Energy Efficient Global Clock Synchronization for Wireless Sensor Networks

Vinod Namboodiri, Suresh Ramamoorthy
Department of Electrical Engineering and Computer Science
Wichita State University, Kansas, U.S.A
{vinod.namboodiri, sxramamoorthy2}@ wichita.edu

Abstract

Clock synchronization is critical to many sensor networks for the success of the application as well as energy efficiency. Achieving a global time frame through localized averaging of clock values for multiple rounds till convergence is a promising approach to clock synchronization due to the decentralized nature of computation coupled with scalability. However, it is not clear what power levels for all nodes would make the synchronization process energy-efficient. Large power levels lead to faster convergence but consume a lot of energy per round of synchronization. On the other hand, smaller powers consume little energy per round, but convergence is very slow requiring a lot of rounds to achieve synchronization. In this paper we look at the problem of finding a power assignment that achieves global clock synchronization in the most energy-efficient manner possible. We look at the problem through two dimensions; rate of convergence and energy consumed per round of synchronization. A centralized algorithm is presented that uses the path congestion of the induced communication graph to estimate which power assignments have good convergence properties and find one that minimizes the total energy to achieve clock synchronization. Our evaluation demonstrates that the power assignment derived from this algorithm is very energy-efficient and is applicable for wireless communication environments with various distance-power gradients. Further, we present a simple distributed algorithm which nodes can execute locally to derive energy-efficient power levels for global clock synchronization, and is especially useful in large-scale deployments.

I. Introduction

One of the most important important problems in wireless sensor networks is clock synchronization. Clock synchronization is important for efficient duty cycling, avoiding redundant data collection, coordinated event tracking and many other applications. These sensors are required to keep themselves time synchronized in the most energy efficient manner possible so as to be able to utilize their lifetime for the designated applications as much as possible. For many sensor applications, like monitoring, data traffic is rare and the control messages required to achieve clock synchronization can be a sizable percentage of the total data traffic. This makes the problem of energy-efficient clock synchronization even more important.

Sensor networks generally do not require UTC based clock synchronization. All that is required is for the nodes to be synchronized among themselves. Due to offsets between initial values and drifts due to different frequencies of oscillation for clock crystals, clock readings at nodes tend to be different with time. The effect of drift on all clocks means that all nodes have to re-synchronize periodically. Recent studies have advocated for all nodes to distributively average their individual clock readings and use the average value as the global time frame [1]. Such a scheme is localized and requires only message passing between one hop neighbors. However, this does not take into account energy efficiency. The energy expended by each node for communication and the rate at which clock values of all nodes converge, decides how energy efficient the whole scheme is. The transmit powers or corresponding range of each node plays an important part in making this whole process energy-efficient.

It is not clear what the power assignment for all nodes should be to achieve this goal. A minimalist power assignment for all nodes such that they are just connected is one choice. Such power assignments are known to do well when transmission power is the only consideration [2]. However, for global clock synchronization, the rate of convergence is an important factor to be considered. Such minimal power assignments require a large number of operations among nodes before achieving a target degree of synchronization. On the other hand, a maximal power assignment for all nodes will achieve fast synchronization, but at the cost of energy expended to use large transmit powers. Both these solutions will work well only for certain limited scenarios and will not be suitable for any arbitrary network setting.

We propose a centralized algorithm *CPA* which produces a power assignment for all nodes of the network that achieves energy efficient global clock synchronization in sensor networks. We define our energy efficiency metric so as to minimize the total energy consumed for clock synchronization, while at the same time ensuring that the energy burden is shared by all the nodes. Our approach to tackle this problem is to look at it through two dimensions; namely how fast clock values converge for a given power assignment for all nodes and what is the energy cost of this power assignment. We demonstrate that the notion of path congestion of a graph can be used as a good estimator to how fast clock values converge, based on which a energy efficient power assignment can be done. Our results indicate that the energy required for clock synchronization can be reduced significantly by this approach. It is found that the value of distance-power gradient (also called path loss exponent) for wireless communication has a big impact on what is a energy efficient power assignment. Our path congestion based *CPA* algorithm is found to be effective regardless of the value of this gradient, making it applicable to different types of wireless environments. Further, we present a simple distributed algorithm *MaxMin* which nodes use to derive energy efficient power levels for themselves using just locally available information. We also derive sufficient conditions that can be checked locally by nodes to determine convergence of the synchronization algorithm. Our evaluation shows that the simplicity of our distributed *MaxMin* algorithm does not compromise a lot in terms of energy efficiency and is very practical for large scale deployments.

This paper is divided into the following sections. Section II surveys related work on clock synchronization especially those on wireless sensor networks. Section III presents background material on global clock synchronization which we will use as a base for our work. Section IV formulates the problem by defining the propagation model and energy efficiency metric followed by Section V that presents the concept of path congestion in a graph and how it can be used to estimate the rate of convergence of clock synchronization. Section VI analyzes the energy consumed per round of synchronization. Section VII presents our centralized algorithm for finding a energy efficient power assignment for all nodes. This is followed by our distributed algorithm in Section VIII which along with the centralized algorithm is evaluated in Section IX. We present a small-scale prototype implementation on sensor nodes to determine the feasibility of global clock synchronization based on message-passing. Concluding remarks are made in Section XI.

