# Heuristics Based Tree Switching in Two-sink Sensor Networks

TEJAS MUKESH VASAVADA[1]
SANJAY SRIVASTAVA[2]

DA-IICT, Gandhinagar

**Abstract.** In sensor networks, tree is a well-known topology formation method and TDMA is a desirable MAC protocol due to guaranteed channel access and no collisions. Many times node distribution across the region is not uniform. If finer observations are required in a region, node density is kept high. But in other regions where accurate readings are not needed, network may be sparse. Often multiple sinks are deployed in WSNs. Use of multiple sinks provides fault tolerance and load balancing. When multiple sinks are deployed, more than one sink-rooted trees are formed. The trees with dense node deployment would have higher schedule lengths than the trees with sparse node deployment. Thus trees part of the same network have different schedule lengths. In other words, schedule lengths are not balanced. As a result, nodes of some trees (with higher schedule length) have to wait for longer duration for transmission turn compared to the nodes of the other trees (with lower schedule length). As all the nodes belong to the same network, it is desirable that the waiting time for transmission turn should not be very different. So, schedule length balancing is required to ensure fairness. In this work, an algorithm known as HTSTSN (Heuristics based Tree Switching in Two-sink Sensor Networks) algorithm for two-sink network is proposed. It helps every node to decide which sink (i.e. tree) to join such that schedule lengths of trees remain balanced. The HTSTSN algorithm executes before actual scheduling algorithm. It is shown through simulations that the proposed algorithm results in average 13% to 74% reduction in schedule length difference and maximum 12% increase in energy consumption. It is found that the HTSTSN algorithm balances schedule length without much affecting the network lifetime.

**Keywords:** Sensor Networks, TDMA Scheduling, Multiple sinks, Heterogeneous Networks

(Received November 19th, 2018 / Accepted December 1st, 2019 )

## 1 Introduction

Sensors are tiny microelectronic devices which can sense physical quantities like temperature, pressure, humidity, solar radiation and many others. As mentioned in [1], there are many real life deployments of sensor networks for applications like habitat monitoring, environmental research, volcano monitoring and wild fire detection.

In addition to sensors, sink (also known as base station) is also deployed in the region. The sink node is connected to the Internet. The sensors send their readings to the sink node. The external world can access the readings from the sink.

Once nodes are deployed, logical topology must be formed so that every node would be able to send its readings towards sink. Tree and cluster are two well-known topology formation methods. We are focused on tree based networks. Sink is the root of the tree. As mentioned in [1], data transfer from sensors to sink is known as convergecast operation.

There are two types of convergecast operations ([1]) : (i) raw convergecast (ii) aggregated convergecast. In raw convergecast, every node forwards all the readings received from children. In aggregated convergecast, every node aggregates all incoming packets with its own packet and sends out only one packet. We are focused on aggregated convergecast.

During data transfer, Medium Access Control (MAC) is required. There are two possible choices:

TDMA (Time Division Multiple Access) and CSMA (Carrier Sense Multiple Access). In TDMA, every node of the tree is assigned time-slot to transmit readings to parent node. As TDMA completely prevents collisions, it is more preferable than CSMA. We have used TDMA in tree-based networks.

If network has to cover large area, large number of nodes need to be deployed. If all the nodes finally join the same tree, the diameter of the tree will increase. In case of aggregated convergecast, increase in tree size will result in increase in schedule length and end-to-end delay. In raw convergecast, funneling effect [1] will also take place.

One solution to reduce the tree size is to deploy more than one sinks. Thus instead of a single large tree, multiple small trees are formed. The schedule length of each small tree would be smaller than that of single large tree.

Sometimes it is required to have accurate readings in some regions of network. In other regions, accuracy may not be needed. To have accurate readings, more sensors are deployed. In the other regions, less sensors are deployed. As a result, some portion of network is dense whereas the other portion is sparse. Thus node distribution is not uniform across the entire network. Many times sensor nodes randomly deployed in the region of interest. When deployment is random, uniform node distribution can not always be guaranteed.

Once nodes are deployed, sink-rooted trees are formed. During tree formation, every node has to join one tree (i.e one sink). If every node joins the sink which is at the smallest hop distance, nodes would be evenly distributed across the sinks. The resulting trees would be of almost equal size and their schedule lengths would be almost same. This happens when node distribution is uniform. But when node distribution is not uniform, the trees would not be of same size if every node decides to join the nearest sink. The trees spanning dense region of network would have more nodes than the trees spanning through sparse region of the network. As a result, the schedule lengths of trees would also be different.

For example, two trees $T_1$ and $T_2$ are formed with schedule lengths $SH_1$ and $SH_2$ respectively. The overall schedule length SH for the entire network will be max($SH_1$,$SH_2$). If $SH_1 > SH_2$, SH would be equal to $SH_1$ and vice versa for $SH_1 < SH_2$.

If schedule length of a tree is $SH_i$, every node of that tree will get its turn to transmit after $SH_i$ time-slots. Thus if $SH_1 > SH_2$, every node of tree $T_1$ will have to wait for longer duration to get transmission turn compared to the nodes of tree $T_2$. As a result, packets generated by nodes of tree $T_1$ suffer from longer end-to-end delay compared to packets generated by nodes of $T_2$. If schedule lengths are balanced, $SH_1$ and $SH_2$ would be almost equal. Thus all the nodes would have to wait for almost equal time to transmit.

If all the nodes are owned by the same user, the nodes and the sinks may co-operate with one another so that both the trees have almost same schedule length. If it is found that one tree is larger than the other tree, nodes from the larger tree may switch to the smaller tree. Accordingly, in this work, an algorithm named as HTSTSN (Heuristics based Tree Switching in Two-sink Sensor Networks) is proposed. The HTSTSN algorithm should run prior to actual scheduling and tree formation algorithm. It guides every node to join a tree such that the resulting trees have balanced schedule lengths. The algorithm is designed for the case that only two sinks are present in the network. It is extensible for more than two sinks.

As summarized in Section II, there are many papers addressing load balancing across multiple sinks for raw convergecast. As per our knowledge, there is no work addressing balancing of schedule lengths of trees in multi- sink aggregated convergecast networks. Our work seems to be only one of its kind.

Rest of the paper is organized as follows: Related Papers are explained in Section II. The Proposed Algorithm is presented in Section III. Section IV covers simulation results. Conclusion and Future Work are presented in Section V and VI respectively.

## 2 Related Work

Some papers addressing issue of tree formation & scheduling in multi-sink sensor networks are summarized in this section.

In [7], 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 [8], 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 $r$. Thus the sink with more number of neighbors is preferred.

The given node may have multiple neighbors through which the selected sink can be reached. For each packet different neighbor is selected probabilistically. As each time different neighbor is selected, workload remains distributed across the neighbors also.

