Available online at www.sciencedirect.com
SCIENCE DIRECT®
Ad Hoc Networks xxx (2006) xxx–xxx
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Spanning tree based algorithms for low latency and energy efficient data aggregation enhanced convergecast (DAC) in wireless sensor networks

S. Upadhyayula, S.K.S. Gupta *

Department of Computer Science and Engineering, Fulton School of Engineering, Arizona State University, Tempe, AZ 85287, United States

Received 28 February 2005; received in revised form 26 March 2006; accepted 22 April 2006

Abstract

Many wireless sensor networks (WSNs) employ battery-powered sensor nodes. Communication in such networks is very taxing on its scarce energy resources. Convergecast – process of routing data from many sources to a sink – is commonly performed operation in WSNs. Data aggregation is a frequently used energy-conversing technique in WSNs. The rationale is to reduce volume of communicated data by using in-network processing capability at sensor nodes. In this paper, we address the problem of performing the operation of data aggregation enhanced convergecast (DAC) in an energy and latency efficient manner. We assume that all the nodes in the network have a data item and there is an *a priori* known application dependent data compression factor (or *compression factor*), γ, that approximates the useful fraction of the total data collected.

The paper first presents two DAC tree construction algorithms. One is a variant of the Minimum Spanning Tree (MST) algorithm and the other is a variant of the Single Source Shortest Path Spanning Tree (SPT) algorithm. These two algorithms serve as a motivation for our Combined algorithm (COM) which generalized the SPT and MST based algorithm. The COM algorithm tries to construct an energy optimal DAC tree for any fixed value of α ($= 1 - \gamma$), the *data growth factor*. The nodes of these trees are scheduled for collision-free communication using a channel allocation algorithm. To achieve low latency, these algorithms use the β-constraint, which puts a soft limit on the maximum number of children a node can have in a DAC tree. The DAC tree obtained from *energy minimizing* phase of tree construction algorithms is restructured using the β-constraint (in the *latency minimizing* phase) to reduce latency (at the expense of increasing energy cost). The effectiveness of these algorithms is evaluated by using energy efficiency, latency and network lifetime as metrics. With these metrics, the algorithms' performance is compared with an existing data aggregation technique. From the experimental results, for a given network density and data compression factor γ at intermediate nodes, one can choose an appropriate algorithm depending upon whether the primary goal is to minimize the latency or the energy consumption.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Convergecast; Spanning trees; Data aggregation; Energy-efficiency; Wireless sensor networks

* Corresponding author. Tel.: +1 480 965 3806; fax: +1 480 965 2751.
E-mail addresses: sarma@asu.edu (S. Upadhyayula), sandeep.gupta@asu.edu (S.K.S. Gupta).

1570-8705/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2006.04.004

1. Introduction

One important function of many wireless sensor networks (WSNs) is to gather data from hostile or remote environments. It is expected of such networks to work untended for a long duration. Due to limited energy resources, the above requirements put constraints on the energy usage. Examining various functionalities of sensor networks, communication can be singled out as one function that devours big share of the energy resources. A review of the literature suggests that many attempts have been made to minimize energy consumption during communication [1–7]. The process of routing data from all the nodes to one node (called a sink node) is called a convergecast. In WSNs the energy cost for transmitting data far exceeds the cost of processing it [19]. Data aggregation is an energy conservation technique which tries to reduce the volume of data communicated by collecting local data at intermediate nodes and forwarding only the result of an aggregation operation, such as min and max, towards the sink node. Since a convergecast operation usually follows a broadcast operation [6], the path taken by a broadcast packet is also used for aggregating data in the convergecast. However, research shows that performing data aggregation along this routing path is not energy efficient [5].

In [6], authors study the general data aggregation problem and propose several (suboptimal) data aggregation techniques. The general data aggregation problem is – given m sources and one sink in a n node network ($m < n$), find a minimum weight subgraph that includes all sources. This is a well known NP-complete problem, known as the Steiner Tree Problem (STP) [8]. There are many approximation algorithms for solving STP [9–13]. From sensor networks' perspective, in the literature there are several heuristic approaches to solve this problem [6,7].

The problem addressed in this paper is an important special case of the general data aggregation problem in which *all* the nodes in the network are source nodes. We refer to the task of collecting data from all the nodes in the network while performing data aggregation at intermediate nodes as a *DAC* (Data Aggregation enhanced Convergecast) operation. Some heuristics addressing efficient implementations of DAC operations are presented in [14,15]. Performance metrics vary depending on the system requirements. Energy, latency [5–7,15], reliability [5,15] and network lifetime [3,14] are few that are found in the literature. Node placement models were studied to find their effect on energy consumption in WSNs [6,16].

This paper addresses the problem of optimal implementation of DAC operations incorporating constraints such as data compression at intermediate nodes. Data compression factor $\gamma \in [0,1]$ is used to denote the effectiveness of aggregation in reducing the data volume in comparison to the volume of the input data at an intermediate node. Under these constraints, this paper presents solutions for minimum energy data aggregation with emphasis on reducing the latency. Two algorithms are proposed, one based on Minimum Spanning Tree (MST) and another based on Shortest-Path spanning Tree (SPT), to motivate the combined SPT–MST (COM) algorithm which generalizes the MST and the SPT algorithm and tries to construct energy optimal DAC trees for all values of α ($= (1 - \gamma)$), the data growth factor.

In order to ensure collision-free data aggregation, the nodes are scheduled using a combined Code-Division Multiple Access (CDMA) and Time-Division Multiple Access (TDMA) based channel allocation algorithm. Latency of the proposed algorithms is improved by introducing a concept called β-constraint. This rule places an upper (soft) limit on the number of children each node in the tree can have. This limit is soft in the sense that β-constraint is violated in cases where there is a risk of a node being left out of the tree.

The performance of the algorithms is tested against some of the existing techniques [5,14,15] on randomly generated graphs. Network lifetime is also observed as a by-product while running these algorithms. Empirical observations are made with regards to answering questions such as – given a network of source nodes and data growth factor (α), the tree constructed by which algorithm is recommended? and, what is the suggested value for β?

The paper is organized as follows. In Section 2, a formal definition of the problem this paper deals with is presented. It also identifies the performance metrics used in this work. In Section 3, system model used in the paper is given. In Section 4, the tree construction and channel allocation algorithms are explained. Section 5 describes the simulation environment and Section 6 discusses the experimental results. The paper is concluded in Section 7.

2. Problem description and design goals

In this paper, we address the problem of constructing energy and latency efficient DAC trees

taking into account the degree of data aggregation that an application permits. Formally, for a given network (of n sensor nodes and a base station (sink node), with *each* sensor node having a unit data to be sent to the sink node) and an (average) compression ratio γ (determined by the application specified aggregation operator), the problem is to determine the data aggregation enhanced convergecast (DAC) tree structure (determined by combination of tree type (MST or SPT) and β value (range) that would be better for a desired primary optimization metric: total energy or latency.

Total energy spent by a network in communication is defined as the sum of the energy spent by each node in the network. Since WSNs have low energy resources, their efficient use is highly desired. More so because, typically, energy consumed for communication in wireless networks is high (proportional to r^{λ} where r is the range of communication and λ is between 2 and 4) [4]. Therefore, to minimize this high communication costs, data aggregation algorithms adopt multi-hop routing for their data [5,14].

Total latency is the time taken to perform one instance of a DAC operation. Low latency indicates the promptness with which data is delivered to the base station. Hence, it is a desired communication feature in wireless networks. Collision-free communication saves energy which would otherwise be wasted in retransmissions and hence, should become a consideration for the scheduling algorithm. Network lifetime is defined as the longevity of network in terms of number of rounds of DAC operations performed before the failure of a predefined number of nodes. Since WSNs are generally deployed in inaccessible environments, long network lifetime is highly desired.

The objective in this paper is to simultaneously minimize total energy consumption and achieve low latency during the entire DAC operation. In the process, variation of network lifetime is observed. Since collisions result in data loss and/or retransmissions, which wastes energy, collision avoidance is an inherent design goal in this problem.

3. System model

The following system model is used in this paper. The network consists of n wireless sensors, each containing a transceiver with a maximum transmission range. This network is represented by a undirected graph $G(V, E)$ where nodes in the set V represent the sensor nodes and an edge (i,j), where $i \in V$ and $j \in V$, in the set E denotes that the two nodes i and j are in the communication range of each other. The weight $w(i,j)$ associated with an edge $(i,j) \in E$ indicates the cost (in terms of energy consumption) of communication between the two vertices i and j.

This work adopts the use of multi-hop transmissions in its communication model. For this purpose, a tree is constructed, i.e., a parent–child relationship is established among various nodes. In-network aggregation at intermediate nodes usually results in reducing the size of data (in comparison to cumulative data obtained from all its children and the data generated by node itself) that is forwarded by an intermediate node to its parent. We refer to the fraction by which the data obtained from the children node gets compressed due to the aggregation operation performed at the parent node as the *compression factor* γ. An example of 100% data *compression* ($\gamma = 1$) case would be *max*, *min* or *avg* temperature in a room. An example of 100% data *concatenation* ($\gamma = 0$) case would be temperatures at various points in the environment.

If a node i transmits a packet of size $data_i$ to a node j then the size of the packet at the node j (denoted as $data_j$) after receiving the packet from the node i is modeled as

$$data_j \leftarrow data_j + (1 - \gamma) \times data_i. \quad (1)$$

As mentioned above, if $\gamma = 0$ then all the data from individual nodes are appended without compressing them. If $\gamma = 1$ then there is a complete compression of all the individual node data.

3.1. Assumption of system model

All the nodes in the network are static and are loosely synchronized to enable TDMA scheduling. The topology graph constructed from the network is connected. Weights associated with all the edges are positive and hence, there are no negative weight cycles. A node can adjust its power level depending on its requirements. This means that one node can be neighbor of the other, but vice versa need not be true. In this paper, weight of an edge and energy for communicating between the two nodes connected by the edge are used interchangeably. The energy spent for transmitting a data packet from node i to node j is the same as that from node j to node i. Data aggregation in the network is done along a tree. Although, the maximum transmission

power of a node is assumed to be sufficiently large so as to reach every other node in the network, a node employs power control techniques to use signal strength which is sufficient to reliably communicate with its parent node. Every node generates a constant sized data packet (k units). Each node is assigned a *time slot* during which it has to transmit its data. Communication is assumed to be done in time slots. Every node in the network is equipped with only one transceiver. Therefore, multiple receptions by a node in the same time slot is not possible. In addition to time slots, the nodes use CDMA codes for communication. When a node is not communicating, it goes to power-save mode, thereby expending negligible amount of energy in running the transceiver circuitry. The scheduling algorithm assumes that a node does not transmit until it receives data from all its children. All the nodes in the network use a fixed compression factor while collecting data. In this paper, DAC operation is evaluated in terms of latency of a round of DAC operation. A *round* is defined as the collection of one data unit from every node in the network and delivering the resulting aggregated data to the sink node. The latency of one round of DAC operation is the time taken starting from the first data packet transmission along the tree till the last data packet reception by the sink. In every round, each parent in the tree will wait till it receives data from all its children. A node after participating in a round of a DAC can immediately participate in its next round.

3.2. Energy model

Let E_{elec} be the electrical energy required to run the circuitry in the transceiver. The following generic model is used to account for the energy consumed during communication. If a node i is transmitting k bits of data to a node j, then the amount of energy spent in this process by node i, $E_{Tx}(i,j,k)$, and node j, $E_{Rx}(j,k)$, is given as:

$$\begin{aligned} &E_{Tx}(i,j,k) = E_{\text{elec}} \times k + E_{\text{amp}}(i,j) \times k, \quad \text{and} \\ &E_{Rx}(j,k) = E_{\text{elec}} \times k, \end{aligned} \tag{2}$$

where $E_{\text{amp}}(i,j)$ is the minimum amplification energy per bit required to reliably transmit data from node i to node j (measured in nJ/bit). The value of amplification energy required between any two nodes depends upon the radio communication channel characteristics. In many works, $E_{\text{amp}}(i,j)$ is approximated using a distance based formula $E_{\text{amp}}(i,j) = \epsilon_{\text{amp}} \times dist(i,j)^{\lambda}$, where ϵ_{amp} is the amplification energy required to transmit a bit a unit distance, $dist(i,j)$ is the distance between nodes i and j, λ is a positive integer typically having values, depending on the communication medium, from 2 to 4 [3,4]. The formula in Eq. (2) is very general and encompasses non-distance based models for energy consumption. For example, the $E_{\text{amp}}(i,j)$ can be assigned the value obtained from actual in-field measurements conducted by the two nodes i and j. Note that, we are assuming that the nodes employ power control in order to optimize power expended for communication and to minimize interference to other concurrent communication.