II. Related Work

Clock synchronization in general is a very old problem. A brief history of the problem can be found in [3]. NTP, which is the de facto standard for clock synchronization in the Internet, relies on a hierarchy of servers with clients synchronizing to these servers [4]. Sensor networks do not have the benefit of such a hierarchy. Also the tightness of synchronization desired by sensor networks is often greater compared to what NTP typically offers. Over the past few years, several time synchronization schemes directed specifically at sensor networks have been proposed [5]–[7]. Reference Broadcast Scheme (RBS) takes advantage of the broadcast nature of wireless networks by using a common sender to synchronize two receivers [5]. This enables both receivers to eliminate the clock uncertainty at the sender. The Timing-Sync protocol for Sensor Networks (TPSN) scheme improves on RBS by utilizing the fact that MAC level time stamping can be done in sensor nodes [6]. This removes all the uncertainty above the MAC layer on both ends of a link. The Flooding Time Synchronization Protocol (FTSP) further improves on these techniques by reducing jitter of the interrupt handling and encoding/decoding schemes with linear regression to compensate for drift [7]. All these techniques, while achieving tight levels of synchronization, deal with *local neighborhood synchronization only*. They need to be coordinated through the whole network to achieve a global time frame.

The concept of distributed gossip style protocols has received a lot of attention recently due to the obvious advantages it offers for large-scale networks [8], [9]. The localized nature of computation resulting in global consensus offers exciting possibilities to solve the time synchronization problem. A global time frame can be computed by relying on synchronous or asynchronous gossiping style protocols [1]. Nodes can use diffusion or averaging to synchronize clocks with their neighbors and rely on multiple such rounds of computation to achieve global synchronization. The above mentioned local synchronization methods can be integrated with this scheme to give tight pair-wise synchronization. Our work is based on the same synchronization protocols presented by [1], but we address the crucial issue of energy efficiency. For energy constrained wireless sensor networks, energy efficiency has to be integrated into every scheme and plays a major role in the trade-offs considered by any protocol. Our scheme achieves this by finding the appropriate transmit power for each node that will achieve this goal.

We use the theory of mixing times of Markov chains to find power assignments which are energy efficient. Similar work has been done on geometric random graphs, but restricted to uniform range for all nodes [10]. The work gave general bounds on mixing time with respect to the range, but did not cater to any application and aspects of its power consumption. We specifically address the application of clock synchronization and do not restrict ourselves to a homogeneous range for all nodes. Our aim is to integrate energy efficiency with the synchronization process by assigning each node a suitable transmit power.

III. Background

Any clock synchronization algorithm needs to have the quality of being localized in order to be feasible for sensor networks which are typically large and dense. Network wide broadcast schemes will drain a lot of energy and severely limit the lifetime of the network. Here we present the clock synchronization algorithm by [1], and set up the background for our work to make global clock synchronization energy efficient.

A. *Diffusion Algorithm for Clock Synchronization*

Algorithm 1 shows a slightly modified version of the diffusion algorithm presented by [1].

Algorithm 1 Diffusion Algorithm for Clock Synchronization

```
1: Repeat the following with a certain frequency
2: for each node $n_i$ do
3:     Request clock values of all its neighbors
4:     Receive clock values of all its neighbors
5: for each node $n_i$ do
6:     for each neighbor $n_j$ do
7:         $c_i = c_i - r_{ij}(c_i - c_j)$
```

The number of operations required to achieve clock synchronization can be logically separated into rounds which is defined as the time till every node completes a diffusion (explained below) operation with each of its neighbors. Within each round, each node requests the clock values of its neighbors and then receives them. It then computes the difference in time between its current value and each neighbor's value $c_i - c_j$. Algorithm 1 relies on the concept of diffusion to average the clock readings of all nodes to achieve a global time frame. A node with a higher clock value, loses some of its value to a neighbor with lower clock value. The value lost is proportional to the difference of their values. Similarly, a node with a lower clock value gains some fraction of the value of a neighbor with higher clock value. After multiple such rounds, the clock values of all nodes will converge to the average value. The proof of convergence of this rate based diffusion algorithm was given by [1] with the requirement that the induced graph be connected with symmetric neighbor relations. The key aspect of the algorithm is the rate diffusion matrix R. The entries r_{ij} of this matrix decide what fraction of the clock differences are diffused between node i and node j. All nodes adjust the diffused values to account for the elapsed time after their diffusion operations. The total time to achieve synchronization is considered small enough to ignore the effects of drift. [1] also present an asynchronous diffusion based algorithm similar to the one above without requiring nodes to participate in a set order. We stick to the synchronous version for sake of simplicity of analysis. The asynchronous version can be analyzed in a similar fashion by considering the probabilities of nodes to participate in a round.