In [9] and [10], it is proposed to change the path towards sink when energy of nodes in the current path goes beyond specific threshold. In [11], 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 [12], 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 [13],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.

As explained in the previous section, schedule lengths of the trees may not be balanced because of non-uniform node distribution. None of the above mentioned papers address balancing of schedule lengths in multi-sink tree based networks. Thus it seems that the problem of schedule length balancing must be studied in detail. In [14], core idea of schedule length balancing is proposed without regorous simulations and proof of correctness. The current work is a substantial extension of the work done in [14]. Following are the key differences between the current work and [14].

- In this work, relationship presenting dependenace of schedule length of a tree on node density of the tree and height of the tree is derived empirically. Then it is used for the purpose of shifing nodes from a tree with higher schedule length to a tree with lower schedule length. In [14], shifting of nodes is done but without systematically looking at the relation among schedule length, density of nodes and height of the tree. Thus schedule length estimation and tree switching both are different than in [14].
- Here simulation-based evaluation is more robust than done in [14]. Following points explain the reasons:
  - Different sink positions are considered.
  - Performance of the proposed algorithm is compared with three other algorithms. In [14], performance is compared with only one trivial algorithm.
  - In [14], only performance paramter considered is difference in schedule length. Here, different other parameters like overall schedule length, control overhead, energy consumption during control phase and energy consumption during data phase are considered.
  - In [14], simulation results are presented by just single simulation run for fixed node deployment. Here, each performance parameter is studied with respect to density denviation of the network for multiple simulation runs with random node deployment.
- The proposed algorithm is supported by the proof of correctness in this work. In [14], correctness of algorithm is not addressed.

Thus our work is a significant extension of the work presented in [14].

## 3 HTSTSN Algorithm

The HTSTSN algorithm runs before actual scheduling & tree formation algorithm. In [2],[3],[4], DICA([5]) and [6] various scheduling algorithms are proposed. The DICA([5]) seems to be most appropriate approach for scheduling and tree formation in single sink aggregated convergecast network due to reason explained next. It follows joint approach i.e. scheduling and tree formation are not dealt separately but considered as a single problem. Every node selects slot and parent at the same time. If tree is formed first and then slot selection is done, the tree structure limits the performance of scheduling algorithm.

A simple extension of DICA([5]) for multiple sinks networks is explained below. It is refered to as hop-count based approach in rest of the paper. It is assumed that number of sinks is two.

To begin with, sinks take turns and flood HELLO packets in the network. The HELLO packet contains a field: depth. As explained in [5], depth field is used by nodes to find hop distance from the sink.

When flooding of HELLO is finished, every node knows its distance in hop count from each sink. Every node selects the sink at the least hop distance as home-sink. Also for each neighbor, following information is known: (i) ID of each neighbor (ii) Distance from each sink in hop count (iii) home-sink. Thus every node creates a table $nbr\_table$ to maintain above mentioned three types of information for each neighbor.

Thus nodes are divided into two different disjoint sets because every node has selected one of the two sinks as home-sink. Now joint scheduling & tree formation as proposed in [5] is executed in each group. Every node selects a parent node from the neighbors who belong the same home-sink as the given node. As a result, two different trees are formed and every node is assigned one or more time-slots.

If node distribution is not uniform, the hop-count based approach would results in very different schedule lengths of trees. This is justified through simulation results later. As mentioned earlier, HTSTSN algorithm is proposed in this work to balance the schedule lengths. Detailed discussion of the same algorithm is presented below.

Following are the assumptions used in HTSTSN algorithm: (i) There are two sinks in the network. (ii) Every node should be part of exactly one tree. (iii) Every sink would be root of exactly one tree. (iii) The node distribution may not be uniform.

The HTSTSN algorithm is described considering that two sinks namely $S_1$ and $S_2$ are available. But generalized version for the case where number is sinks is more than two is also possible. It is kept as future research work. The HTSTSN algorithm has two phases: (i) Estimation of schedule length (ii) Tree Switching Process. During the first phase, every sink estimates the schedule length of its possible tree. In the second phase, nodes from the tree with higher schedule length are shifted to the tree with lower schedule length. Different notations used in sub-sequent discussion are summarized in Table 1.

| Symbol | Description |
|---|---|
| $nbr\_tbl$ | Table of neighbors |
| $d_i$ | Distance in hop-count from sink $S_i$ |
| h | Hop distance from home-sink |
| temp-home-sink | Sink selected as home-sink temporarily |
| temp-parent | Temporarily selected parent |
| temp-child[] | List of temporary children |
| nbrcount | Sum of neighbor count of all the nodes present in the sub-tree |
| | rooted at sender of JOIN message |
| ndcount | Total number of nodes present in the sub-tree rooted at sender of JOIN message |
| $h_t$ | Height of the sub-tree rooted at the sender of JOIN message |
| $S_i$ | Sink i |
| $T_i$ | Tree rooted at Sink $S_i$ |
| $\sigma_i$ | Average node density of temporary tree rooted at $S_i$ |
| $h_i$ | Height of the sub-tree rooted at $S_i$ |
| $SH_i$ | Schedule length of tree $T_i$ |
| $SH_{bal}$ | Balanced Schedule Length |

Table 1: Notations used in HTSTSN Algorithm

### 3.1 Estimation of Schedule Length

1. The sinks $S_1$ and $S_2$ send HELLO packets in the network one by one. So, every node knows $d_1$ and $d_2$. The nbr_table is also created.

2. The nearest sink is selected as temp-home-sink. That is, temp-home-sink is $S_1$ if $d_1$ = min($d_1$,$d_2$), else $S_2$ is temp-home-sink. The height $h$ of every node is set to min($d_1$,$d_2$).

   Thus nodes are divided into two different disjoint sets. First set consists of the nodes who have selected $S_1$ as temp-home-sink and those who selected $S_2$ are in the second set. Nodes in both the sets will perform following two steps.

3. Starting from leaf nodes, every node selects one node as temp-parent. The temp-parent is nearer to temp-home-sink compared to the given node. In other words, given node can send packets to its temp-home-sink via temp-parent.

4. Every node sends JOIN message to its temp-parent to inform that it is selected as a parent. The flow of JOIN messages takes place in bottom-up manner. That is, it starts from the leaf nodes. Every non-leaf node at height $h$ sends JOIN message to its parent only after it overhears JOIN from all the nodes at height $h + 1$ with respect to the same temp-home-sink.

   The fields present in the JOIN message are as follows: nbrcount, ndcount and height $h_t$. The nbrcount is the sum of neighbors of all the nodes present in the subtree rooted at the sender node. The ndcount is the count of nodes present in the sub-tree rooted the sender node. The height $h_t$ is the height of the subtree rooted at given node. The given node considers the nodes belonging to the

same temp-home-sink as itself while calculating nbrcount and ndcount.