The total energy needed to transmit k bits of data from node i to node j would be denoted by $E(i,j,k)$, where

$$E(i,j,k) = E_{Tx}(i,j,k) + E_{Rx}(j,k). \tag{3}$$

Nodes are assumed to have two states: a high energy consuming *on* state and a low energy consuming *sleep* state. A node is in on state only when it is either sending or receiving data. For sake of simplicity, we assume that the energy consumed in the sleep state is negligible. Slot duration are expressed in units of time duration for transmitting one bit. If a slot is of duration of B bits and the node is only on for receiving or transmitting k bits then it is assumed to spend energy required for transmitting or receiving only k bits instead of B bits.

3.3. β-constraint

Consider any two nodes i and j having a common parent p. If one of these nodes (say node i) has a greater number of children than the other node, then the parent node p may have to wait for a longer duration before receiving messages from node i. If these nodes have roughly the same number of children (i.e. the tree is roughly balanced), then it is *likely* that both node i and node j will be prepared to transmit data at the same time and node p will have to be awake for a shorter duration to collect data from all its children. This is advantageous from the perspective of energy consumption minimization since nodes have to be in high energy consuming *on* state for a shorter duration.[1] For this purpose a new constraint called the β-constraint is introduced:

[1] Nodes are assumed to enter a low energy-consumption *sleep* state whenever they are not sending or receiving data.

Definition 1 (β-Constraint). A node cannot have more than β children unless an intended child node is likely to be left out from the DAC tree.

The β-constraint ensures that a (almost) balanced DAC tree is constructed while ensuring that all the node are included in the tree. Note that, in the resulting DAC tree, some nodes can have fewer than β children and some other nodes may have more than β children. However, in a dense sensor network most of the nodes will have exactly β children – resulting in an almost balanced tree.

Define $\alpha = 1 - \gamma$ as the *data growth factor*. This term gives an indication of the rate at which the data grows as it moves up the tree. Rest of the paper uses α and γ interchangeably. Theoretically, for a fixed α, the value of β for which latency achieves optimal value is discussed in the following section.

3.4. Data latency

Latency is defined as the time required from the start of data transmission by the first source till the last packet is received by the sink in one round of a DAC. The communication in this model takes place using variable length time slots. On the entire tree, the length of a time slot is variable. However, the transmission slot length of the nodes at a particular level in a tree is fixed (as described below). As it will be seen later in the paper (Section 4.5), this assumption simplifies contention-free channel allocation. Note that a parent node's reception time slots are determined by the transmission time slots of its children, i.e., a parent node is scheduled to be listening (on an appropriate channel) whenever its children node are scheduled to transmit.

In this work, the length of the transmission time slot for a node at level i is defined as follows:

Definition 2 (*Length of a time slot at level i*). It is the length of time required to transmit the largest size data packet from any level i node to its parent at level $(i+1)$ in the DAC tree.

For instance, consider a DAC tree of the height h with the root node at level h and the leaf nodes at level 1. And let all the leaves have a unit of data (say of size k) to transmit. Therefore, the length of the transmission time slot for the level 1 is k unit. Further, let $\{n_{i_1}, n_{i_2}, n_{i_3}, \ldots\}$ represent nodes at level i and $\{d_{i_1}, d_{i_2}, d_{i_3}, \ldots\}$ represent the size of data they have to transmit to their respective parents at level $i+1$. Then the length of the time slot for this transmission is given by the time to transmit a data of size D_i, where

$$D_i = \max\{d_{i_1}, d_{i_2}, d_{i_3}, \ldots\}. \tag{4}$$

Define L_i to be the time required for level i nodes to finish transmitting data to the next level. Assuming that β_i is the maximum number of children any node at level $i+1$ has in a given DAC tree, we have

$$L_i \geqslant \beta_i D_i, \tag{5}$$

since each parent has to wait for receiving data from all its β children.

For a DAC tree of height h, the total latency *Latency* is defined as:

$$Latency = \sum_{i=1}^{h-1} L_i. \tag{6}$$

For example consider a complete *β-ary* tree.[2] The latency of a round of DAC operation for a given complete *β-ary* tree can be calculated as follows. For a tree with height (or level) h, the (maximum) data size at level i, $1 \leqslant i < h$, is given as

$$\begin{aligned} D_i &= k\left[1 + (\alpha\beta) + (\alpha\beta)^2 + \cdots + (\alpha\beta)^{(i-1)}\right], \\ &= k\frac{(\alpha\beta)^i - 1}{(\alpha\beta) - 1}. \end{aligned} \tag{7}$$

For instance,

- At level 1 the largest packet size $= k$ $(= D_1)$
- At level 2 the largest packet size $= k[1 + (\alpha\beta)]$ $(= D_2)$
- At level 3 the largest packet size $= k[1 + (\alpha\varpi\beta) + (\alpha\beta)^2]$ $(= D_3)$
- ...
- At level $h-1$ the largest packet size $= k[1 + (\alpha\beta) + (\alpha\beta)^2 + \cdots + (\alpha\beta)^{h-2}]$ $(= D_{h-1})$
- According to the system model, the root node is the sink node and does not sense or transmit any data itself therefore D_h is not defined.

In the case of complete *β-ary* tree:

$$\text{Latency} \geqslant k\beta \sum_{i=1}^{h-1} \frac{(\alpha\beta)^i - 1}{(\alpha\beta) - 1}, \tag{8}$$

$$\geqslant k\beta \frac{1}{(\alpha\beta) - 1}\left(\frac{(\alpha\beta)^h - (\alpha\beta)}{(\alpha\beta) - 1} - (h-1)\right). \tag{9}$$

[2] A complete *β-ary* tree is a *β-ary* tree in which all leaves are at the same level and all the internal nodes have β children.

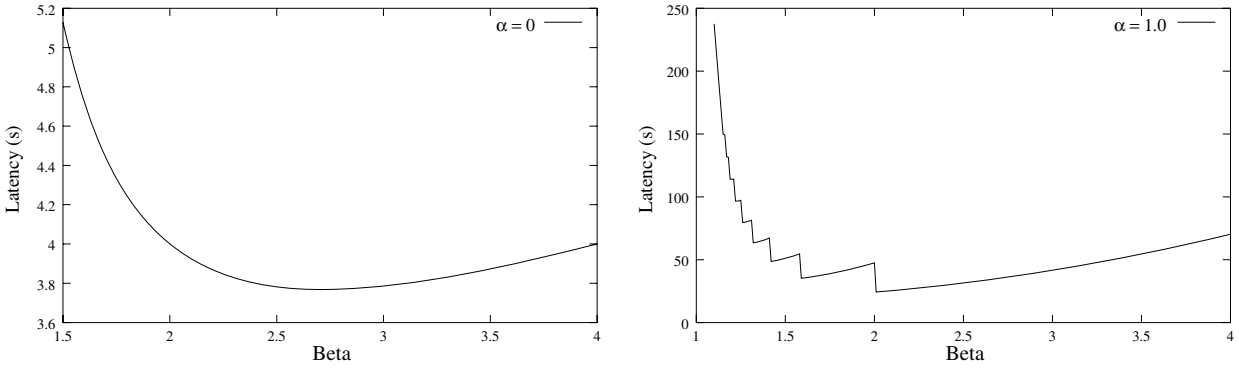


Fig. 1. Plot of theoretical analysis of latency vs β for (a) $\alpha = 0$ and (b) $\alpha = 1.0$.

As shown in Fig. 1, for $\alpha = 0$ the minimum value of latency in Eq. (9) changes to $\lceil \beta \log_\beta N \rceil$ which attains least value for $\beta = e$ (2.76, natural number). When the value of β is restricted to integral values the minimum value is attained for $\beta = 2$ or $\beta = 3$ depending upon the value of N. This derivation assumes that the tree is balanced with each node having β children. It also assumes that each internal node requires β time slots.

3.5. Lifetime of a DAC tree

Lifetime of a DAC tree is defined as the number of rounds of DAC operation that can be performed before the failure of certain percentage of total nodes. A node is said to have failed if, due to depletion of its energy resources, it cannot transmit its data to its parent or if it cannot receive data from at least one of its children. Trees performing DAC operations with minimum energy may not necessarily have a long life. In such trees, it is possible that certain nodes spend more energy than the rest, and therefore, die early. Hence, minimum energy consumption and maximizing life time of network are competing goals. Therefore, in this paper, lifetime of a DAC tree is defined as the failure of 5% of the total nodes of the tree.

3.6. Radio communication model

For communication, each node will be assigned a 3-tuple consisting of two CDMA codes and a time slot. One code is for transmission and another for reception. The time slot indicates the period during which a node has to transmit its data to its parent. Parent node will be prepared to receive data during that time slot. Once a node completes its transmission, it switches itself to the power-save mode until next round of DAC. The reception code of a node is same as the transmission code of all its children. A node does not transmit its data until it has finished receiving data from all its children. Every node fuses α fraction of the data it receives with its own data and the resultant packet is transmitted. This exercise continues till the base station receives data from all its children.

In this paper, Direct Spread Spectrum Sequence CDMA (DSSS-CDMA) codes are used for communication. However, for scenarios where there is more possibility of interference (like large network size, dense networks) frequency hopping can be used as it offers lesser interference. Our channel allocation technique eliminates any packet collisions for transmission using a specific channel, i.e. (time-slot, code) pair, however, there exists possibility of interference from simultaneous communication using different codes. Our channel allocation algorithm takes this into account by considering: (1) the number of codes which can be used, and (2) the communication range of each node. However, we make the simplifying assumption that no packets are lost due to channel bit errors. Hence, the results presented in this paper can be viewed as an upper bound on the achievable performance.

4. Algorithms

In this section algorithms for tree construction are presented. The notations used in this algorithm is explained in Table 1. A channel allocation algorithm for the constructed tree is also presented. Channel allocation determines a time slot for each node and assigns two CDMA codes to each node (one for transmission and one for reception). The

Table 1
Notations for tree construction algorithms

Notations	Definitions	Initial value
S	$\{v \mid v \in V(T)$, where T is the tree$\}$	$\emptyset$
s	Root node	Input
$w(u,v)$	Weight on edge connecting u and v	Input
PQ	Priority Queue of the nodes keyed by their d values	–
$Adj[u]$	Adjacent nodes of u	Input
$Child[u]$	List of child nodes of u	$\emptyset$
$d[u]$	Weight of path to node u from base station (for SPT algorithm)	∞ if $u \neq s$ 0 otherwise
	Minimum weight of any edge connecting node u to the tree (for MST algorithm)	∞ if $u \neq s$ 0 otherwise
$dist[u,v]$	Distance between node u and v	–
$\pi[u]$	Parent node of u	NULL
$\Pi[u]$	Sorted list of alternate parent nodes for node u	$\emptyset$
α	Data growth factor	Input
γ	Data compression factor	$1-\alpha$
$\Gamma(u)$	Time slot for transmission assigned to node u	Φ
β	$\lvert Child[u]\rvert \leqslant \beta$	Input
$\mathscr{T}(u)$	Transmission code of node u	0
$\mathscr{R}(u)$	Reception code of node u	0
$\mathscr{P}(i)$	$\{u \mid u \in V[G]$ and $\mathscr{R}(u) \neq i\}$	$\emptyset$
$\mathscr{C}(i)$	$\{u \mid u \in V[G]$ and $\mathscr{T}(u) \neq i\}$	$\emptyset$
$\mathscr{L}$	Set of leaf nodes	Input
$\{1,\ldots,\Theta\}$	Set of allowed CDMA codes	Input

time slot assigned to a node indicates when to transmit its data to its parent node. Concurrently, the parent node will wake up from sleep state to receive data from its children during this time slot.

4.1. Tree construction

A node can directly transmit its sensed data to the base station (sink node) or route it through other nodes. Examining the energy model in Section 3, it can be concluded that multi-hop routing of data will result in energy savings whenever single hop communication involves transmission over long distances. Hence, in this work, a tree based approach is adopted to solve the problem of convergecast. It is easy to see that a tree based approach can represent both single hop and multi-hop communications. Following are some of the basic definitions we use in tree construction algorithms:

Definition 3. Let path from u to v, $p(u,v) = \{u_0, u_1, u_2, \ldots, u_n\}$ be defined as $\forall i,\ 1 \leqslant i \leqslant n$, $\pi[u_{i-1}] = u_i$, where $u_0 = u$ and $u_n = v$, and $\pi[u_i]$ indicates parent node of u_i.

