and Lifetime Maximization Routing Protocol in Wireless Sensor Networks”, in *IEEE Vehicular Technology Conference*, Spring 2008.

[38] Y. K. Sia et. al, “Spanning Multi-tree Algorithms For Load Balancing in Multi Tree Wireless Sensor Networks with Heterogeneous Traffic Generating Nodes”, in *12th IEEE International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2015.

[39] B. Yu et. al, “Minimum Time Aggregation Scheduling in Multi-Sink Sensor Networks”, in *8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad hoc Communications and Networks*, 2011.

[40] W. Zao et. al, “Scheduling Data Collection with Dynamic Traffic Patterns in Wireless Sensor Networks”, in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, Issue 4, 2013.

[41] P.Van, et. al, “Delay Efficient Data Collection with Dynamic Traffic Patterns in Wireless Sensor Networks”, in *International Conference of Wireless Networks*, 2013.

[42] L.Zang et. al, “Fault Tolerant Scheduling for Data Collection in Wireless Sensor Networks”, in *proceedings of GLOBECOM*, 2012.

[43] Suchetana Chaktraborty et. al, “Convergecast tree management from arbitrary node failure in sensor network”, in *Ad Hoc Networks Journal*, Elsevier Publications, Vol. 11, Issue 6, 2013.

[44] Suchetana Chakrobarty et. al, “Topology Management Ensuring Reliability in Delay Sensitive Sensor Networks with Arbitrary Node Failure”, in *International Journal of Wireless Inf. Networks*, Springer Publications, Vol. 21, Issue 4, 2014.

[45] F. Ren et. al, “Attribute-Aware Data Aggregation Using Potential-Based Dynamic Routing in Wireless Sensor Networks”, in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, Issue 5, 2012.

[46] T. Winter et. al, “IPv6 Routing Protocol for Low-Power and Lossy Networks”, in *Internet Engineering Tast Force, RFC 6550.*

[47] M. Palattella et. al, “Standardized Protocol Stack for Internet of (Important) Things”, in *IEEE Communications Surveys & Tutorials*, Vol. 15, Issue 3, 2013.

Appendices

Appendix A

Pseudo-code for SLBMHM Algorithm

A.1 Procedures Executed at Sensor Nodes

In this subsection, pseudo-code for procedures executed by non-sink nodes is given. Execution begins with procedure MAIN.

```
procedure MAIN
    $Packet\ p = recv()$
    if $p.msg = HELLO$ then
        $recv\_HELLO(p)$
        if $HELLO\ recd\ from\ all\ N\ sinks$ then
            $decide\_home\_sink()$
    if $p.msg = JOIN$ then
        $forward\_JOIN$
    if $p.msg = BAL\_NOT\_REQD$ then
        if $sender\ sink\ is\ home\ sink$ then
            $send\_SINK\_CONFIRM()$
    if $p.msg = BAL\_REQD$ then
        $modify\_home\_sink()$
    if $p.msg = SINK\_CONFIRM$ then
        $recv\_SINK\_CONFIRM()$
    if $p.msg = SINK\_MODIFIED$ then
        $recv\_modified\_home\_sink$
```

```
end procedure
procedure $recv\_HELLO(Packet\ p)$
    if $sender\_node$ not in $neibors[]$ then
        Add sender into $neibors[]$ table
    else
        update height of neibor
    update height of current node
    Broadcast HELLO to forward it
end procedure
procedure $decide\_home\_sink$
    $home\_sink$ = nearest sink in terms of hop count
    $temp\_parent$ = node nearer to $home\_sink$ than current node
    $level$ = distance from $home\_sink$ in hop count
    $decide\_is\_leaf()$
end procedure
procedure $decide\_is\_leaf$
    Decide $home\_sink$ and level for every neighbor
    if current node has no child then
        $is\_leaf = 1$
        $send\_JOIN()$
end procedure
procedure $send\_JOIN()$
    Create a packet of type JOIN
    Store neighbor count and height in JOIN packet
    Send JOIN packet to $temp\_parent$
end procedure
procedure $forward\_JOIN$(Packet p)
    if sender of packet is temporary child then
        Update total neighbor count
        Update max. height
    if JOIN packets are received from all temporary children then
        Create new packet of type JOIN
        Store total neighbor count and maximum height in JOIN
        Send JOIN packet to $temp\_parent$
```

```
end procedure
procedure send_SINK_CONFIRM()
    Create a packet of type SINK_CONFIRM
    Store home_sink in SINK_CONFIRM packet
    Broadcast SINK_CONFIRM packet
end procedure
procedure recv_SINK_CONFIRM()
    Update home_sink of sender
    if sch_len of home_sink of sender < sch_len of own tree then
        Decrement count of neighbors belonging to sinks having less schedule lengh
        than home − sink
    if count of neighbors belonging to sinks having less schedule length than
    home − sink is 0 AND balancing is required then
        modify_home_sink()
end procedure
procedure modify_home_sink()
    target_set = set of sinks having schedule_length less than balanced_schedule_length
    estimate current schedule length of each sink present in target_set assuming
    node will join the sink
    find the sink S from target set whose current estimated schedule length is
    minimum and less than balanced schedule length
    Create a packet of type SINK_MODIFIED
    Store new home_sink S in SINK_MODIFIED packet
    Set sink_changed field to 1 in SINK_MODIFIED packet
    Write new estimated schedule length of sink S in SINK_MODIFIED packet
    Broadcast SINK_MODIFIED packet
end procedure
procedure recv_sink_modified
    Update home_sink of sender
    if sch_len of home_sink of sender < sch_len of current node then
        Decrement count of neighbors belonging to sinks having less schedule length
        than home − sink
    if count of neighbors belonging to sinks having less schedule length than
    $home - sink$ is 0 then
        $modify\_home\_sink()$
end procedure
```

A.2 Procedures Executed at Sink Nodes

In this subsection, pseudo-code for procedures executed by sink nodes is given. Execution begins with procedure MAIN.

```
procedure MAIN
    Packet p = recv()
    if $p.msg = JOIN$ then
        $recv\_JOIN(p)$
end procedure
procedure send_hello
    Create a packet of type HELLO
    Store ID of current sink in HELLO packet
    Set LEVEL field in HELLO packet to 0
    Broadcast HELLO packet
end procedure
procedure recv_JOIN(PacketP)
    calculate average density (avg_density) and height of tree rooted at current sink
    Estimate schedule length based on avg_density and height
    Exchange estimated schedule length with other sinks
    Calculate bal_sch_len
    if estimated schedule length ¡= bal_sch_len then
        Create a packet of type BAL_NOT_REQD
        Broadcast BAL_NOT_REQD
    else
        Create a packet of type BAL_REQD
        Broadcast BAL_REQD
end procedure
```

Appendix B

Pseudo-code for SLBMHT Algorithm

The pseudo-code for SLBMHT algorithm is almost similar to that of SLBMHM. Entire pseudo-code is not presented. But differences are highlighted below:

- In JOIN message, every leaf node will write its type in addition to neighbor count and height. Every non-leaf node will write list of types present in the sub-tree rooted at itself in addition to total neighbor count and height of sub-tree.

- Every sink will estimate schedule length of its temporary tree based on average density, height and type-count.

- When a node tries to switch to a different tree, it will estimate new schedule length of that tree considering how many hops its packet will travel in the tree before it is aggregated. The corresponding formula is as given in equation 6.1.