B. *Construction of Rate Diffusion Matrix*

Let each node i have a clock value c_i^t at time t. Let $c^a = \frac{1}{n}\sum_{i=1}^{n} c_i^0$ be the average value of all nodes at time 0. Then the aim of all nodes is to achieve $c_i^k = c^a$ after k units of time, with every round of diffusion operations done within a period of unit time. The number of rounds needed to achieve convergence of all clock values to the vector $C^a = (c^a, c^a, ..., c^a)^T$ largely depends on the convergence of the rate diffusion matrix R.

To maintain the law of conservation, the value diffused between two nodes should be the same. That is, R is a symmetric matrix with $r_{ij} = r_{ji}$. Each node i gets to pick the rate at which it wants to diffuse with its neighbors j as long as each node j picks the same diffusion rate to i. Thus, we require $r_{ij} = r_{ji}$, and for each node i, $\sum_{j=1}^{n} r_{ij} = 1$ with $r_{ij} = r_{ji} = 0$ only if i and j are not neighbors of each other. Assign $r_{i,i} = 1 - \sum_{i \neq j} r_{ij}$. with the convergence of the rate-diffusion algorithm already proved, our main interest here is in the rate of convergence to the average value.

The communication links between n nodes can be modeled by a graph $G(V, E)$ with nodes representing vertices. There is an edge between any two vertices who are able to communicate to each other. A good way then for all nodes to construct the matrix R is to assign a value of $r_{ij, i \neq j} = 1/(d_{max} + 1)$ for all neighbors j of node i, where d_{max} is the maximum degree for any vertex in G. This ensures symmetry easily due to the fact that G is a graph with symmetric neighbor relations. If finding d_{max} proves difficult, all vertices can simply choose a value $1/n$ (or for that matter any value which all nodes can agree on that conforms to the requirements stated above) for all their adjacent edges. The downside to choosing $1/n$ compared to $1/(d_{max} + 1)$ is slower convergence to the average value. For further reading on choosing entries of R refer [11]. Since all nodes diffuse the same amount of their clock values to all neighbors in our work, this common diffused value will be referred to as r in our analysis later.

C. *Rate of Convergence*

The graph G above can be described in terms of a Markov Chain with stationary distribution $\pi(x) = c^a = 1/n, \forall x \in N$, represented by vector C, and with entries of R representing the transition matrix. Every time the synchronization algorithm is run, matrix R is applied to the current clock reading vector. That is, $C^t = R^t.C^0$ leading to $C^a = (c^a, c^a, ..., c^a)^T$ eventually after a finite number of rounds.

Scale all the clock values c_i such that they sum to 1. i.e. $\sum_{i=1}^{n} c_i = 1$. This gives us $c^a = 1/n$. We assume the induced graph from the network to be strongly connected, making matrix R irreducible. Also from above, it is symmetric and positive. Thus, the eigenvalues of R are real and by perron-frobenius theorem, $\lambda_1 = 1 > \lambda_2 \geq \lambda_3 \geq ...\lambda_n > -1$ [12]. Define $\lambda_{max} = (|\lambda_i| : 2 \leq i \leq n)$. It is then well known that the rate of convergence depends on the value of λ_{max}, and the relative error is bounded by $rerr \leq \frac{\lambda_{max}^k}{min_{i \in N} \pi_i}$ [13]. This gives, $rerr = max_{i,j} \frac{r_{ij}^k - 1/n}{1/n} \leq \frac{\lambda_{max}^k}{1/n}$ after k rounds. Thus we can bound the minimum required rounds to achieve a target relative error by

$$k \geq \frac{ln(rerr) - ln(n)}{ln(\lambda_{max})} \tag{1}$$

We use this lower bound on the number of rounds to converge, k, to compare among different topologies (formed for various transmit power levels at nodes of the network). We would ideally prefer a topology that converges fast using up as little energy per round as possible making the process of synchronization energy-efficient.

IV. Problem Formulation

To look for an energy efficient power assignment for clock synchronization, we need to define energy efficiency for our scenario and what is the metric we seek to optimize. But before that, we define the propagation model and transmit power capabilities of each node.

A. Propagation Model and Power Assignment

We use the log-distance model where the transmit power required at a sender to reach a receiver is equal to $c.d^{\alpha}$, where d is the distance between the two, c is the constant of proportionality and α is the path loss exponent or distance-power gradient [14]. α typically varies between 2 and 5 depending on the environment. It is assumed that the value of α is known to the algorithm which decides on an energy efficient power assignment.

Our aim is to find a power assignment which synchronizes the clocks in the most energy efficient manner possible. However, sensor nodes carry other application traffic, which may not have any correlation with the synchronization traffic. So we assume that each node can use its application specific power assignment as well as our power assignment for clock synchronization. Most sensor motes nowadays come with multiple adjustable power levels which can be controlled by the node itself.