Definition 4. Weight of a node u, denoted as $d[u]$, is defined as follows:

$$d[u] = \begin{cases} \text{cost of the (known) shortest path from} \\ \quad \text{base station to } u \text{ – SPT algorithm,} \\ \text{cost of shortest link connecting} \\ \quad \text{node } u \text{ to MST – MST algorithm.} \end{cases}$$

Note that (1) $d[u]$ is used as the key value of u when placing the node in a priority queue, (2) "shortest" is in terms of energy cost, and (3) the link costs (weights) are assumed to be symmetric, i.e. $w(i,j) = w(j,i)$, implying that the path costs are also symmetric.

We now motivate our algorithm for tree construction (COM) by first presenting two tree construction algorithms: one based on the Minimum Spanning Tree (MST) algorithm and the other based on the Single Source Shortest Path (SPT) algorithm. Conditions (values of α and β) are given under which performance of each of these algorithms is optimum in terms of energy. Then we present the Combined SPT and MST algorithm, referred to as COM, which works for any value of α.

4.1.1. Rationale for SPT approach

Since each node in the network has some bits to transmit to the base station, it can be viewed as a network with $(n-1)$ sources and one sink (n nodes in the network). If the compression factor is 0% ($\alpha = 1$), then it can be interpreted that the entire data sensed by each node travels to the base station. Let the energy expended for each *packet* be the sum of energy spent by every node involved in its transmission from the source to the base station. From the energy model in Section 3, it can be concluded that shorter the path traveled by the packet and lesser the number of hops, lower is the energy expended. SPT constructs trees based on this notion. A node in a SPT is at a *shortest-distance* away from the root node. Since, weight on the edges in the graph represents energy for communication, SPT also accounts for the number of hops between a node and the base station. Therefore, SPT based approach is used for tree construction when $\alpha = 1$.

Energy spent by a node in communication also depends on the amount of data it is transmitting.

Contribution of a node b at height j to the amount of data transmitted by its ancestor node at height i, $(i < j)$ is given by $(\alpha)^{j-i}k$, where k is the number of bits transmitted by b. For 0% compression ($\alpha = 1$), it can be observed from Eq. (7) that the data transmitted by any node in the tree will travel, uncompressed, along the tree to the base station. Hence, the energy consumed for delivering the data packet originating at a node will be minimized if it is routed to the base station along the shortest (minimum energy) *path* from the node to the base station. Clearly, to minimize the energy cost for the entire DAC operation, the path traveled by a packet from every node should be minimized. This is achieved by a SPT.

4.1.2. Rationale for MST approach

Consider a DAC scenario with 100% ($\alpha = 0$) compression. In this case, a packet transmitted by any node does not exist after one hop. To spend least energy in transmitting a packet, a node should transmit it to its closest (by weight) neighbor (towards the sink node). Total energy consumed in this type of model is given by:

$$E_{\text{Total}} = k \times \sum_{\forall (u,v) \in T} w(u,v), \tag{10}$$

where k is a constant packet traveling along the graph, $w(u,v)$ is the weight of the link between nodes u and v, and T is the DAC tree. In Eq. (10), E_{Total} is minimized only when $\sum_{\forall (u,v) \in T} w(u,v)$ is minimized. This is the problem of finding a minimum (edge-weight) spanning tree.

4.2. SPT based algorithm

The first tree construction algorithm is based on the concept of single source shortest path algorithm. This paper uses Dijkstra's SPT algorithm for this problem. Due to reasons given in Section 3, a constraint is put on the number of children per node in the tree (called the β-constraint). β is an input to the algorithm. If an intended child node is likely to be left out of the tree due to the β-constraint, then it is overlooked. This is to ensure that all the nodes in the network are included in the DAC tree. The notations used in this algorithm is explained in Table 1.

Algorithm 4 is a two-phase algorithm. In the first phase, it constructs a Dijkstra's Single Source SPT which minimizes energy for communication. In the second phase, it restructures the tree trying to balance it so as to reduce latency. By constructing a balanced tree there is a greater chance of multiple simultaneous transmissions, thereby reducing latency. However, a restructured tree may not show any improvement in latency. In such cases, the behavior of network lifetime is observed to find an explanation for this behavior.

For $\beta \geqslant |V|$ Algorithm 4 behaves similar to the Dijkstra's Single Source SPT. It is well-known that upon termination of Dijkstra's Single Source Shortest path algorithm with source node s on a graph $G(V,E)$, the resulting *predecessor subgraph* $G_\pi = (V_\pi, E_\pi)$, where $V_\pi = \{v \in V : \pi[v] \neq \text{NULL}\} \cup \{s\}$ and $E\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$ induced by π values is a shortest-path tree rooted at s [18].

Algorithm 1. Relax(TYPE, u, v)

```
1:  if TYPE = SPT
2:    if d[v] > d[u] + E(u,v,k)
3:      d[v] = d[u] + E(u,v,k)
4:      π(v) = u
5:      Π[v] = Π[v] ∪ {u}
6:  if TYPE = MST
7:    if d[v] > E(u,v,k)
8:      d[v] = E(u,v,k)
9:      π(v) = u
10:     Π[v] = Π[v] ∪ {u}
11: if TYPE = COM
12:   if d[v] > α * d[u] + E(u,v,k)
13:     d[v] = α * d[u] + E(u,v,k)
14:     π(v) = u
15:     Π[v] = Π[v] ∪ {u}
```

Algorithm 2. AdjustTree(u)

```
1:  C = ∅ /* C is a stack */
2:  push(C,u)
3:  while (C ≠ ∅)
4:    u = pop(C)
5:    for each v ∈ Child(u)
6:      push(C,v)
7:    p = π(u)
8:    if (u ≠ s) /* s is base station */
9:      if (|Child(p)| > β)
10:       find t ∈ Π(u) such that |Child(t)| < β
11:       π(u) = t
12:       Child(p) = Child(p) − {u}
13:       Child(t) = Child(t) ∪ {u}
```

Algorithm 3. AdjustWeight(TYPE, u)

```
1:  for each v ∈ Child(u)
2:    if TYPE = SPT
3:      d[v] = d[u] + w(u,v)
4:    if TYPE = MST
5:      d[v] = w(u,v)
6:    if TYPE = COM
7:      d[v] = α * d[u] + w(u,v)
8:    AdjustWeight(TYPE,v) /*Adjust Weight in
      the subtree */
```

Algorithm 4. SPT Based Tree Construction

```
1:  /*Energy Minimizing Phase*/
2:  Initialize(G,s)
3:  S = ∅
4:  PQ = V[G]
5:  while PQ ≠ ∅
6:    u = Extract_min(PQ)
7:    if (u ≠ s)
8:      p = π[u]
9:      Child[p] = Child[p] ∪ {u}
10:   S = S ∪ {u}
11:   for all v ∈ Adj[u]
12:     Relax(TYPE,u,v) /* TYPE is SPT,
      MST or COM (for Combined-SPT–MST) */
13:
14: /*Latency Minimizing Phase*/
15: AdjustTree(s)
16: AdjustWeight(TYPE,s) /* TYPE is SPT,
      MST or COM (for Combined-SPT–MST) */
```

Before discussing the algorithm in further detail, the following terms needs to be explained: `Extract_min` points to the minimum value from a priority queue. `Next_min` is a pointer to the next minimum value in the queue. For any queue, initially, `Next_min` is same as `Extract_min`. `Initialize(G,s)` sets $d[s] = 0$, where, s is the base station and for the rest of the nodes their d value is set to infinity.

Fig. 2 explains the working of the SPT algorithm. An example scenario is presented, where a tree is constructed from the given network. The nodes of the tree are also scheduled (assigned time slots and CDMA codes for communication). Fig. 2(a) shows the network with edge weights representing energy (in nano Joules (nJ)) spent in transmitting one bit of data over a square meter area. The tree, showed as solid lines in Fig. 2(b), shows shortest path (denoted by dark lines) of each node from base station (node s). This tree is the product of *Energy Minimizing Phase* of the algorithm. Energy required for one round of communication in this tree is 44 nJ. This is obtained for a single bit data packet by substituting 100 nJ/bit and $dist(i,j)^2 \times 0.1$ nJ/bit/m^2 for E_{elec} and $E_{amp}(i,j)$ in Eq. (2), respectively. The tree in Fig. 2(c) is the result of **AdjustTree** algorithm of *Latency Minimizing Phase*. It changes the parent of node E from node D to node B. This algorithm only adjusts the tree structure to satisfy *β-constraint*. It does not change the value of energy required to transmit a unit packet to base station. Therefore, $d[u]$ value, where u is any node in the network, is unchanged (e.g., $d[E] = 11$ instead of 12 in Fig. 2(c)). Fig. 2(d) shows the result of running **AdjustWeight** algorithm. It corrects the energy value of each node in the network. This change can be seen by observing the energy value of node E.

The schedules for Fig. 2(b)–(d) (as shown in Fig. 2(e)–(f) respectively) are given before introducing the scheduling algorithm in order to help the reader better understand the reasoning behind the tree construction algorithm. Scheduling algorithm is not enforced until the completion of tree construction. A brief explanation of this algorithm is given here.

Communication latency in Fig. 2(b) is 8 time units. It can be derived as follows. Nodes B and E transmit in the first time slot using transmission codes 2 and 1 respectively. Nodes C and F follow in the second and the third time slots. D concatenates all the data with its own data and transmits a packet of size 4 units in the fourth time slot. Therefore, the length of time required for this communication is 8 time slots. By similar calculation, latency for the tree in Fig. 2(d) is 7 time units. Therefore, this algorithm improves the DAC operation's latency at the expense of energy. In this example, for calculating latency, it is assumed that there is more than one CDMA code available to chose from. The symbol Φ indicates absence of code or time-slot assignment.

4.3. MST based algorithm

Algorithm 5. MST Based Tree Construction

```
1:  /*Energy Minimizing Phase*/
2:  Initialize(G,s)
3:  while PQ ≠ ∅
4:    u = Extract_min(PQ)
5:    if u ≠ s
6:      Child[π(u)] = Child[π(u)] ∪ {u}
7:    for all v ∈ Adj[u]
```

8: **if** $v \in PQ$ and $w(u,v) < d[v]$
9: Relax(MST, u, v)
10:
11: /*Latency Minimizing Phase*/
12: AdjustTree(s)
13: AdjustWeight(MST, s)

Algorithm 5 is the MST based algorithm for DAC tree construction. Similar to the SPT based algorithm, it has two phases: (1) Energy Minimization Phase: that constructs an MST tree, and (2) Latency Minimizing Phase that tries to adjust the number of children a node has in accordance with β constraint.

Working of MST based tree construction algorithm is explained using Fig. 3. Tree resulting from *Energy Minimizing Phase* of Algorithm 5 is given by Fig. 3(b). Energy consumed by this tree to perform one round of DAC is 44 nJ/bit (with E_{elec} and E_{amp} values as mentioned earlier). Fig. 3(c) of this figure is the product of **AdjustTree** algorithm. Fig. 3(d) is the end result of the MST based tree construction algorithm. Latency due to the tree in Fig. 3(b) is 4 time units. Imposing β-constraint with $\beta = 2$ will give a tree whose communication latency is 3 time units. There is an improvement in latency to 3 time units with the energy trade off of 1 nJ/bit.

The **AdjustTree** algorithm rearranges the parent pointers of each node in the network to satisfy the β-constraint.

Note that for $\beta \geqslant |V|$ and $\alpha = 0$, Algorithm 5 results in minimum energy for aggregating data during convergecast. That is because for the above assumption, AdjustTree algorithm will make no changes to the tree constructed by Algorithm 5. The tree obtained in such a case is an MST which ensures that the sum of the weights on edges connecting the nodes is minimum.

Note that, for $0 < \alpha < 1$, neither of the algorithms guarantee minimum energy consumption for the DAC operation.