5. Every sink $S_i$ receives JOIN messages from its children. Every sink $S_i$ calculates average node density ($\sigma_i$) and height of temporary tree ($h_i$). The average node density ($\sigma_i$) is ratio of sum of neighbors of all the nodes present in the tree and total number of nodes present in the tree. The height of temporary tree is the maximum of heights of its sub-trees.

   To study effect of density ($\sigma$) and height ($h$) on schedule length ($SH$), DICA([5])) is executed for different combinations of $\sigma$ and $h$. It is found that following expression represents relationship between $SH$, $\sigma$ and $h$.

$$SH = (0.2 * \sigma * h) + (0.3 * h) + (2 * \sigma) - 5 \quad (1)$$

6. Every sink $S_i$ estimates $SH_i$ using equation 1. Sink $S_1$ sends $SH_1$ to $S_2$ and vice versa. Each of them calculates $SH_{bal}$. The $SH_{bal}$ is an average of $SH_1$ and $SH_2$.

7. If $SH_i > SH_{bal}$, following two sub-steps are performed by $S_i$:

   (a) The value of $h_{bal}$ is estimated. The value of $h_{bal}$ represents the required height so that $SH_i$ equals $SH_{bal}$ for given $\sigma$. The nodes at hight greater than $h_{bal}$ are asked to move to a different tree.

$$h_{bal} = \frac{SH_{bal} - (2 * \sigma_i) + 5}{(0.2 * \sigma_i) + (0.3)} \quad (2)$$

   (b) The $LOAD\_BAL\_REQD$ message is flooded in the tree rooted at $S_i$. Flooding is done by $S_i$. Following fields are present in the message: $\sigma_1$, $\sigma_2$, $h_1$, $h_2$, $SH_1$, $SH_2$, $SH_{bal}$ and $h_{bal}$.

   (c) The nodes receiving $LOAD\_BAL\_REQD$ message attempt to shift to a different tree using tree switching process described in the next sub-section.

8. If $SH_i \leq SH_{bal}$, sink $S_i$ floods $NO\_BAL\_REQD$ message in its tree. Nodes receiving $NO\_BAL\_REQD$ message understand that they need not move to a different tree. They broadcast a message termed as $SINK\_CONFIRMED$.

### 3.2 Tree Switching

Let us assume that $S_1$ floods $NO\_BAL\_REQD$ message and $S_2$ floods $LOAD\_BAL\_REQD$ message. Every node present in the tree rooted $S_2$ will receive $LOAD\_BAL\_REQD$ message. The $NO\_BAL\_REQD$ message is received by all the nodes in tree of $S_1$. Consider that a node $n$ is present in tree rooted at $S_2$. If its height is less than or equal to $h_{bal}$, it would not change the tree. Else it will switch to a different tree using pseudo-code mentioned in Algorithm 1. The shifting of nodes to tree $T_1$ will start from the boundary separting the two trees and will continue towards the left boundary of the entire area.

At the end, when all the nodes decide their trees (i.e. sinks), joint scheduling & tree formation DICA([5]) is executed for slots and parents selection.

**Algorithm 1:** Pseudo-code for Tree Switching Process

```
try_again:
if distance_from_boundary = 0 then
    sch_est = SH_1 + neibor_count_S_1 + 1
else
    sch_est = recd_sch_est + neibor_count_S_1 + 1
if sch_est < SH_bal AND nodes_sink1_count > 0 then
    home_sink = S_1
    broadcast HOME_SINK_MODIFIED message with field sink_modified = 1 and value of sch_est
else
    if sch_est < bal_sch_len AND nodes_sink1_count == 0 then
        wait until few HOME_SINK_MODIFIED messages are overheard from nodes nearer to S_1
        goto try_again
    else
        home_sink = S_2
        broadcast HOME_SINK_MODIFIED message with field sink_modified = 0;
```

The pseudo-code of Algorithm 1 is explained using Figure 1 in next few paragraphs. In Figure 1, two sinks $S_1$ and $S_2$ are present. There are total 218 nodes, numbered from 0 to 217. The area is divided into two subregions, Region 1 and Region 2. Both the regions have one sink present in the center (nodes filled with black color). The Region 1 has nodes 0 to 48. The Region 2 has nodes 49 to 217.

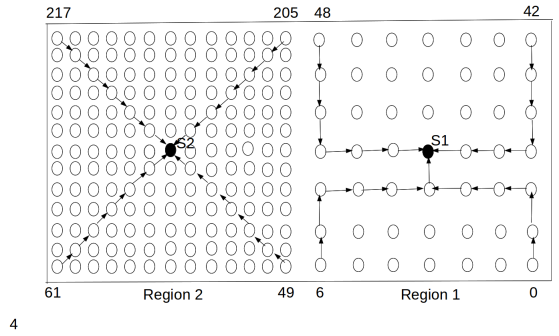



Figure 1: Sample node deployment to illustrate tree switching process

In the figure, sample tentative trees are shown. Not all the edges are shown. The tree $T_1$ is made from nodes of Region 1 and tree $T_2$ is made of nodes of Region 2. The schedule lengths of $T_1$ and $T_2$ are $SH_1$ and $SH_2$ respectively. As Region 2 is denser than Region 1, $SH_2$ would be higher than $SH_1$.

Sinks $S_1$ and $S_2$ would flood $NO\_BAL\_REQD$ and $LOAD\_BAL\_REQD$ messages respectively in their trees. Every node of Region 1 would broadcast $SINK\_CONFIRMED$ message.

The nodes 49,62,75,88,101,114,127,140,153,166,179, 192 and 205 are at the boundary of Region 1 and 2. So, first if condition would be true for them. They would overhear $SINK\_CONFIRMED$ message. They are at one hop distance from Region 1. As a result, they will consider schedule length of tree rooted at $S_1$ as $SH_1$. Then each of those nodes would estimate the resulting schedule length if it would join tree of $S_1$ i.e. $sch\_est$.

The $sch\_est$ is sum of $SH_1$, $neibor\_count\_S_1$ and 1. Initially, schedule length of tree rooted at $S_1$ is $SH_1$. Assume that 5 neighbors of given node have switched to $S_1$. Each of them will require one slot to transmit their packets. So, $SH_1$ should increase by 5. Here $neibor\_count\_S_1$ is represents the number of neighbors of given node who have moved to $S_1$. The given node would require 1 slot to transmit its packet. So, finally '1' is added.

The given node would switch to $S_1$ if the estimated schedule length is less than $SH_{bal}$ and there is at least one node nearer to $S_1$ in its neighborhood ($nodes\_sink1\_count > 0$). This is reflected in compound condition mentioned in second if in pseudo-code. The node would also broadcast $HOME\_SINK\_MODIFIED$ message (with $sink\_modified$ flag set to 1)to inform its neighbors that it has changed the tree. If the estimated schedule length is less than $SH_{bal}$ but there is no node nearer to $S_1$ is neighborhood, given node would wait for its neighbors to switch to $S_1$.