B. Metric for Energy Efficiency

Energy efficiency is a broad term. The idea is to use the metric which is most pertinent to sensor networks. Energy is expended not just on clock synchronization but other activities specific to the application. So the focus should be on minimizing the energy per set of synchronization rounds. Sensor nodes are typically deployed as a group with the energy goal being maximum possible network lifetime. That is, collect data, record events e.t.c. as long as at least some subset of the nodes are alive. The network becomes disconnected when more than a certain number of nodes die. Thus, it is important to minimize the maximum energy expended by any single node. At the same time, when nodes do clock synchronization operations for many iterations until they converge, the total energy consumed also becomes important as the goal is to collectively achieve a global time frame for all. So an effective energy efficiency metric should take care of both these considerations.

Our approach strives to achieve this as follows. We consider various power assignments for all nodes and choose the assignment among these which minimizes the total energy consumption for clock synchronization. Each power assignment we consider is one that minimizes the maximum energy used over all nodes per round of synchronization. This combination of both maximum energy and total energy as metrics provides for topologies that minimize the total energy consumption for synchronization while ensuring that the energy burden is distributed among all nodes. This further allows us to evaluate power assignment algorithms independent of the underlying control and data traffic for the application which might decide the ultimate network lifetime for these nodes.

C. Problem Definition

We are looking to minimize the total energy expended by all nodes for clock synchronization. If it takes k rounds for the clocks of all nodes to synchronize, the total energy consumed by all nodes for achieving clock synchronization is

$$E_{metric} = k.E_{pr} \tag{2}$$

where E_{pr} is the total energy cost of all the nodes per round of clock synchronization. Thus our problem can be formally stated as

Given a set of nodes N and their location coordinates, find a power assignment $P = \{p_1, p_2, \cdots, p_n\}$, $P : p_i \rightarrow Z^+$ for all nodes $i \in N$ such that the total energy consumed for clock synchronization is minimized.
In other words, find P that minimizes E_{metric}. In the following sections we specifically look at how to find the parameters k and E_{pr} which would lead us to finding the power assignment that minimizes E_{metric}.

V. Estimation of Number of Rounds to Convergence

Convergence based on λ_{max} is very difficult to analyze directly. Lot of previous work on mixing times of markov chains have thus focused on using bounds on its value to prove convergence properties [13], [15], [16]. To estimate k, the number of rounds required to converge, we apply a commonly used technique of *canonical paths* to bound λ_{max} in Equation 1. The basic idea behind the technique is to try and associate *canonical paths* between every pair of nodes, in such a manner that no edge in G is used by too many paths. Intuitively, the existence of such a set of paths means that the topology has no severe bottlenecks that can impede the mixing of clock values. This notion is formalized below. For a more detailed treatment of this technique refer to [13].

A. Canonical Paths Technique

Let G be a weighted directed graph on the set of N nodes with directed edges between any pair of nodes which can communicate with each other (i.e. a directed graph with symmetric edge relations). Let the weight of each edge of G be $w(e) = c^a.r_{ij}$ whenever $r_{i,j} > 0$ with rate diffusion matrix R (with entries $r_{ij}, \forall i \in N, j \in N$) and desired clock values for all nodes at convergence $c_i^a = c^a = 1/n$, with $n = |N|$.

A set of canonical paths of G is a set Γ of simple paths $\{\gamma_{ij}\}$ in G, one between each ordered pair (i, j) of distinct vertices. If the paths are chosen in such a way that no edge is overloaded by paths, then the graph cannot have a constriction. Path congestion ρ which quantifies this degree of overloading is measured as

$$\rho \equiv \rho(\Gamma) = \max_e \frac{1}{w(e)} \sum_{\gamma_{ij} \ni e} c_i^a c_j^a \tag{3}$$

where the maximum is over oriented edges e in G. The value of path congestion ρ gives us a handle on λ_{max} based on the following theorem for markov chains [13].

Theorem 1. *For any reversible Markov chain, and any choice of canonical paths Γ, the second eigenvalue λ_{max} satisfies*

$$\lambda_{max} \leq 1 - \frac{1}{\rho l} \tag{4}$$

where $l \equiv l(\Gamma)$ is the length of the longest path in Γ.

The length of paths is taken into account because intuitively, the existence of short paths which do not overload any edge should imply rapid mixing of clock values. This also leads to better bounds on λ_{max} [16]. Using Equations 1 and 4, we can get a lower bound on the number of rounds to convergence.

$$k \geq \frac{ln(rerr) - ln(n)}{ln(1 - \frac{1}{\rho l})} \tag{5}$$

The significance of Equation 5 is that now we can estimate the number of rounds to convergence k by calculating ρ which is computationally much easier than finding λ_{max}.

Before we move on, we present a relation between the mincut size of a graph and path congestion ρ which will be used by our power assignment algorithm in Section VII.