4.4. Combined SPT–MST algorithm

In this section, we present the combined SPT–MST algorithm which builds a DAC tree for all

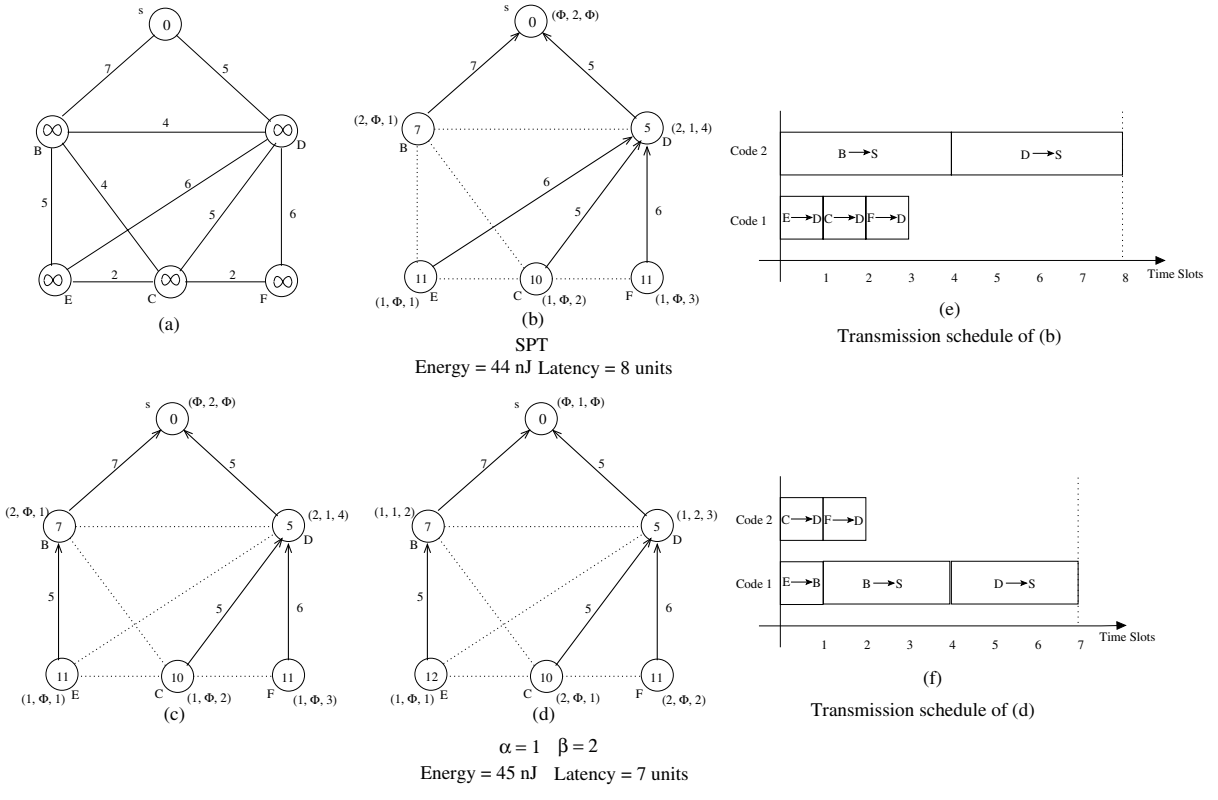


Fig. 2. Example of SPT based algorithm. The three tuple indicate (Transmission Code, Reception Code, Time Slot), the value of Φ indicates that no code has been assigned, and the value within each circle is the $d[\,]$ value for that node.

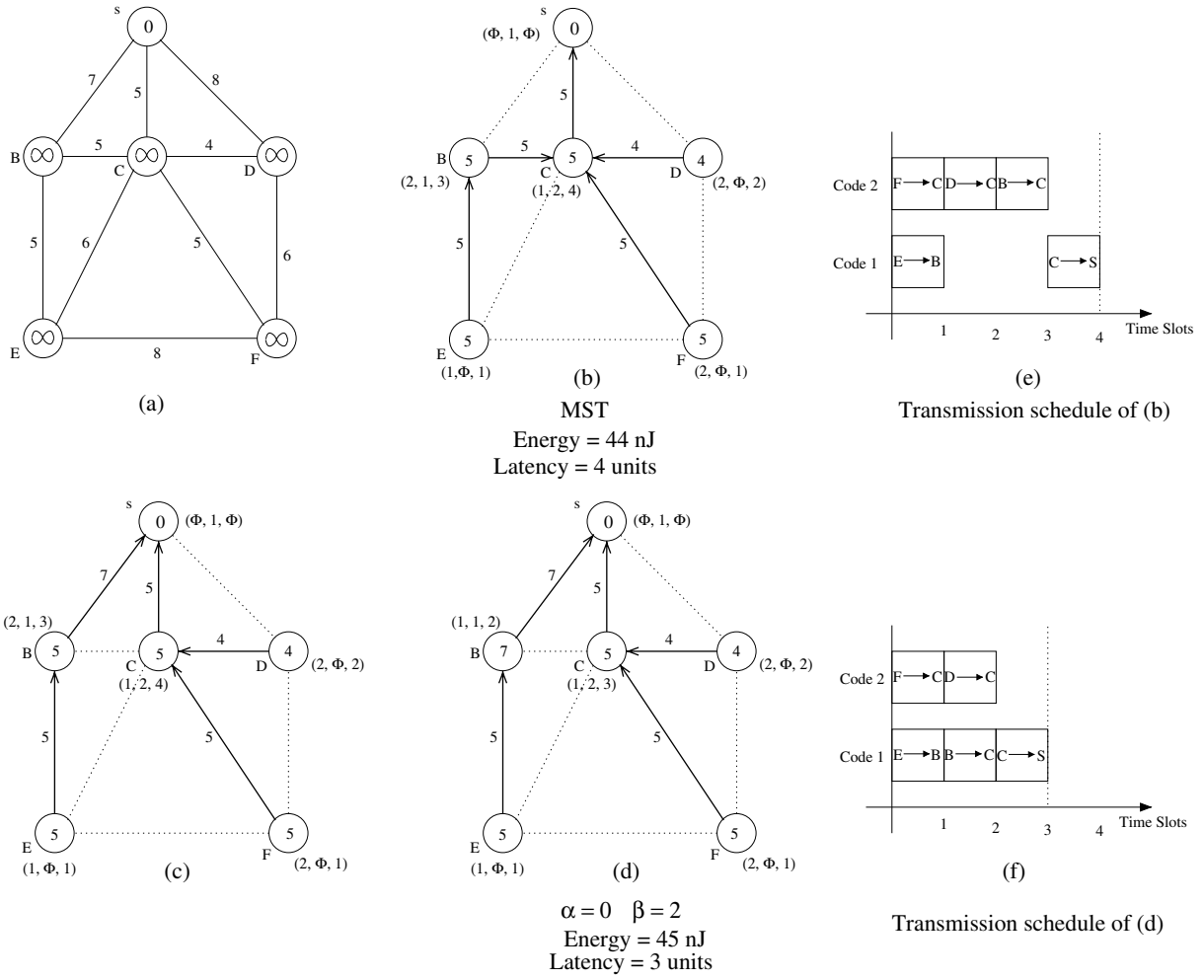


Fig. 3. Example of MST based algorithm. The three tuple indicate (Transmission Code, Reception Code, Time Slot).

values of α. The change that is needed for SPT to operate as the Combined SPT–MST algorithm (COM) is in the relaxation operation (see lines 12–13 of Algorithm 1). For $\alpha = 1$ the relaxation operation is same as that used in the original Dijkstra's SPT algorithm and for $\alpha = 0$ the relaxation operation is same as the original Prim's MST algorithm. In the following we motivate the algorithm. We begin with the following definition:

Definition 5. Assuming path $p(u,v) = \{u_0, u_1, \ldots, u_l\}$, $u_0 = u$ and $u_l = v$, define

$$\begin{aligned} EC(p(u,v),k,\alpha) &\equiv E(u_0,u_1,k) + \alpha E(u_1,u_2,k) \\ &\quad + \cdots + \alpha^{l-1} E(u_{l-1},u_l,k) \\ &= \sum_{i=1}^{l} \alpha^{i-1} E(u_{i-1},u_i,k). \end{aligned} \tag{11}$$

$EC(p(u,v),k,\alpha)$ corresponds to the energy expended for delivering node u's contribution to the aggregated data of (its ancestor) node v along the (DAC tree) path $p(u,v)$ assuming that the size of data originating at node u is k bits and the data compression factor is $(1-\alpha)$. Specifically, for $\alpha = 1$ the $EC()$ path metric is same as the shortest-path metric used by SPT.

COM is a greedy algorithm. At every step, it adds to the DAC tree a node which has minimum estimate of $EC()$-value (d-value) path to a node in the partial DAC tree. Since COM constructs an acyclic graph which covers all the nodes in the network, it constructs a tree. However, it should be noted that the DAC tree constructed by COM's Energy Minimizing phase may not optimal for the intermediate value of α, i.e. when $0 < \alpha < 1$. However, our

simulation results show that COM's DAC tree are energy-efficient compared to SPT or MST tree for non-one and non-zero value of α, respectively.

4.5. Channel allocation

For completeness, channel allocation algorithm is presented in this section. The problem of optimal channel allocation is NP-complete [17]. The algorithm presented here assigns codes with the objective of avoiding collisions. The allocation of channels in this algorithm starts with the leaf nodes and continues along the tree until the root node is reached.

In Table 1, the symbol $\mathscr{P}(i)$ indicates the set of nodes that cannot receive data using the code i in the current time slot. Similarly $\mathscr{C}(i)$ is a set of nodes that cannot transmit data using code i in the same time slot. Stated otherwise, $\mathscr{P}(i)$ and $\mathscr{C}(i)$ indicate the set of nodes that cannot become parent and child, respectively, using code i in the current time slot.

Algorithm 6. NodeInfo$(u, \aleph, \mathscr{L}_{\text{new}}, time)$

$\mathscr{T}(u) = \aleph$
$\Gamma(u) = time$
for all $v \in Adj(u) \wedge E(u, v, 1) \leqslant E(u, \pi(u), 1)$
  $\mathscr{P}(\aleph) = \mathscr{P}(\aleph) \cup \{v\}$
for all $v \in Adj(\pi(u)) \wedge E(v, \pi(u), 1) \leqslant E(v, \pi(v), 1)$
  $\mathscr{C}(\aleph) = \mathscr{C}(\aleph) \cup \{v\}$
if all children of u are scheduled
  $\mathscr{L}_{\text{new}} = \mathscr{L}_{\text{new}} \cup \{u\}$

Algorithm 7 starts assigning two CDMA codes and a time slot to each node in the tree. This algorithm takes as input the set of leaf nodes, $\mathscr{L}$, and set of CDMA codes $\{1, \ldots, \Theta\}$. The algorithm assigns code-time slot pairs (called *channels*) with the objective to avoid interference during communication. While examining a node u for assignment of code $\aleph$ in a given time slot *time*, the algorithm checks two sets: (a) Set $\mathscr{P}(\aleph)$ for presence of $\pi[u]$ and (b) set $\mathscr{C}(\aleph)$ for u. If either of the nodes are present in the respective sets, then the node cannot be assigned code $\aleph$. The algorithm then goes on to check for other codes for assignment. The sets $\mathscr{P}(\aleph)$ and $\mathscr{C}(\aleph)$ contain elements that *cannot* become parent node and child node respectively using code $\aleph$. If a node cannot be assigned any code in a given time slot, then that node should wait for the next time slot for transmission.

Algorithm 7. Channel Allocation$(\mathscr{L}, \{1, \ldots, \Theta\})$

```
1:  L_new = ∅, time = 1
2:  while L ≠ ∅
3:    for all u ∈ L
4:      if (π(u) ≠ s) ∧ (R(π(u)) > 0)
5:        /* Parent of u has been
          assigned code */
6:        ℵ = R(π(u))
7:        if u ∉ C(ℵ) ∧ π(u) ∉ P(ℵ)
8:          NodeInfo(u, ℵ, L_new, time)
9:        else
10:         L_new = L_new ∪ {u}
11:     else
12:       for all 1 ⩽ ℵ ⩽ Θ
13:         if u ∉ C(ℵ) ∧ π(u) ∉ P(ℵ)
14:           R(π(u)) = ℵ
15:           NodeInfo(u, ℵ, L_new, time)
16:           break /* Exit the For loop */
17:       if T(u) = 0
18:         /* Node u could not be scheduled
            in the current time slot*/
19:           L_new = L_new ∪ {u}
20:   L = L_new, L_new = ∅, time = time + 1
21:   for all 1 ⩽ ℵ ⩽ Θ, C(ℵ) = P(ℵ) = ∅
```

Motivation for constructing the sets $\mathscr{P}(\aleph)$ and $\mathscr{C}(\aleph)$ is as follows: When a node is assigned a code $\aleph$ during a time slot, no node in its transmission range can receive data using code $\aleph$ in the same time slot. Therefore, all the neighbors of the assigned node will be added to the set $\mathscr{P}(\aleph)$. Similarly, when a node is receiving data using code $\aleph$ in a time slot, it should not be in the transmission range of other nodes. The set $\mathscr{C}(\aleph)$ is filled using this constraint.

Working of this algorithm is illustrated in Fig. 3. Consider the scenario in Fig. 3(b). Node E is scheduled for transmission in time slot 1 using code 1. Node C, which is the parent of node F, cannot receive data using code 1 since it is in the transmitting range of node E. So, the algorithm assigns code 2 to node F for transmission. Therefore node F will transmit its data in the first time slot. Since node C is receiving data using code 2, node D will use the same code for transmission but in the next time slot. By similar reasoning node C will have a transmission code 1 and reception code 2 and it will transmit its data in the fourth time slot (node B transmits to it in the third time slot).