If neither the estimated schedule length is less than $SH_{bal}$ nor there is at least one node nearer to $S_1$ in its neighborhood, node would not switch to a different tree. But it would stick to $S_2$. Still it would broadcast $HOME\_SINK\_MODIFIED$ message (with $sink\_modified$ flag set to 0) to inform its decision to neighbors.

Nodes of Region 2 other than 62,75,88,101,114,127,140,153,166,179,192 and 205 are not at the boundary i.e. more than one hop distance from Region 1. They would not hear $SINK\_CONFIRMED$ message from nodes of Region 1. But they would hear $HOME\_SINK\_MODIFIED$ messages from neighbors. Each $HOME\_SINK\_MODIFIED$ message contains new schedule length of tree rooted at $S_1$. Given node waits to receive multiple such messages. It selects the largest value of new estimated schedule length and assigns to variable $recd\_sch\_est$. The same variable is used to calculate $sch\_est$ (else part of first if).

## 4 Correctness of HTSTSN Algorithm

In this section, various proofs are presented to show that the HTSTSN algorithm works correctly.

**Lemma 4.1.** *In aggregated convergecast, the schedule length ($SH$) of tree T depends on its average node density ($\sigma$) and height ($h$).*

*Proof.* It is always required that the time-slot assignment must be collision-free in nature. That is, when a node transmits a packet in the assigned time-slot, the receiver of the packet should not be receiving or overhearing from a different node in the same time-slot. As wireless devices are generally half-duplex, the receiver can not even transmit in the same slot.

Let us denote number of neighbours of receiver as $x$ and every node is assigned one time-slot. When a time-slot is assigned to the transmitter node, the time-slot should be different than the slots used by those $x$ neighbors of the receiver. Because if the transmitter transmits to the receiver in any of those $x$ slots, packets sent from the transmitter would collide with the packets sent by some neighbor of the receiver.

If node deployment is dense, nodes would have large number of neighbors. That is, value of $x$ will be high for the receiver. Thus to create collison-free schedule, the number of unique time-slots required would

increase. As every node selects a time-slot considering that collision does not occur at the receiver, dense neighborhood would result in increase of the total slots used to schedule the entire network. That is, the schedule length is going to increase.

In aggregated convergecast, it is desired that parent should first receive packets coming from the children. Then send its own packet. This would allow the parent to perform aggregation and send the aggregated packet in the same TDMA cycle with the children. Thus assignment of time-slot should start from the leaf nodes and progress towards the sink. Suppose, there are $p$ nodes in the path from node to the root, $p$ time-slots are needed (one for each node). Thus as length of the path increases, the count of slots used to schedule the entire path increases. In other words, number of slots required to schedule a tree depends on height of the tree. The tree height is distance from the farthest leaf.

From above discussion, it can be concluded that in aggregated convergecast network, schedule length depends on node density and height of the tree.

□

**Lemma 4.2.** *The HTSTSN algorithm ensures that when actual trees are formed, every node gets at least one path to the selected home-sink.*

*Proof.* If given node wants to switch to a different home-sink $S_i$, it would switch if following two conditions are true: (i) The new estimated schedule length of tree $T_i$ would be less than balanced schedule length. (ii) There is at least one node present in the neighborhood of the given node such that the neighbor node has selected sink $S_i$ as home-sink and is at a smaller hop distance from $S_i$ compared to the given node. These two conditions are reflected in if statement as: $sch\_est < SH_{bal}\ AND\ nodes\_sink1\_count > 0$.

If new estimated schedule length is smaller than balanced schedule length but condition (ii) mentioned above is not satisfied, the given node decides not to change its home-sink.

When actual tree formation is initiated by sink $S_i$, the given node must have at least one node in neighborhood which is part of tree rooted at $S_i$ and at a smaller hop-distance from $S_i$. So, that node may be selected as parent. If that potential parent node has selected sink $S_i$ as home-sink from begining, it means it has received 'HELLO' packet from sink $S_i$. Thus that node is able to find a path to sink $S_i$. If potential parent node has switched to sink $S_i$ from some other sink, it would switch only if it has a neighbor which is nearer to sink $S_i$. In any case, sink $S_i$ is reachable through potential parent. Thus the given node would always find at least one path to sink $S_i$ when actual tree formation takes place. □

**Lemma 4.3.** *The HTSTSN algorithm ensures that when actual trees are formed, their schedule lengths remain balanced.*

*Proof.* As mentioned earlier, if given node wants to switch to a different home-sink $S_i$, it would switch if following two conditions are true: (i) The new estimated schedule length of tree $T_i$ would be less than balanced schedule length. (ii) There is at least one node present in the neighborhood of the given node such that the neighbor node has selected sink $S_i$ as home-sink and is at a smaller hop distance from $S_i$ compared to the given node. These two conditions are reflected in if statement as: $sch\_est < SH_{bal}\ AND\ nodes\_sink1\_count > 0$.

If the given node is just one hop distance from tree $T_i$, it would use $SH_i$ to estimate new schedule length of tree $T_i$. The value of $SH_i$ is calculated by sink $S_i$ itself based on average node density and height of the tree. To form collision free schedule, number of neighbors of the give node already joined tree $T_1$ is added into $SH_1$. Then finally '1' is added to take into account transmission slot consumed by the given node. Accordingly, new estimated schedule length $sch\_est$ is calculated as $SH_1$ + neibor_count_$S_1$ + 1. If it is less than $SH_{bal}$ (i.e. average of initial estimate of $SH_1$ and $SH_2$), node would attempt to switch to tree $T_i$.

If the given node is more than one hop distance from tree $T_i$, it is likely that some other nodes around the given node have already switched to tree $T_i$. When a node switches to a different tree, it broadcasts $SINK\_MODIFIED$ message. The message contains new estimated schedule length of that tree. The given node may receive multiple $SINK\_MODIFIED$ messages. Thus multiple values of new estimated schedule lengths of tree $T_i$. To be on safer side, node selects the maximum of all the received values as new estimated schedule length of tree $T_i$. It is denoted as $recd\_sch\_est$. Now $neibor\_count\_S_1$ + 1 are added in $recd\_sch\_est$ to estimate schedule length of tree $T_i$ if the given node joins $T_i$. If it is less than $SH_{bal}$, node would attempt to switch to tree $T_i$

Thus the given node would switch to new tree only if new estimated schedule length of that tree is smaller than $SH_{bal}$. As a result, schedule length of node's tree is reduced but the schedule length of the other tree does not increase beyond balanced schedule length ($SH_{bal}$).