Definition 1. *Let $G = (V, E, c)$ be an undirected graph with vertex set V, edge set E and non-negative real edge capacities $c : E \rightarrow R^+$. Let $n = |V|$ and $m = |E|$. A cut then is the partition of the vertices into two nonempty sets A and $\bar{A}$. The capacity of a cut $c(A, \bar{A})$ is defined by*

$$c(A, \bar{A}) = \sum_{u \in A, v \in \bar{A}, (u,v) \in E} c(u, v)$$

Definition 2. *A minimum cut or mincut of an n-vertex, m-edge capacitated, undirected graph is a partition of the vertices into two sets that minimizes the total capacity of edges with endpoints in different sets. The total capacity across this cut is called the mincut value. The cardinality of the cut in terms of number of edges is the mincut size.*

Theorem 2. *Given a weighted directed graph G with n vertices and m edges, with weights $w(e) = r.c^a$, and desired average values c^a, path congestion ρ satisfies*

$$\rho \geq \frac{1}{2\kappa.r} \tag{6}$$

where κ is the mincut size of the graph and r is the common diffusion rate between all neighbors i, j.

Proof. For any cut $c(S,\bar{S}) = c(S, N-S)$, the number of paths that cross the cut from S to $N-S$ is

$$|S|(|N-S|) \geq \frac{|S||N|}{2}$$

where $|S| \leq |N-S|$. Let κ be the minimum cut size among all such cuts. That is, all the paths from S to $N-S$ corresponding to this cut passes through κ edges. To minimize path congestion we would want all these edges to be as evenly loaded as possible. Thus, $\rho = \frac{|S||N-S|}{\kappa}.\frac{c^a c^a}{r\, c^a} \geq \frac{|S||N|c^a}{2\kappa r} = \frac{|S|n.1/n}{2\kappa r} = \frac{|S|}{2\kappa r} \geq \frac{1}{2\kappa r}$. $\square$

Having found a way to estimate the number of rounds to convergence, k through a lower bound in this section, we look at how to find the remaining component E_{pr} of Equation 2 in the following section.

VI. Quantifying Energy Consumption Per Round

Looking at Algorithm 1, it is clear that in every round of clock synchronization, each node transmits requests for neighbor clock values once, receives clock. If each node i has D_i neighbors, and assigned power p_i, the energy consumed by each node per round is proportional to $p_i.t_{tx} + p_i.D_i.t_{tx} + E_{rec}.D_i = p_i(1+D_i)t_{tx} + E_{rec}.D_i$, where E_{rec} is the energy cost of the radio to receive a packet. Thus the total energy consumed per round for synchronization operations can be quantified as

$$E_{pr} = \sum_{i}^{n=|N|} \{p_i(1+D_i)t_{tx} + E_{rec}.D_i\} \tag{7}$$

The node with maximum energy depletion per round among all the set of nodes N, also called the 'bottleneck' node, consumes

$$E_{max} = max_{\forall i \in N}\{p_i(1+D_i)t_{tx} + E_{rec}.D_i\} \tag{8}$$

VII. Congestion based Power Assignment (*CPA*) Algorithm

Having formulated the problem in the previous sections and analyzed the the key parameters we need to find E_{metric} in Equation 2, we are now ready to present our centralized heuristic algorithm that finds a power assignment to minimize E_{metric}. As explained earlier, it is difficult to estimate the value of k in Equation 2 and we use the notion of path congestion to guide us towards an energy efficient power assignment. Let k_{est} be the estimated value of number of rounds to convergence though Equation 5. Our strategy will be to find the value of

$$E_{est} = k_{est}.E_{pr} \tag{9}$$

for different power assignments, and choose the one which minimizes E_{est}. The claim is that this power assignment will be power efficient for E_{metric} also. This claim is evaluated in Section IX.

A. CPA Algorithm

The *CPA* algorithm shown, Algorithm 2, takes as input the set of nodes and their locations. The locations are primarily used for computing the distance between all nodes and thus the power required to communicate. Increasing the power of nodes possibly results in a different induced graph G every time. To start with, a topology is constructed using procedure *minPA* that minimizes E_{max} and has powers for all nodes to just keep the network connected. This algorithm constructs a minimum spanning tree (MST) on the induced complete graph of all nodes at their maximum power levels and the derived power assignment is shown to minimize the maximum power used by any node of the network [17]. This provides us with a connected graph that minimizes the maximum energy used over all nodes per round of synchronization and, acts as a starting point for the algorithm. The power assignment of this topology will further be used to compare against the power assignment of our algorithm in the evaluation section.

It is inefficient to search for energy-efficient power assignments by looking at all possible powers for all nodes. So in line with the idea of *Theorem* 2, we look at only those power assignments which increase the mincut size, κ. The strategy is to augment powers of nodes to increase κ and find the value of E_{est} for each value of κ. The power assignment that results in the minimum value of E_{est} among those considered is returned as the output of the algorithm.