4.6. Complexity analysis

The SPT based tree construction algorithm begins by calling `Initialize(G,s)`. Its running time is $\mathcal{O}(V)$. Algorithm 1 takes constant time for execution. In Algorithm 4 if PQ is implemented as an array, then `Extract_min` operation takes $\mathcal{O}(V)$. Since the **while** loop runs for $\mathcal{O}(V)$, total number of `Extract_min` operations are $\mathcal{O}(V^2)$. Each edge in the adjacency list $Adj[u]$ is relaxed exactly once in the algorithm. So, Algorithm 1 is executed a total of $|E|$ times. Thus, the running time of the algorithm without AdjustTree, Initialize(G, s) and AdjustWeight code is $\mathcal{O}(V^2 + E)$. This is the running time of the *Energy Minimizing Phase* of the SPT based tree construction algorithm.

In AdjustTree algorithm, the while loop runs exactly once for each node. The running time of the algorithm is same as the running time of lines 8–10 which is $\mathcal{O}(V)$. Therefore, the complexity of AdjustTree algorithm is $\mathcal{O}(V^2)$. AdjustWeight algorithm executes once for every node in the network. The **for** loop runs for $\mathcal{O}(\beta)$ times. The complexity of the AdjustWeight algorithm is $\mathcal{O}(\beta V)$. Therefore, the complexity of Algorithm 4 is $\mathcal{O}(V^2 + E + \beta V)$.

It should be noted that the $\mathcal{O}(V^2)$ is the complexity of AdjustTree algorithm in the worst case scenario. For every node u in the tree, entire list $\Pi(u)$ may not be always scanned. Moreover, not all nodes may be present in $\Pi(u)$. However, tight bound on the complexity of AdjustTree algorithm does not improve the overall complexity of Algorithm 4 because of the complexity of *Energy Minimizing Phase*.

The run time complexity of MST based tree construction algorithm can be determined on similar lines as the SPT based algorithm. Initialize will take $\mathcal{O}(V)$ time. The **while** loop runs for $\mathcal{O}(V)$ time. `Extract_min` operation takes $\mathcal{O}(V)$ time due to array implementation of PQ. The **for** loop will run for $\mathcal{O}(E)$ times because each edge is visited only once. The AdjustTree, as stated above, will take $\mathcal{O}(V^2)$ in worst case scenario. AdjustWeight algorithm will take $\mathcal{O}(\beta)$ time. Running time of this algorithm, therefore, is $\mathcal{O}(V^2 + E + \beta V)$.

Run time complexity of NodeInfo algorithm is $\mathcal{O}(V)$, because a node can have at most all other nodes in its adjacency list. Algorithm 7 runs at least once for every node in the network. If the parent of a node is receiving on a channel, then the node will check if it can transmit on that channel. Line 7 performs this check which, in the worst case, takes $\mathcal{O}(V)$. If the parent of a node has decided on reception channel then the node examines which CDMA code it should choose to avoid interference. Lines 12–15 performs this check which, in the worst case, takes $\mathcal{O}(V\Theta)$. Rest of the code takes constant time. Therefore, the runtime of the algorithm is $\mathcal{O}(V^2\Theta)$.

5. Simulation environment

The simulations are based on the input network graphs generated with varying node densities, number of nodes and β value. In this work, node density is considered as number of nodes per unit area. The underlying assumption in this definition is that there cannot be more than one node per unit area. Node density is computed as $\frac{\text{nos. nodes}}{\text{total area}} \times 100$. Therefore, 100% node density corresponds to one node per unit square area and 10% node density corresponds to one node per 10 unit square area. For simulations, networks with node density varying from 10% to 80% in the increments of 10% are considered. For each node density, network with sizes varying from 50 nodes to 300 nodes are generated. The value of β, an input to tree construction algorithm, is varied from 1 to 3. From the numerous trial graphs generated, it was observed that typically, most of the nodes in MST and SPT will have no more than three children. Therefore increasing β greater than 3 will not help attain significant latency gain. Another simulation test was performed with $\beta > |V|$. The reason for taking such inputs is to observe the performance of the proposed algorithm on sparse and small networks to dense and large networks while varying β for each configuration. The number of codes (Θ) used is 3. The reason is that during our trial runs, we did not see significant improvement in the latency by increasing Θ beyond 3. The energy model used is r^2 energy model and the performance of the proposed algorithms is tested against Pegasis [14] for the above inputs. Performance was measured using the following metrics: total energy consumption, latency and network life

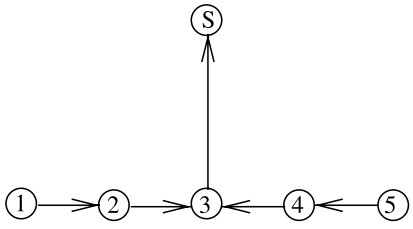


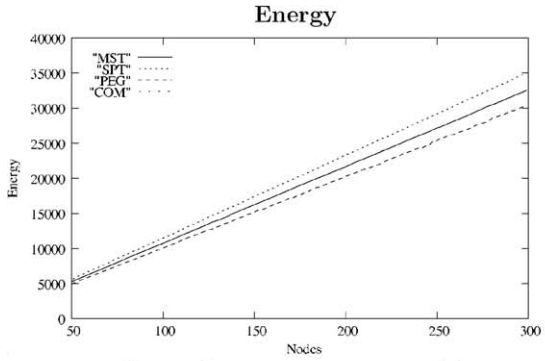
Fig. 4. Working of Pegasis.

time. For a given node density and network size, input data was randomly generated, and algorithms tested. This simulation was performed for 50 different random graphs (for each set of inputs) and the average performance was measured. Values of parameters used in Eq. (2) are, $E_{elec} = 50$ nJ/bit, ϵ_{amp}=100 pJ/bit/m^2. These parameters are same as those used in [3]. Communication range of each node is set to 15 m. The results for lifetime are obtained for the 95% confidence interval.

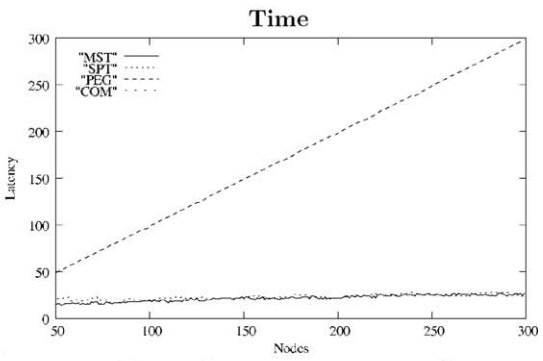
The reason for choosing Pegasis for comparison was its network longevity and better performance

α	Network Density	Priority (Latency or Energy)	Recommended Algorithm	Recommended β
$= 0$		Latency	MST	$\beta = 2$ or 3
		Energy	MST or Pegasis	$\beta = 2$ or 3 - MST
$< 25\%$	low($<30\%$)	Latency	SPT	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$
	Medium(40-60%)	Latency	SPT	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$
	High($>60\%$)	Latency	MST	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$
$\approx 50\%$	low($<30\%$)	Latency	SPT	$\beta = 2$
		Energy	COM	$\beta = 2$
	Medium(40-60%)	Latency	MST	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$
	High($>60\%$)	Latency	MST	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$
$\geq 75\%$	low($<30\%$)	Latency	SPT	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$
	Medium(40-60%)	Latency	MST	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$
	High($>60\%$)	Latency	MST	$\beta = 2$ *or* 3
		Energy	COM	$\beta = 2$

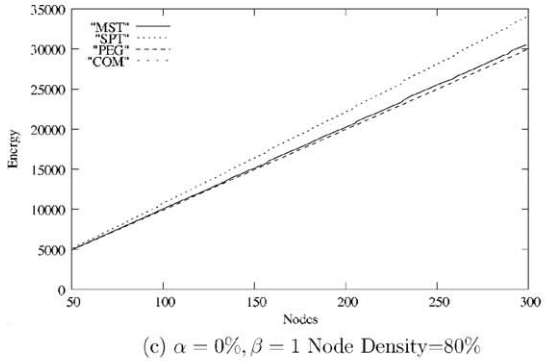
Fig. 5. Table summarizing the experimental results.



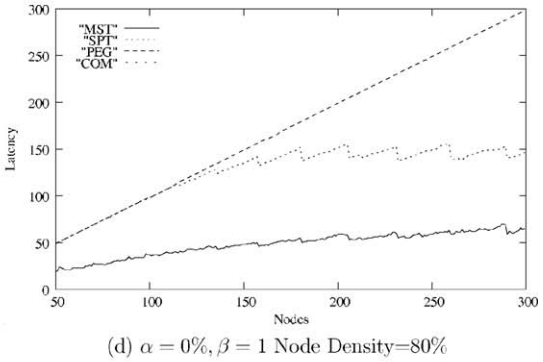
(a) $\alpha = 0\%, \beta = 1$ Node Density=10%



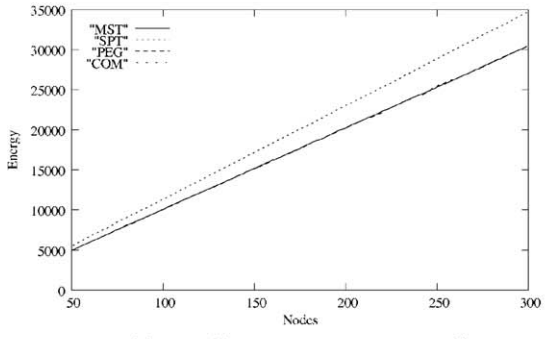
(b) $\alpha = 0\%, \beta = 1$ Node Density=10%



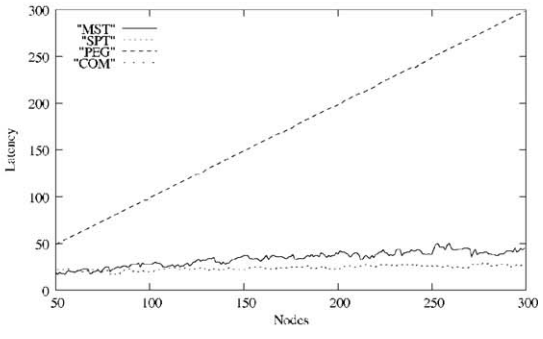
(c) $\alpha = 0\%, \beta = 1$ Node Density=80%



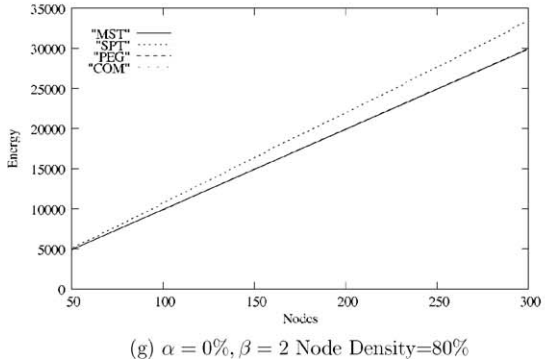
(d) $\alpha = 0\%, \beta = 1$ Node Density=80%



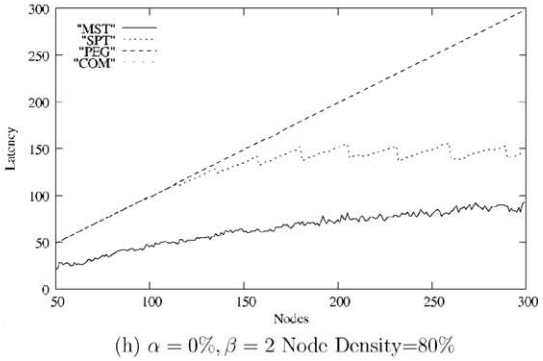
(e) $\alpha = 0\%, \beta = 2$ Node Density=10%



(f) $\alpha = 0\%, \beta = 2$ Node Density=10%

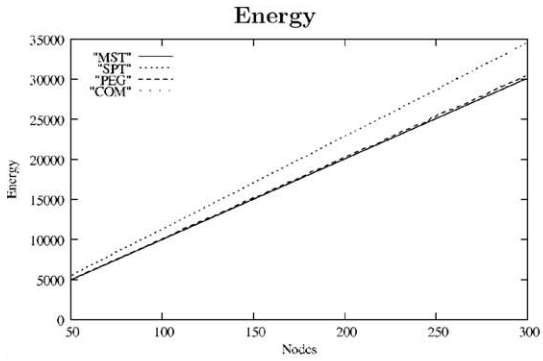


(g) $\alpha = 0\%, \beta = 2$ Node Density=80%

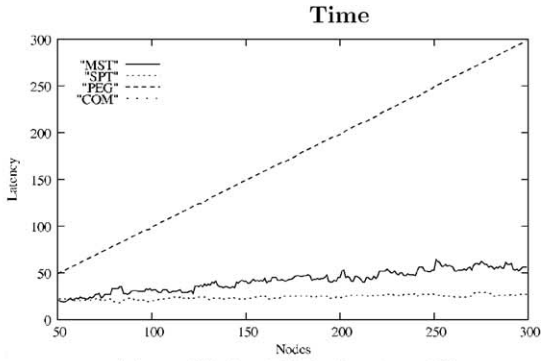


(h) $\alpha = 0\%, \beta = 2$ Node Density=80%

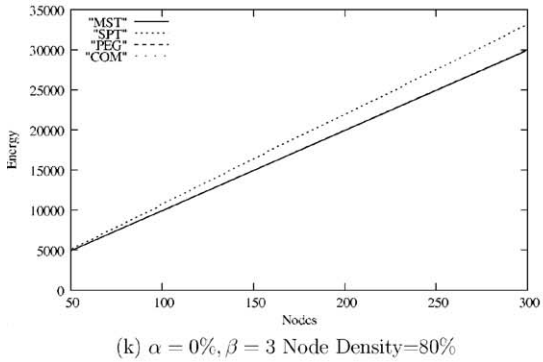
Fig. 6. Behavior of Pegasis, COM, MST-based and SPT-based Algorithms for $\alpha = 0\%$.