□

| Sr. No. | $p_1$ | $p_2$ | $\sigma_{dev}$ |
|---|---|---|---|
| 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 2: Simulation Scenarios

| Parameter | Value |
|---|---|
| Area | 200m X 200m |
| Topology | Random |
| Radio Range | 30m |
| Transmission Power | 0.66W |
| Receive Power | 0.395W |
| Sleep Power | 0W |
| Packet Generation Rate | 1 packet per 10 seconds |
| Simulation Time | 5000sec |

Table 3: Simulation Parameters

## 5 Simulation Results

### 5.1 Simulation Design

All the simulations are performed using Network Simulator 2 (NS-2.35). The nodes are deployed in a region of 200m x 200m like in Figure 1. In Figure 1, node deployment is in grid. But for simulations, random node deployment is used. The region is divided into 2000 x 2000 grid points. Any two neighboring horizontal or vertical grid points are at a distance of 10m. There are two sub-regions. Each is of size 100m x 200m.

Node deployment is probabilistic in nature. The probability that a node is present at a grid point in Region 1 ($p_1$) is 0.3 and probability for the same in Region 2 ($p_2$) is varied from 0.3 to 0.9 as shown in Table 2.

In Table 2, four different scenarios are presented. For each scenario, density deviation ($\sigma_{dev}$) is calculated as explained next. Consider that there are $m$ nodes in the network. Let $\sigma_i$ be the neighbor count of node $i$, then average density $\sigma$ is defined as follows:

$$\sigma = \frac{\sum_{i=1}^{m} \sigma_i}{m} \tag{3}$$

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

$$\sigma_{dev} = \sqrt{\frac{\sum_{i=1}^{m} (\sigma - \sigma i) * (\sigma - \sigma i)}{m}} \tag{4}$$

As seen from Table 2, density deviation varies from 3 to 6. From scenario 1 to 4, density of nodes in Region 1 remains fix. But density of nodes in Region 2 increases. So, overall density deviation increases from scenario 1 to 4. The performance of the proposed algorithm is studied with reference to density deviation. In simulation results, X-axis in graphs is density deviation. Table 3 summarizes various simulation parameters.

To keep the evaluation robust, three different types of sink placements are used. In Figure 2, sink placements are illustrated. In the first case, one sink is present in the center of each region. In the second case, each sink is present at diagonal corner. Lastly, sinks are very near to each other in the third case. Simulation results are generated for each case of sink placement with random deployment of sensor nodes as explained in previous paragraphs.

In graphs, results are plotted against density deviation. For every scenario, five different simulation runs are performed. The node deployment is done randomly and varied from one run to other. Each point in graph is an average of values obtain in five different runs. The error bars indicate corresponding standard deviation.

The performance of the HTSTSN algorithm is compared with following three different existing algorithms.

- Hop count based extension of [5]
- LBR (Load Balanced Routing)[8]
- SMTLB (Spanning Multi Tree Load Balanced routing) [12]

### 5.2 Performance Measures

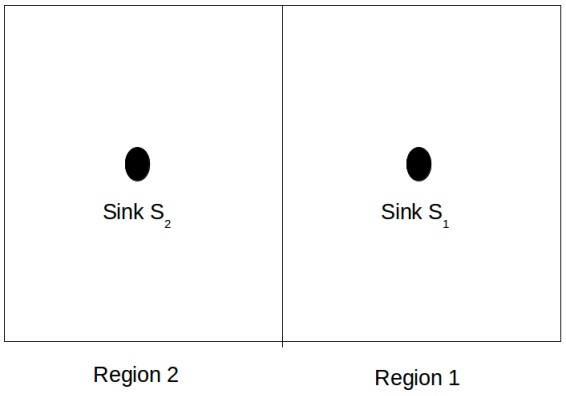
Different performance parameters for evaluation of the proposed algorithm are explained in this sub-section. The notations used are as follows: The symbol $T_i$ means tree spanning Region i. Its schedule length is denoted as $SH_i$. The density of nodes of Region i is denoted as $\sigma_i$. Total number of sensor nodes is denoted as $m$.

- Percentage Difference in Schedule Length ($SH_{diff}$) : It indicates the gap between $SH_1$ and $SH_2$. It is defined as follows:

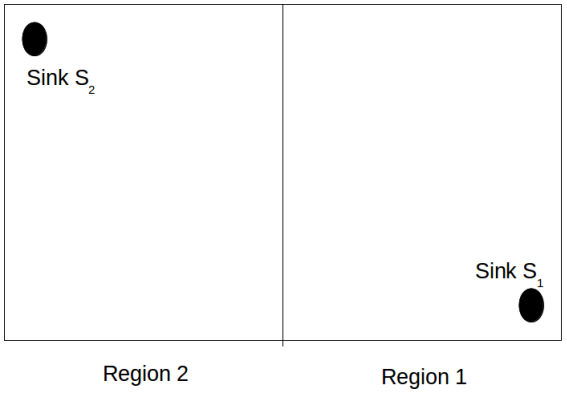
$$SH_{diff} = \frac{\mid SH_1 - SH_2 \mid}{maximum(SH_1, SH_2)} * 100$$

- Maximum Schedule Length (SH) : It is overall schedule length of the network. It is defined as follows:

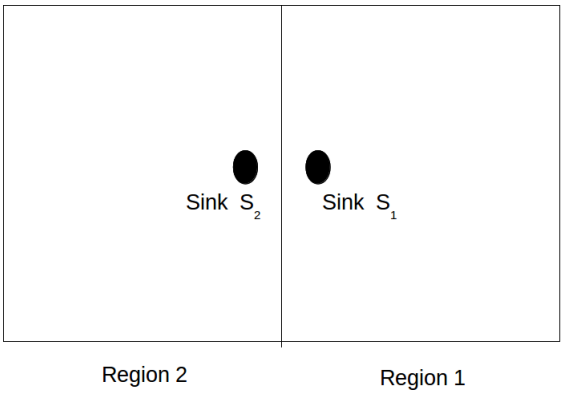
$$SH = maximum(SH_1, SH_2)$$




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




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




(c) Sinks are nearby

Figure 2: Deployment of Sinks

- Control Overhead ($CO$): It is total number of packets generated during control phase. It is the duration from network initialization till beginning of data transfer. It involves tasks like sending of HELLO packets, packets required for load balancing and scheduling & tree formation.