Given the topology graph of a power assignment, two partitions are made of all nodes into sets $S1$ and $S2$ corresponding to the mincut, with $S1 \leq S2$. Then we increase the power of a pair of nodes on opposite sides of this cut by procedure *improveLocalCut*. With every such local augmentation, we attempt to increase the mincut size, eventually terminating when the mincut size is $n-1$. We are guaranteed to terminate because mincut size

Algorithm 2 *CPA* Algorithm

Input: Network $M(N, L)$,Path Loss Exponent α and Constant of Proportionality c.
Output: Power level p_u for each node $u \in N$ that induces a connected graph and minimizes the energy cost metric

```
1: $\kappa = 0$
2: $minPA()$
3: while $\kappa < n - 1$ do
4:   $constructR()$
5:   $\langle \kappa, S \rangle = findMincut()$
6:   Assign $S$ and $N - S$ to $S1$, $S2$ $s.t$ $|S1| \leq |S2|$
7:   if $\kappa$ has increased over last loop then
8:     $\langle \rho, l \rangle = findCongestion()$
9:     Calculate $k_{est}$ and $E_{pr}$
10:    Calculate $E_{est}$
11:  $improveLocalCut(S1, S2)$
12: return powers $p_u$ $\forall u \in N$ that minimizes $E_{est}$.
```

Procedure (*CPA*) *improveLocalCut*()

```
1: for all $u \in S1$ do
2:   for all $v \in S2, (u, v) \notin E$ do
3:     $tp_u = c.d_{uv}^{\alpha}$, $tp_v = c.d_{vu}^{\alpha}$
4:     Find neighbors $D_u$ of $u$ with power $tp_u$
5:     Find neighbors $D_v$ of $v$ with power $tp_v$
6:     Calculate $metric_{uv} = max\{tp_u(|D_u| + 1)t_{tx} + |D_u|E_{recv}, tp_v(|D_v| + 1)t_{tx} + |D_v|E_{recv}\}$
7: Find $(u, v)$ that minimizes $metric_{uv}$ storing their $tp$ values
8: $p_u = tp_u$, $p_v = tp_v$
```

is a non-decreasing function of increasing power with $n - 1$ being the maximum possible mincut size (also since mincut size increases after a finite number of power augmentations. Refer to Lemma 1 presented later for more details).

For each power assignment, the rate diffusion matrix R is constructed based on the induced graph. The next step is to calculate the value of path congestion ρ using Equation 3), E_{pr} (Equation 7), k_{est} (Equation 5) and finally E_{est} using Equation 9. If this value of E_{est} is the smallest till now, it is stored along with the corresponding power assignment P. At the end of the algorithm, the stored power assignment is the one we are looking for, one that minimizes E_{est}.

B. Local Augmentation Heuristic

The important step is to decide which node pair on either side of a cut should be chosen for the local augmentation in *improveLocalcut*. There could be many choices on either side. We could choose a pair, which are closest across cut, thus requiring minimal power increase to the nodes to improve the size of cut. However, we need to take care of the fact that the 'bottleneck' node may be one of those nodes and increasing its power further will only increase its burden. Thus we choose the node pair (i, j) which keeps E_{max} of Equation 8 minimum. This ensures that the energy consumed by the bottleneck node is kept to the minimum possible while improving the mincut size and satisfies our requirement (see Section IV) that the derived power assignment distribute the energy burden among all nodes.

C. Finding MinCut

Our procedure *findMincut* employs a standard technique using network flows to find the mincut size [18]. Given a source s and sink t in a graph $G(V, E)$ with unit capacities, the Ford-Fulkerson algorithm $maxflow(s, t)$ can be used to find the maximum flow between s and t. By the maxflow-mincut theorem, this flow corresponds to the mincut value in G which gives us the required mincut size. Since we do not have any specific s and t in our case, to find the mincut size we can run $maxflow(s, t)$ for all s and t. However, we can do better. By finding the node with minimum degree in the graph and using it as our s, and computing $maxflow(s, t)$ for all possible $t \neq s$, we can find the mincut size. To find the partition of nodes S, we just need to do a breadth first search from s in the residual graph. All nodes reachable from s, S, belong to one side of the mincut partition.

Procedure (*CPA*) $findMincut()$

```
1: let $s$ be the node with minimum degree $\in N$
2: for all $t \neq s$ do
3:     $cut_t = maxflow(s,t)$
4: $\kappa = min_{\forall t \neq s, t \in N} cut_t$
5: $S$ = All nodes reachable from $s$ in the
   final residual graph returning $\kappa$
6: return $\langle \kappa, S \rangle$
```

Procedure (*CPA*) $findCongestion()$

```
1: for binary search over $b = [0, n^2]$ do
2:     for all distinct node pairs $(x,y)$ forming a family of paths $\Gamma$ do
3:         Find a simple path between $x$ and $y$. If none found, exit loop
4:         If any edge used more than $b$ times, remove it from graph
5:         Store length of this path if maximum for $\Gamma$

6: Path Congestion, $\rho = b.\frac{c^a}{r}$
7: return $\langle \rho, l \rangle$, where $l$ is the maximum path length corresponding to the $b$ found by binary search
```