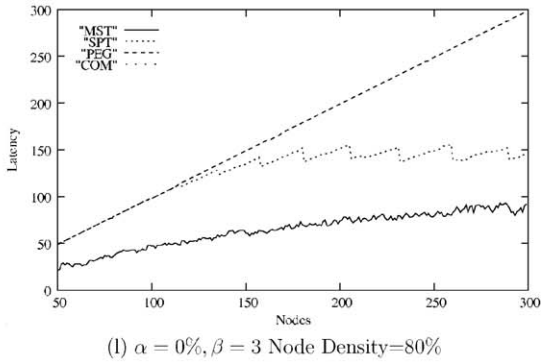
(i) $\alpha = 0\%, \beta = 3$ Node Density=10%



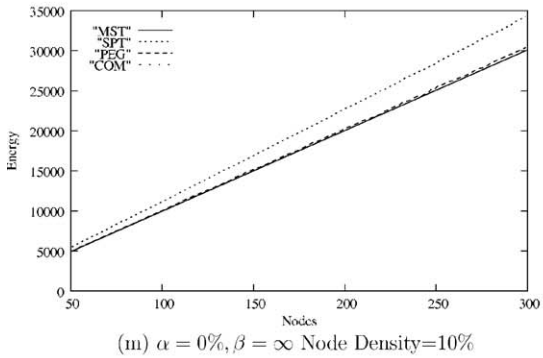
(j) $\alpha = 0\%, \beta = 3$ Node Density=10%



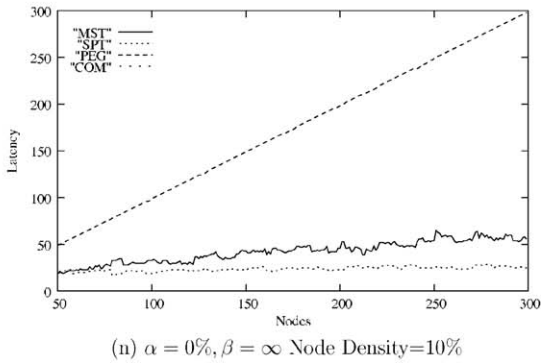
(k) $\alpha = 0\%, \beta = 3$ Node Density=80%



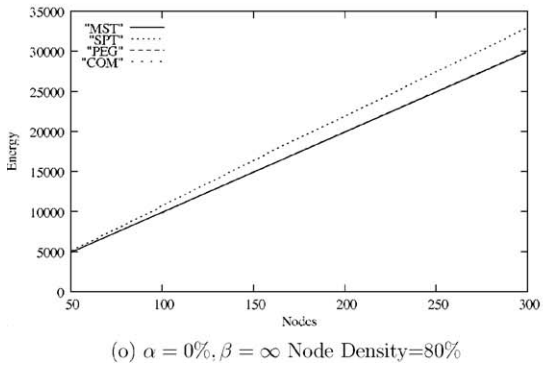
(l) $\alpha = 0\%, \beta = 3$ Node Density=80%



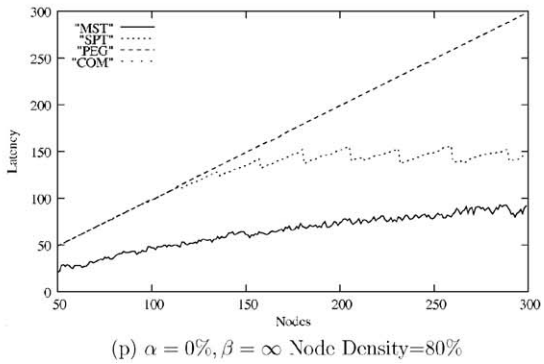
(m) $\alpha = 0\%, \beta = \infty$ Node Density=10%



(n) $\alpha = 0\%, \beta = \infty$ Node Density=10%

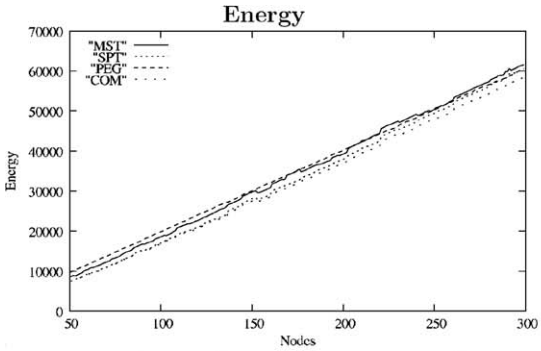


(o) $\alpha = 0\%, \beta = \infty$ Node Density=80%

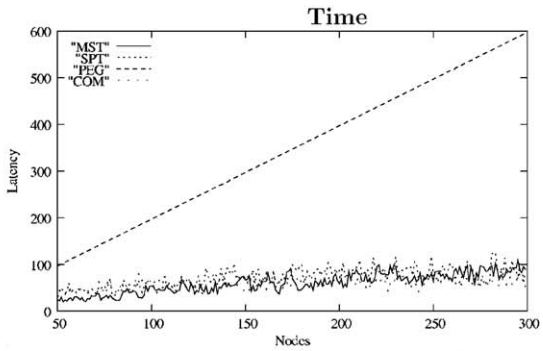


(p) $\alpha = 0\%, \beta = \infty$ Node Density=80%

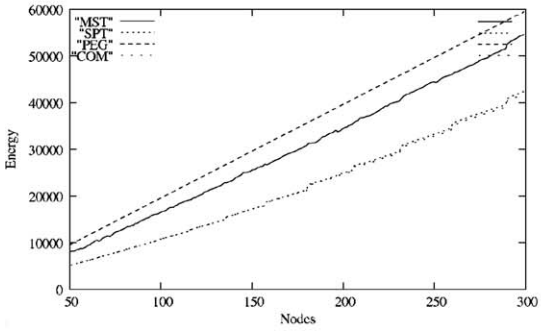
Fig. 6 (*continued*)



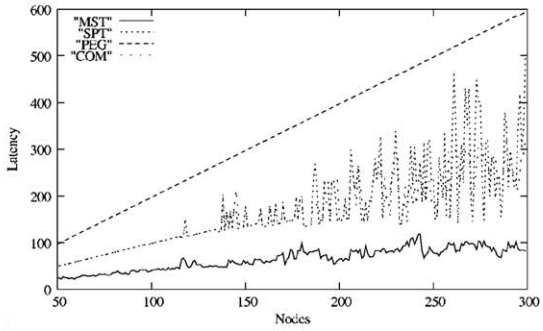
(a) $\alpha = 50\%, \beta = 1$ Node Density=10%



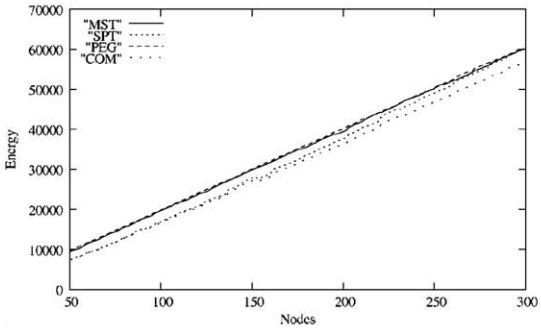
(b) $\alpha = 50\%, \beta = 1$ Node Density=10%



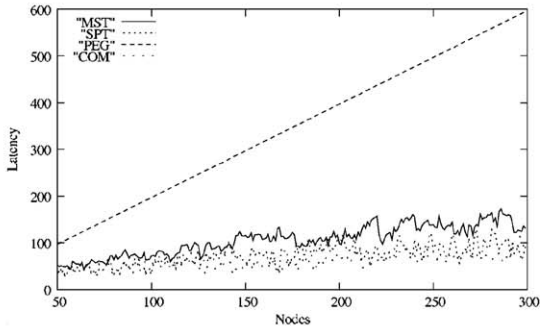
(c) $\alpha = 50\%, \beta = 1$ Node Density=80%



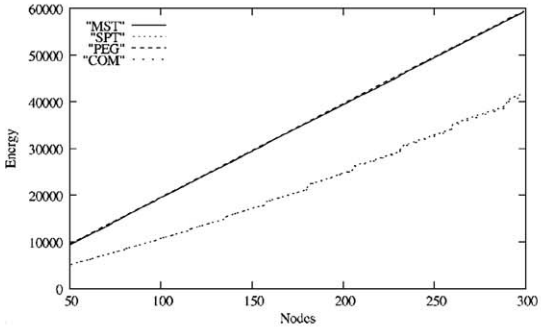
(d) $\alpha = 50\%, \beta = 1$ Node Density=80%



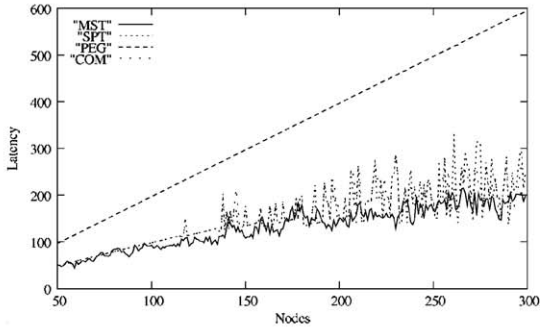
(e) $\alpha = 50\%, \beta = 2$ Node Density=10%