- Energy Consumption During Control Phase ($E_{ccons}$): It is average energy consumed per node due to transfer of control packets by node. It is measured in mJ (milli Joules). Let initial energy of all nodes be $E_{init}$. At the end of control phase, residual energy in node i be $E_{cresi_i}$. Average energy consumption ($E_{ccons}$) during control phase is defined as follows:

$$E_{ccons} = \frac{\sum_{i=1}^{m} E_{init} - E_{cresi_i}}{m}$$

- Energy Consumption During Data Phase ($E_{dcons}$): It is energy consumed due to transfer of data packets. It is also measure in mJ (milli Joules). At the end of control phase, residual energy in node i be $E_{cresi_i}$. At the end of data phase, residual energy in node i be $E_{dresi_i}$ Average energy consumption ($E_{dcons}$) during data phase is defined as follows:

$$E_{dcons} = \frac{\sum_{i=1}^{m} E_{cresi_i} - E_{dresi_i}}{m}$$

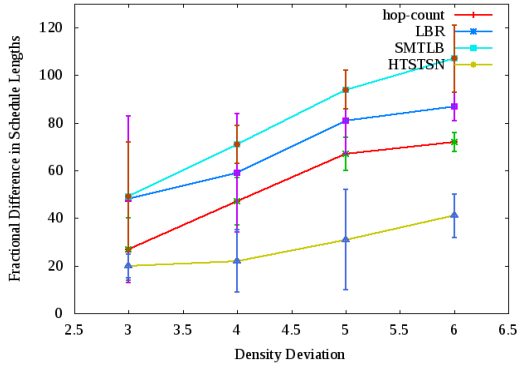
The HTSTSN algorithm is compared with hop-count based approach, LBR algorithm and SMTLB algorithm. The SMTLB algorithm is a centralized algorithm. It is implemented as a C language program by us. Only tree formation and slot assignment are implemented for SMTLB. So, density difference, schedule length difference and maximum schedule length are calculated for SMTLB. As control and data packets are not generated, remaining performance measures are not derived for SMTLB algorithm.

### 5.3 Results & Discussion

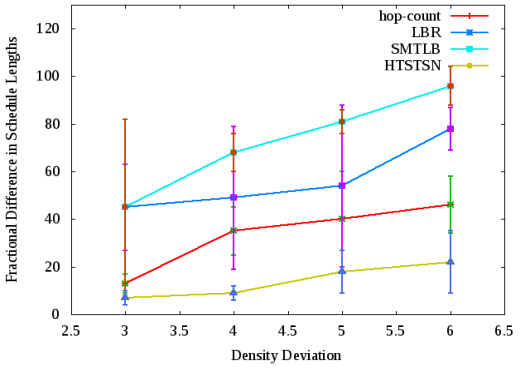
In this sub-section, simulation results and related analysis is presented.

#### 5.3.1 Schedule Length Difference

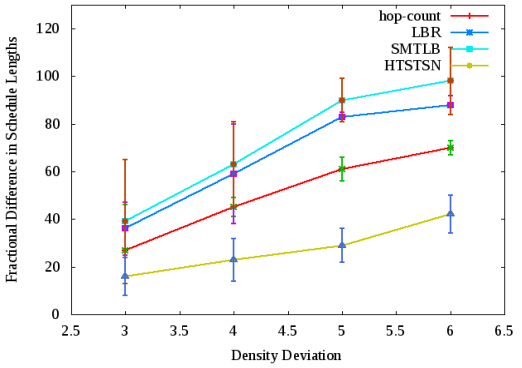
In Figure 3, the graphs of Fractional Schedule Length Difference v/s. Density Deviation are presented. It is observed from the graphs that, as density deviation increases, the difference in schedule length also increases.




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




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




(c) Sinks are nearby

Figure 3: Dependency of Fractional difference in Schedule Lengths on Density Deviation

| | Density Deviation ($\sigma_{dev}$) | | | |
|---|---|---|---|---|
| Sink Positions | 3 | 4 | 5 | 6 |
| Center | 26 | 55 | 35 | 36 |
| Diagonal | 46 | 74 | 13 | 39 |
| Nearby | 40 | 47 | 52 | 40 |

Table 4: Percentage Improvement in Schedule Length Difference

Increase in density deviation means increase in density of nodes in Region 2, but density of nodes in Region 1 is constant. Thus increase in density deviation results in increase in node density of tree $T_2$ compared to that of tree $T_1$. As a result, $SH_2$ increases without much change in $SH_1$. Thus, difference in schedule length increases.

The HTSTSN algorithm results in the least schdule length difference between the two trees. In HTSTSN, nodes move from tree $T_2$ to tree $T_1$ until their estimated schedule lengths become balanced. So, the difference between $SH_1$ and $SH_2$ is also reduced.

The average percentage improvement in schedule length difference achieved by HTSTSN algorithm is summarized in Table 4. The improvement is calculated with reference to the second best performing algorithm. Here the second best performing algorithm is 'hop-count' in all three cases i.e. sinks in center, sinks at diagonal corners and sinks nearby. The percentage improvement ranges between 26% to 74%.

The other three algorithms do not perform as good as HTSTSN algorithm due to reasons explained in next few paragraphs.

The SMTLB algorithm is aimed at balancing workload of sub-trees. We have two sinks $S_1$ and $S_2$. Assuming that sink $S_1$ has $x$ next-hop neighbors and sink $S_2$ has $y$ next-hop neighbors. As the first step, $x$ neighbors will select sink $S_1$ as parent and $y$ neighbors will select sink $S_2$ as parent. Thus total $x + y$ sub-trees are initiated.

It is considered in SMTLB that different nodes have different workloads i.e. number of packets generated by the node per second. The algorithm progresses in top-down fashion. That is, the sub-trees grow gradually from sink to leaf nodes. The sub-tree with the least work-load is expanded first. The workload of a subtree is the sum to packets generated by all the nodes part of that sub-tree. From the available nodes in range, the node with the least workload is selected to join the sub-tree under consideration.

In our simulation setup, in all three different sink de-

ployments, Region 2 has higher node density than Region 1. We have assumed that all the nodes have the same workload. So, here total workload of a sub-tree is the total number of nodes part of the sub-tree. As the sub-trees expand in top-down fashion (i.e from sink to leaf nodes), not all the nodes have option to join any of the two sub-trees. The nodes nearer to sink $S_1$ join the least loaded sub-tree of $S_1$. The nodes nearer to sink $S_2$ join the least loaded sub-tree of $S_2$. Only the nodes arond the boundary of the two regions may have sub-trees of both $S_1$ and $S_2$ available. As a result, not many nodes of Region 2 join the tree rooted at $S_1$. As a result, tree $T_2$ has large number of nodes and it is denser than tree $T_1$. So, $SH_2$ remains higher than $SH_1$. Thus the schedule length difference in SMTLB remains higher than HTSTSN algorithm.