D. Finding Path Congestion

Path congestion can be found as follows. Specify a parameter b which is the maximum allowed overloading of an edge. Find a set of paths $\Gamma = \{\gamma_{xy}\}$, one between all distinct node pairs (x,y) in the topology graph such that no edge is used more than b times in the same direction. If such a set of paths can be constructed, the maximum overloading is $\leq b$. The least b such that such a set of paths can be found is the required maximum overloading of any edge in the topology. The path congestion can then be found by scaling this by $\frac{c^a.c^a}{c^a.r} = c^a/r$. The value of b can be found by doing a binary search over all possible values of b which ranges from 1 to n^2. The paths between a pair of nodes can be found using standard graph traversal techniques like Breadth First Search (BFS) or Depth First Search (DFS).

E. Running time analysis of CPA Algorithm

Lemma 1. *Checking all possible mincut sizes takes $O(n^2)$ iterations*

Definition 3. *A local cut is a cut whose size equals that of the mincut size of the graph.*

Proof. Each time we run procedure *improveLocalCut*, we increase at least one local cut. However, there could be many such local cuts in the graph. The mincut size of the whole graph will only increase after all these local cuts are increased. The maximum possible such local cuts we need to take care of is $O(n)$. This is because, with each local augmentation, we improve the size of cut between a pair of nodes and total such pairs which will increase the mincut size of the graph is no more than $n-1$ (easily follows from the simple example of a graph with n isolated nodes, thus having mincut size 0, will need add only $n-1$ edges to be connected and increase mincut size to 1). Also the maximum possible mincut size is $n-1$ in a graph with n nodes. Thus, all possible mincut sizes can be checked in $O(n^2)$ iterations. □

By Lemma 1 the main while loop executes for $O(n^2)$ iterations. The Ford-Fulkerson algorithm in $findMinCut$ takes $O(m\delta)$ time for any s, t pair, with maximum number of steps of flow augmentation denoted by δ and $m = |E|$. Since we have integer flows and unit capacities for all edges, the maximum flow in G can be no more than $n-1$. Thus, $\delta = O(n)$. Thus computing maximum flow between any s, t takes $O(nm)$ time. Running for all possible t takes another $O(n)$ time. Thus $findMincut$ takes $O(n^2m)$ to find κ. *improveLocalCut* takes $O(n^3)$. $findCongestion$ takes $O(log\, n)$ for the binary search, $O(n^2)$ for considering all possible distinct pairs of nodes with finding a path between each pair taking $O(n+m)$ time using BFS. $findCongestion$ is executed $O(n)$ times, only when the mincut size increases in the main while loop. Thus, the total running time of *CPA* algorithm is $O(n^2(n^2m+n^3)) + O(n.log\, n.n^2.(n+m))$ which is equal to $O(n^4m)$, since for large powers, the induced graphs are dense.

VIII. Distributed Algorithm

For distributed operation of the global clock synchronization algorithm based on diffusion, nodes need to be able to determine locally when the clocks are synchronized and also assign power levels to themselves such that the synchronization process is energy-efficient. In our prior presentation, a condition for nodes to individually determine whether convergence was reached was not given. In this section we present such sufficient local convergence conditions for nodes whereby they can determine when to stop synchronization operations themselves. Moreover, we present a simple algorithm *MaxMin* which nodes can execute distributively to come up with power levels for themselves that are energy-efficient for global clock synchronization. The *CPA* algorithm was suited for small scale sensor network deployments where a centralized entity could assign power levels to nodes both at the time of deployment and whenever a reconfiguration was required. As the scale of the network deployed increases, the emphasis on self-organization increases which calls for an algorithm which can be executed locally at all nodes without requiring a lot of computation.

A. Termination Condition for Synchronization Algorithm

The diffusion based clock synchronization algorithm will converge within a certain target error. We are interested in how nodes can locally determine that they do not need to execute any more rounds of the algorithm. One possibility of when nodes can stop diffusion operations is to look at the difference in clock values with its neighbors, i.e local one hop error, and the total network diameter χ. By ensuring that its local one hop error is less than $\frac{\epsilon}{\chi}$, where ϵ is the target error, a node can ensure that the global error is no more than ϵ. This is because the worst case total error would be the difference in clock values of the two nodes corresponding to the two ends of the network diameter. With this strategy, however, error accumulates per hop. This scenario is shown in Figure 1. We can do better based on the definition of local synchronization where error now accumulates only per two hops.