(f) $\alpha = 50\%, \beta = 2$ Node Density=10%

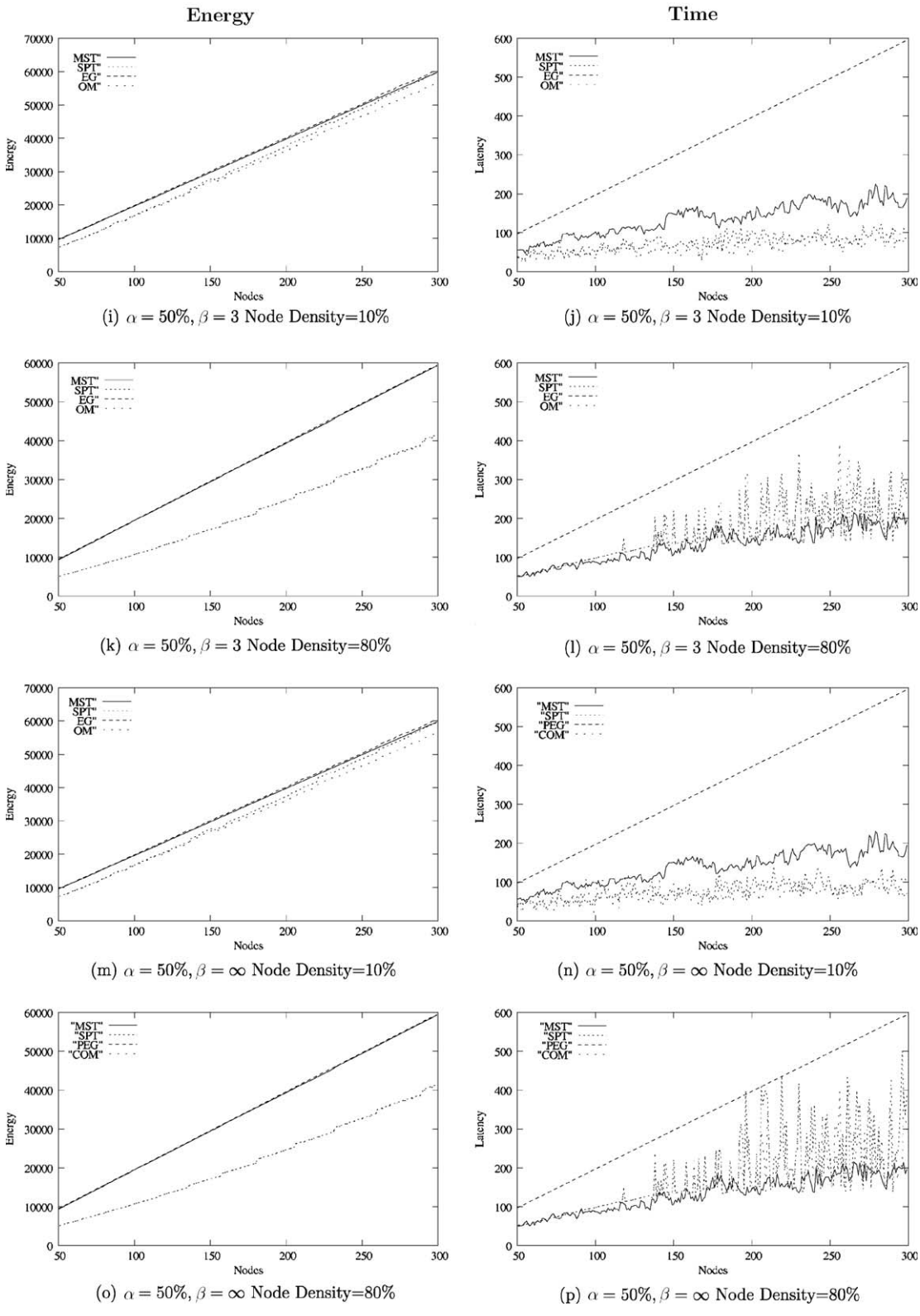


(g) $\alpha = 50\%, \beta = 2$ Node Density=80%



(h) $\alpha = 50\%, \beta = 2$ Node Density=80%

Fig. 7. Behavior of Pegasis, COM, MST-based and SPT-based algorithm for $\alpha = 50\%$.



(i) $\alpha = 50\%, \beta = 3$ Node Density=10%

(j) $\alpha = 50\%, \beta = 3$ Node Density=10%

(k) $\alpha = 50\%, \beta = 3$ Node Density=80%

(l) $\alpha = 50\%, \beta = 3$ Node Density=80%

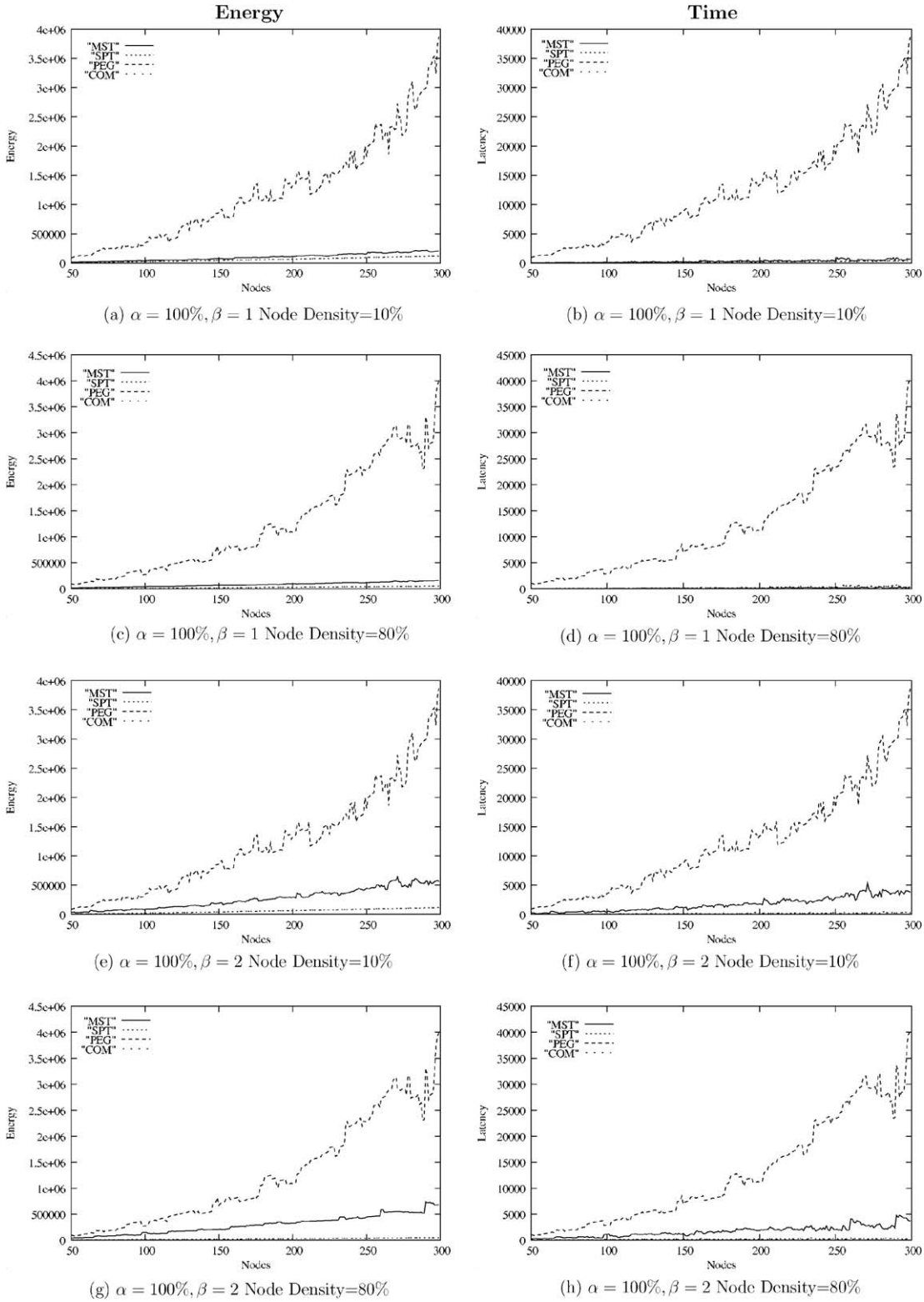
(m) $\alpha = 50\%, \beta = \infty$ Node Density=10%

(n) $\alpha = 50\%, \beta = \infty$ Node Density=10%

(o) $\alpha = 50\%, \beta = \infty$ Node Density=80%

(p) $\alpha = 50\%, \beta = \infty$ Node Density=80%

Fig. 7 (*continued*)



(a) $\alpha = 100\%, \beta = 1$ Node Density=10%

(b) $\alpha = 100\%, \beta = 1$ Node Density=10%

(c) $\alpha = 100\%, \beta = 1$ Node Density=80%

(d) $\alpha = 100\%, \beta = 1$ Node Density=80%

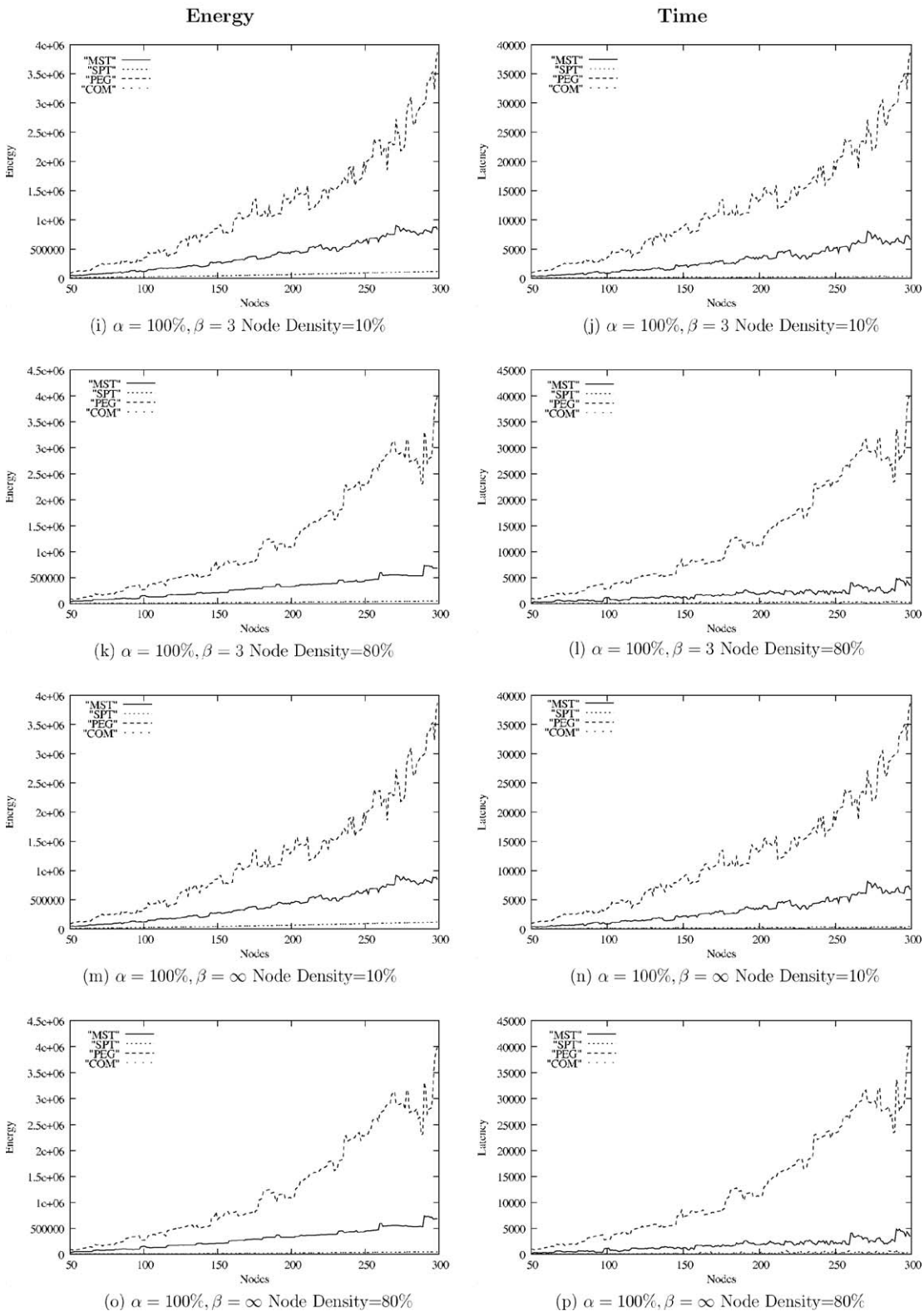
(e) $\alpha = 100\%, \beta = 2$ Node Density=10%

(f) $\alpha = 100\%, \beta = 2$ Node Density=10%

(g) $\alpha = 100\%, \beta = 2$ Node Density=80%

(h) $\alpha = 100\%, \beta = 2$ Node Density=80%

Fig. 8. Behavior of Pegasis, COM, MST-based and SPT-based algorithm for $\alpha = 100\%$.