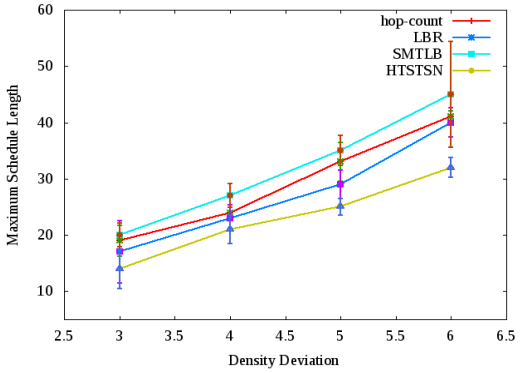
The parent selection in hop-count based approach takes place based on distance from the sink node. That is, given node joins the tree rooted at the sink at the least hop-distance. All the nodes of Region 2 join tree $T_2$ and nodes of Region 1 join tree $T_1$. As explained earlier, as tree $T_2$ is denser than tree $T_1$, $SH_2$ remains higher than $SH_1$. So, the schedule difference in hop-count based approach remains higher than HTSTSN algorithm.

In LBR algorithm, for each sink $S_i$, given node finds the ratio '$r_i$' of number of neighbor nodes of the sink and hop distance between given node and sink $S_i$. The given node joins the tree $T_i$ rooted at the sink $S_i$ for whom the ration '$r_i$' is maximum. As Region 2 has denser node deployment than Region 1, most of the nodes of Region 2 join tree $T_2$. Only those nodes of Region 2 which are far from sink $S_2$ such that $r_2 > r_1$, join tree $T_1$. As very few nodes of Region 2 join tree $T_1$, $SH_2$ remains higher than $SH_1$. Thus, LBR algorithm also is not able to reduce the difference between schedule lengths.

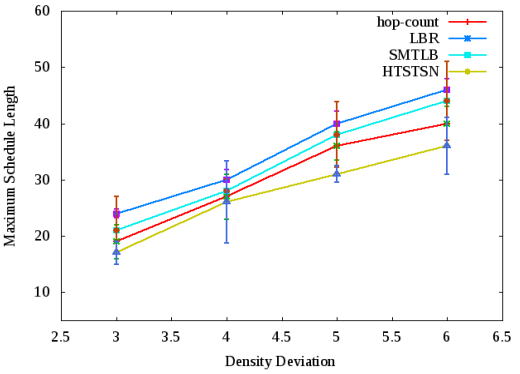
### 5.3.2 Maximum Schedule Length

The graphs of Max. Schedule Length v/s. Density Deviation are shown in the Figure 4. As defined earlier, maximum schedule length is max($SH_1$,$SH_2$). The HTSTSN algorithm results in the least schedule length difference compared to the other three algorithms. As a result, the maximum schedule length as achieved by HTSTSN algorithm is the least. In our simulation setup, $SH_2 > SH_1$. As in HTSTSN algorithm, nodes of tree $T_2$ switch to tree $T_1$, $SH_2$ goes down and $SH_1$ goes up. But $SH_1$ does not increase beyond average of original values of $SH_1$ and $SH_2$. Thus max($SH_1$,$SH_2$) goes down.

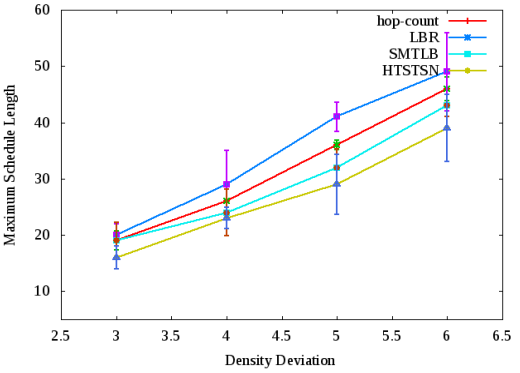
The maximum schedule length increses with increase in density deviation. It is explained earlier how




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




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




(c) Sinks are nearby

Figure 4: Dependency of Max. Schedule Length on Density Deviation

| | Density Deviation ($\sigma_{dev}$) | | | |
|---|---|---|---|---|
| Sink Positions | 3 | 4 | 5 | 6 |
| Center | 17 | 9 | 18 | 20 |
| Diagonal | 10 | 23 | 13 | 20 |
| Nearby | 16 | 14 | 14 | 24 |

Table 5: Percentage Improvement in Maximum Schedule Length

the schedule length difference increases with increase in density deviation. The maximum schedule length is max($SH_1$,$SH_2$). In our simulation setup, increase in density deviation means increase in density of nodes in Region 2. That is, increase in $SH_2$. As $SH_2$ increases, value of max($SH_1$,$SH_2$) also increases.

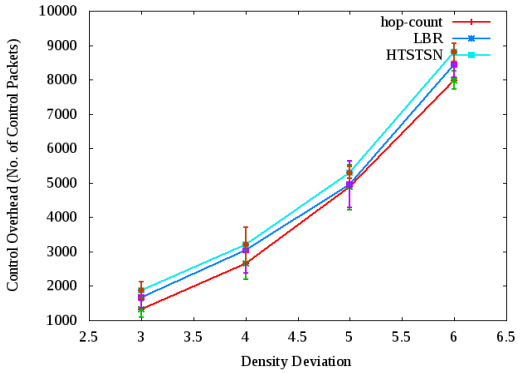
The average percentage improvent in maximum schedule length achieved by HTSTSN algorithm is summarized in Table 5. The improvement is calcuated with reference to the second best performing algorithm. The second best performing algorithm is LBR when sinks are in center, SMTLB when sinks are at diagonal corner and hop-count based approach when sinks are nearby.

### 5.3.3 Control Overhead

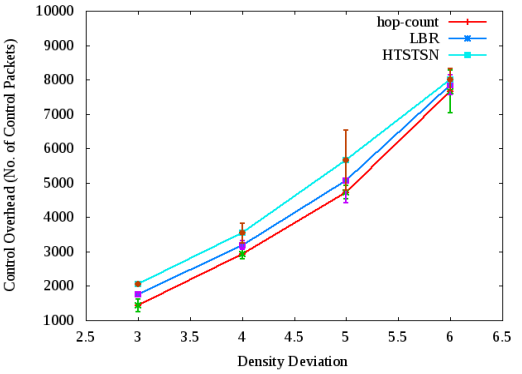
In Figure 5, graphs of Control Overhead v/s. Density Deviation are presented. It is seen from the graphs that HTSTSN algorithm results in higher control overhead than the other two algorithms. As SMTLB is implemented as a stand-alone application program (control and data packets are not simulated), its control overhead is not calculated.

The HTSTSN algorithm uses control messages for three different objectives: (i) estimation of schedule length of each tree (ii) shifting of nodes from one tree to the other (iii) selection of slot and parent. The other two algorithms use control messages only for selection of slot and parent. As HTSTSN algorithm uses more number of control messages, its control overhead is higher than the other algorithms.

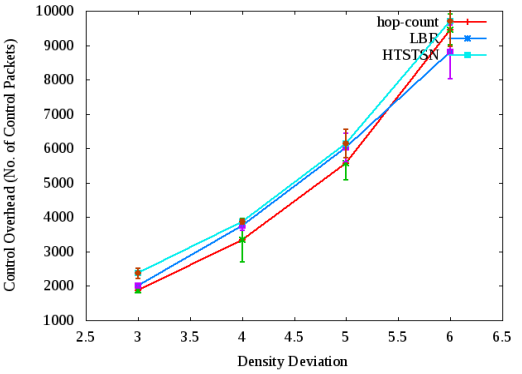
The control overhead increases with increase in density deviation. As explained earlier, increase in density deviation means increase in density of Region 2. Thus number of nodes in tree $T_2$ increases with increase in density deviation. As number of nodes increases, the number of required slots also increases. So, the control overhead for slot and parent selection also increases. Thus with increase in density deviation, control overhead increases.




(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: Dependency of Control Overhead on Density Deviation

| | Density Deviation ($\sigma_{dev}$) | | | |
|---|---|---|---|---|
| Sink Positions | 3 | 4 | 5 | 6 |
| Center | 10 | 9 | 7 | 3 |
| Diagonal | 12 | 10 | 6 | 4 |
| Nearby | 10 | 11 | 5 | 3 |

Table 6: Percentage Increase in Energy Consumption during Control Phase

#### 5.3.4 Energy Consumption during Control Interval

In Figure 6, the graphs for Energy Consumption during Control Interval v/s. Density Deviation are presented. The average energy consumed by nodes during control interval is directly proportional to the control messages exchanged i.e. control overhead. The nature of graphs in Figure 6 is same as in Figure 5. So, more explanation is not given.

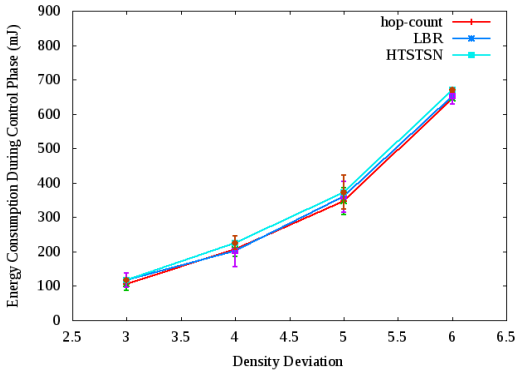
The control phase in TDMA-based tree networks does not take place too often. The control phase takes place just after network deployment so that logical topology can be formed. Then data transfer phase takes place. In the data transfer phase, sensed readings are transferred to the sink. When there is a considerable change in the topology (i.e. many nodes are deleted or new nodes are added), tree and schedule maintenance is required. Thus control phase takes place again. Thus as control phase does not takes place very frequently, the related energy consumption also does not put much burden on the nodes.

The average percentage increase in control energy consumption suffered by HTSTSN algorithm is summarized in Table 6. The difference is calculated with reference to the best performing algorithm which is hop-count based approach in all three cases. It is seen from Table 6 that maximum increase in energy consumption is by 12%.

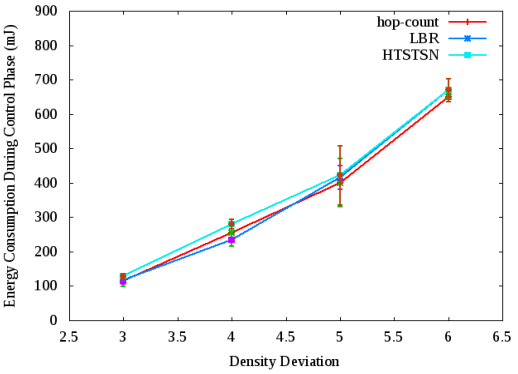
#### 5.3.5 Energy Consumption during Data Interval

The energy consumption during Data Interval remains the same in all the three algorithms. So, corresponding graphs are not shown. All the nodes transmit the packets at the same (1 packet every 10 seconds). In all the three algorithms (i.e. hop-count, LBR and HTSTSN), one hop neighborhood of every node remains the same. So, average number of children per node also remains the same in all the four algorithms. The network is assumed to use aggregated convergecast.

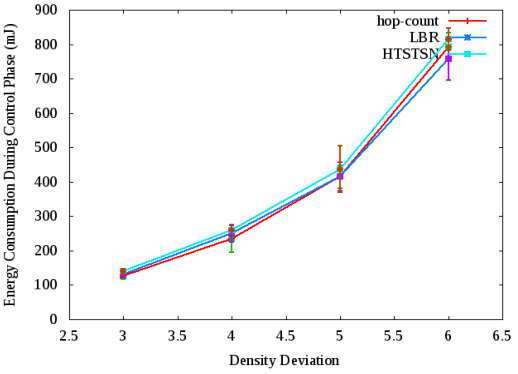
Thus as every node transmits 1 packet per 10 seconds and number of packets received are also almost same in all the three algorithms, the energy consumed




(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: Dependency of Energy Consumption during Control Phase on Density Deviation

during data transmission phase is same in all the three algorithms. Of course, it increases as density deviation increases because, as network becomes denser, average number of neighbors increases. Thus node receives more packets. So, more energy is consumed in packet reception. But for specific value of density deviation, all the three algorithms result in the same data energy consumption.

## 6 Conclusion

In this work, problem of scheduling & tree formation in the case of two-sink sensor networks is examined. When nodes are not deployed in a uniform manner across the area, some regions have higher node density compared to the other regions. As a result, the trees of dense regions have higher schedule length than the trees passing through the sparse regions. If schedule length of a tree is large, every node of that tree has to wait for long time to get its transmission turn.

We have proposed an algorithm termed as HTSTSN (Heuristics based Tree Switching in Two-sink Sensor Networks) to ensure that the two sink-rooted trees have almost equal schedule lengths. Thus all the nodes of the network wait for almost the same time to get transmission opportunity.

The performance of the HTSTSN algorithm is evaluated by varying density deviation of the network. It is found that the HTSTSN algorithm results in smaller schedule length difference and smaller maximum schedule length. It results in around 10% higher control energy consumption. But as control phase does not take place frequently, this additional energy consumption can be accepted because of advantages like reduction in schedule length difference and maximum schedule length. If sensor nodes are deployed in a region where enough sun-light is available, solar energy may be used instead of traditional battery. In that case, this extra energy consumption may not be an issue at all. The energy consumption during data interval remains the same for all the three algorithms.

Thus it can be concluded that the proposed HTSTSN algorithm results in reduction in overall schedule length and schedule length difference between the trees without affecting the network lifetime much.

## References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] M. Wafa et. al,“Energy-Efficient Scheduling in WMSNs",INFOCOMP Journal of Computer Science, Vol. 8, Issue 1, p. 45-54, March 2009.

[7] 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.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] 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.

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