(i) $\alpha = 100\%, \beta = 3$ Node Density=10%

(j) $\alpha = 100\%, \beta = 3$ Node Density=10%

(k) $\alpha = 100\%, \beta = 3$ Node Density=80%

(l) $\alpha = 100\%, \beta = 3$ Node Density=80%

(m) $\alpha = 100\%, \beta = \infty$ Node Density=10%

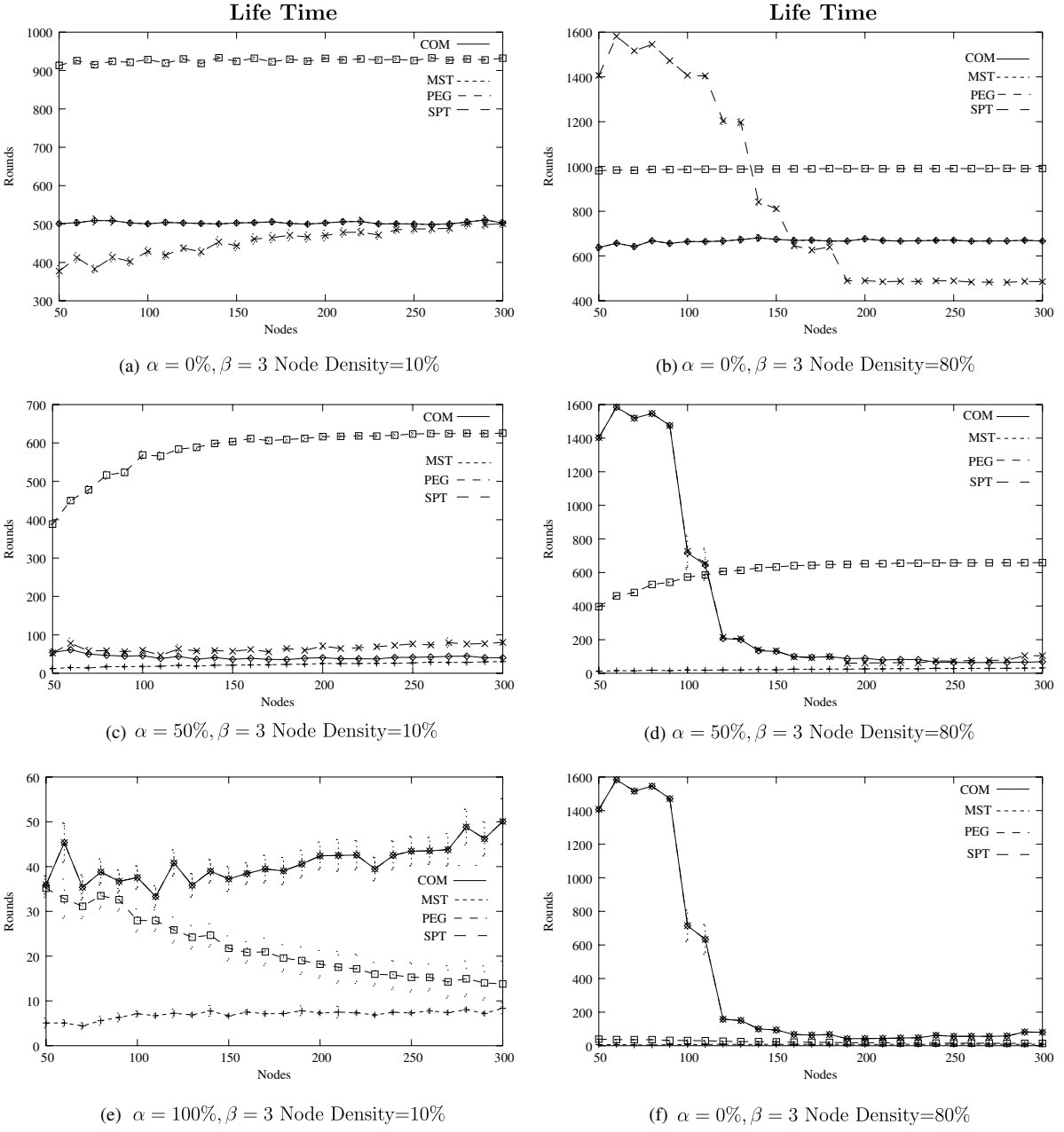
(n) $\alpha = 100\%, \beta = \infty$ Node Density=10%

(o) $\alpha = 100\%, \beta = \infty$ Node Density=80%

(p) $\alpha = 100\%, \beta = \infty$ Node Density=80%

Fig. 8 (*continued*)

[14] – especially for $\alpha = 0$ case. Pegasis is a chain-based protocol. It works by taking turns to transmit data to the base station. The motivation to perform such a task is to average out the energy required *per node* to transmit the data to base station. In every round a node collects the data from all other nodes along the chain and transmits it to base station. The designated node fuses the data it has collected and transmits them to the base station. Performance of Pegasis in terms of energy saving and network longevity is shown to be much better than some of the existing protocols [3]. To allow efficient use of this technique, a threshold is put on the distance of the designated node from base station. Only those nodes within this threshold can transmit data to base station. Working of Pegasis is shown with a simple figure (Fig. 4).



(a) $\alpha = 0\%, \beta = 3$ Node Density=10%

(b) $\alpha = 0\%, \beta = 3$ Node Density=80%

(c) $\alpha = 50\%, \beta = 3$ Node Density=10%

(d) $\alpha = 50\%, \beta = 3$ Node Density=80%

(e) $\alpha = 100\%, \beta = 3$ Node Density=10%

(f) $\alpha = 0\%, \beta = 3$ Node Density=80%

Fig. 9. Life Time of Pegasis, MST-based, SPT-based algorithm and COM algorithms.

6. Results and analysis

In the following section, the terms *low density* and *high density* refers to densities in the range of 10% and 80%, respectively. The terms *small network* and *large network* implies network size containing about 50 nodes and 300 nodes, respectively. The word *closeness* refers to the weight on the edges and not the distance between nodes.

From the simulations conducted using various input data (Section 5), the following observations were made (also see Figs. 5–9).

- For a given β, COM consumes least energy for any $0 < \alpha < 100\%$ when compared to all algorithms. However, $\beta = 2$ gives lower latency for COM than other β values. Therefore COM with $\beta = 2$ helps in achieving low energy with low latency.
- MST consumes least energy when $\alpha = 0$. But its energy consumption increases rapidly with the increase in α. For $\alpha > 25\%$, MST consumes more energy than SPT.
- For very low values of $\alpha < 25\%$, Pegasis' energy consumption is comparable to MST's energy consumption. However, it is a prohibitive algorithm for $\alpha > 25\%$.
- Chain structure of Pegasis results in linear dependence of latency on number of nodes in the network.
- For low density networks, SPT or COM are recommended algorithms for latency. And it performs better with $\beta = 2$ or 3.
- As density of the network increases, MST's latency outperforms other algorithms.
- Latency performance of COM is similar to SPT – good for low density networks and poor for high density networks.
- For $\alpha = 0$, the lifetime of the network which uses MST is same as that of COM (regardless of β). Similarly, for $\alpha = 100\%$, lifetime of network using SPT is same as that of COM.
- For $\alpha = 0$, and for low density networks, Pegasis has longest lifetime. But with the increase in density, SPT has longer life than all the algorithms for smaller networks (<150 nodes). For larger networks (>150 nodes), Pegasis has better lifetime performance. This clearly indicates that even though MST consumes least energy for $\alpha = 0$, its fixed tree structure will drain out energies of certain nodes quickly. Among all the three algorithms (MST and COM are similar for $\alpha = 0$) networks using MST has least life expectancy.
- As with the latency, SPT networks' life time depends greatly on density of the network. Denser the network, higher is the life time.

7. Conclusions

In this paper, we first presented two algorithms, one based on MST and the other on SPT, to solve the problem of minimum energy data aggregation enhanced convergecast (DAC) operations in wireless sensor networks. These algorithms were shown to be energy efficient for different extreme value of α (data growth factor). Based on these observations, a (generalized) combined SPT–MST algorithm was proposed which constructs DAC trees for all values of α. The data latency was reduced by re-structuring the energy-efficient tree using the β-constraint. It was also proved that for a given tree and for $\beta \geqslant |V|$ algorithms presented here consumes least energy for various scenarios (MST for $\alpha = 0$, SPT for $\alpha = 1$, and COM for $0 \leqslant \alpha \leqslant 1$). The performance of these algorithms were then compared with the popular Pegasis technique, which demonstrated superior efficiency of our approach. A summary of the results is presented suggesting the appropriate algorithms for various scenarios of network density and data compression factors.

Acknowledgements

The authors would like to thank the reviewers for their constructive comments which helped to improve the quality of this paper. This work was supported in part by the NSF grant # ANI-0196156.

References

[1] Imrich Chlamtac, Shay Kutten, Tree-based broadcasting in multi-hop radio networks, IEEE Transactions on Computers C-36 (10) (1987).
[2] Imrich Chlamtac, Orly Weinstein, The wave expansion approach to broadcasting in multi-hop radio networks, IEEE Transactions on Communications 39 (3) (1991).
[3] Wendi Rabiner Heinzelman, Anantha Chandrakasan, Hari Balakrishnan, Energy-efficient communication protocol for wireless micro sensor networks, in: Proceedings of the Hawaii International Conference on System Science, January 2000.
[4] Rex Min, Anantha Chandrakasan, Energy-efficient communication for ad-hoc wireless sensor networks, in: Conference

Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers, 2001, vol. 1, 4–7 November 2001.
[5] Sarma Upadhyayula, Valliappan Annamalai, Sandeep Gupta, A low-latency and energy-efficient algorithm for convergecast in wireless sensor networks, in: IEEE Global Communications Conference, 2003.
[6] Bhaskar Krishnamachari, Deborah Estrin, Stephen Wicker, Impact of data aggregation in wireless sensor networks, in: International Workshop on Distributed Event-Based Systems (DEBS'02) Vienna, Austria, July 2002.
[7] Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, John Heidemann, Impact of network density on data aggregation in wireless sensor networks, in: Proceedings of International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, July 2002.
[8] Steven S. Skeina, The Algorithm Design Manual, Springer-Verlag, New York, 1997, pp. 339–342.
[9] Marek Karpinski, Alexander Zelikovsky, New Approximation Algorithms for the Steiner Tree Problems, Journal of Combinatorial Optimization 1 (1997) 47–65.
[10] H. Takahashi, A. Matsuyama, An approximate solutions for the Steiner problem in graphs, Mathematica Japonica 6 (1980) 573–577.
[11] Hans J. Promel, Angelika Steger, RNC-approximation algorithms for the Steiner tree problems, in: Proceedings of 14th Annual Symposium on Theoretical Aspects of Computer Science, 1997, pp. 559–570.
[12] Alexander Zelikovsky, An 11/6-approximation algorithm for the network Steiner problem, Algorithmica 9 (1993) 463–470.
[13] Alexander Zelikovsky, Better approximation bounds for the network and Euclidean Steiner tree problems, Technical Report CS-96-06, University of Virginia.
[14] Stephanie Lindsey, Cauligi Raghavendra, Krishna M. Sivalingam, Data gathering algorithms in sensor networks using energy metrics, IEEE Transactions on Parallel and Distributed Systems 13 (9) (2002).
[15] Valliappan Annamalai, Sandeep K.S. Gupta, On tree-base convergecasting for wireless sensor networks, in: Proceedings of IEEE Wireless Communications and Networking Conference, 2003, Louisiana, USA.
[16] J.L. Gao, Analysis of energy consumption for ad hoc wireless networks using bit-meter-per-joule metric, IPN Progress Report 42-150, August, 2002.
[17] Jie Chen, David Seah, Wen Xu, Channel allocation for cellular networks using heuristic methods. Available from: <http://foulard.ee.cornell.edu/wxu/projects/cs574_project.pdf>.
[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, second ed., MIT Press, 2001.
[19] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, Erdal Cayirci, Wireless sensor networks: a survey, Computer Networks 3 (2002) 393–422.

Sarma Upadhyayula received his master's degree in Computer Science and Engineering from Arizona State University, Tempe. His main research interests include low power data aggregation algorithms in wireless sensor networks (WSN) and self-stabilizing protocols.

Sandeep Kumar S. Gupta is an Associate Professor in the Ira A. Fulton School of Engineering in the Department of Computer Science and Engineering and Arizona State University. He received the B.Tech degree in Computer Science and Engineering (CSE) from Institute of Technology, Banaras Hindu University, Varanasi, India, M.Tech. degree in CSE from Indian Institute of Technology, Kanpur, and M.S. and Ph.D. degree in Computer and Information Science from The Ohio State University, Columbus, OH. He has served at Duke University, Durham, NC as a post-doctoral researcher; at Ohio University, Athens, OH as a Visiting Assistant Professor; and at Colorado State University, Ft. Collins, CO as an Assistant Professor. His current research is focused on dependable and adaptive distributed systems with emphasis on wireless sensor networks, mobile computing, and biomedical applications. His research has been funded by the National Science Foundation (NSF), the Consortium for Embedded Systems (CES), Intel Corporation and MediServe Information Systems. He is co-author of the book "Fundamentals of Mobile and Pervasive Computing" published by McGraw Hill, 2004. He has co-guest edited special issues of IEEE Personal Communication Magazine (on Pervasive Computing, 2001), IEEE Transactions on Computers (on Mobile Computing and Databases, 2002), ACM/Baltzer Winet (Advances in Mobile Computing and Wireless Systems, 2003) and ACM/Baltzer Monet (on Pervasive Computing, 2001). He was program chair for Int'l workshop on Group Communication and a program co-chair for Int'l Workshop on Wireless Networks and Mobile Computing, Int'l Workshop on Pervasive Computing (PC'00), and Future Trends in Distributed Computing Systems (FTDCS'01), Int'l Workshop on Wireless Security and Privacy (WiSPr'2003). He is a member of the ACM and a senior member of the IEEE. He heads the IMPACT (Intelligent Mobile and Pervasive Applications and Computing Technologies) Lab at Arizona State University. For information about his recent research projects and publications visit http://impact.asu.edu.