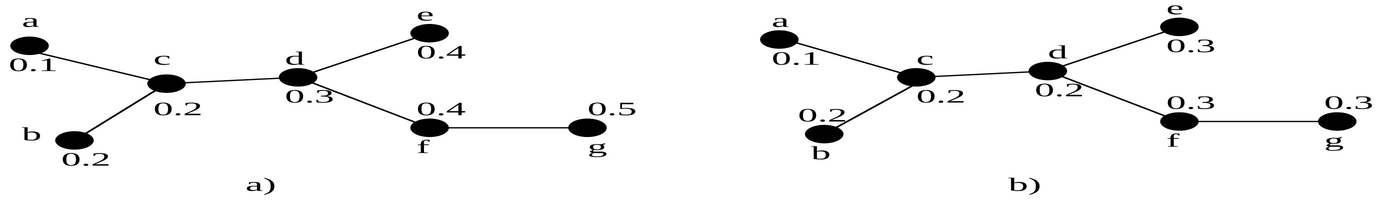


Fig. 1. Clock synchronization error at local and global scale. Node labels and their clock values are shown, with $\epsilon = 0.1$. Look at path from *a* to *g*, the network diameter. a) Neighbors of all nodes are within ϵ. Error accumulates every hop from node *a* b) All nodes are locally synchronized, i.e. all neighbor's of a node are within ϵ of themselves. Error accumulates only every two hops from *a*

Definition 4. *A node is locally synchronized if all the nodes (including itself) in its one hop neighborhood are within a target error ϵ from each other. i.e. if $N(u)$ is the one-hop neighborhood of node u, including u, it is locally synchronized if* $\forall_{v \in N(u), w \in N(u), v \neq w} |c_v - c_w| \leq \epsilon$.

Definition 5. *A network (with set of nodes N) is globally synchronized when the clock values of all nodes are within a target error ϵ. i.e.* $\forall_{u \in N, v \in N, u \neq v} |c_u - c_v| \leq \epsilon$

Lemma 2. *For a connected network, if all nodes are locally synchronized with error ϵ, the network is globally synchronized with error $\lceil \frac{\chi}{2} \rceil . \epsilon$, where χ is the network diameter.*

Proof. If a node u is locally synchronized, any neighbor of u, say v, has to have a time within ϵ of c_u. If this node v is locally synchronized, then any neighbor w of v has to also be within ϵ of c_u based on the definition of local synchronization. That is, nodes within two hops of u will still be within ϵ of c_u. The network diameter, defined as the largest hop distance between any two nodes in the network, is the worst case hop distance between any two nodes. We can accumulate an error of ϵ every two hops, so the total possible error is $\lceil \frac{\chi}{2} \rceil . \epsilon$. □

Figure 1b) shows the error accumulation when nodes are locally synchronized. Based on Lemma 2, nodes can determine when to stop diffusion operations by just checking if all are locally synchronized with error $\frac{\epsilon}{\chi/2} = \frac{2\epsilon}{\chi}$. This guarantees global synchronization with an error ϵ. Each node can easily check if it is locally synchronized because it acquires all its neighbor's values during diffusion operations. The value of χ could either be supplied at the time of deployment as a rough estimate, or inferred during packet level communications among the nodes.

B. Distributed Power Assignment Algorithm

The design goals of our distributed algorithm for an energy efficient power assignment are

1) The nodes should use only locally available information along with any overheard information from neighbors.
2) The communication overhead should be small.
3) The computation overhead should be small.

4) The derived power assignment should be energy efficient.

We assume that nodes do not have location information of any other node except themselves at the beginning of the algorithm. The pathloss exponent is considered to be uniform for all nodes of the network. Moreover, we drop the assumption of unlimited maximum power to a fixed limit.

The interesting point to note is that energy-efficient power assignments for global clock synchronization tends toward either minimal or maximal power levels for nodes (demonstrated later through the evaluation of the *CPA* algorithm) based mainly on the pathloss exponent. We utilize this fact in our *MaxMin* algorithm which distributively derives a power assignment for all nodes that is energy efficient. The main idea is that all the nodes select either a maximal power assignment or a minimal power assignment, with the decision based on a local estimate of the energy metric E_{pr} and the number of rounds for the synchronization algorithm to converge for a chosen power level. Note that it not advisable to just select among either of the maximal or minimal power levels based on the pathloss exponent because, for different number of nodes and size of deployment area, the better choice between a maximal and minimal power assignment might vary.

Algorithm 3 *MaxMin* Algorithm

Input: Set of nodes N, Path Loss Exponent α, Constant of Proportionality c.
Output: Power level p_u for each node $u \in N$ that induces a connected graph and minimizes the energy cost metric

```
1:  for each node $i \in N$ do
2:    $minpower(i) = DistMinPA()$
3:    Calculate $E^{min}_{pr(i)}$
4:    Run Synchronization Algorithm and note $k^{min}_i$
5:    Calculate $E^{min}_{metric(i)} = E^{min}_{pr(i)}.k^{min}_i$
6:    $maxpower(i) = DistMaxPA()$
7:    Calculate $E^{max}_{pr(i)}$
8:    Run Synchronization Algorithm and note $k^{max}_i$
9:    Calculate $E^{max}_{metric(i)} = E^{max}_{pr(i)}.k^{max}_i$
10:   if $E^{max}_{metric(i)} > E^{min}_{metric(i)}$ then
11:     $p(i) = minpower(i)$
12:   else
13:     $p(i) = maxpower(i)$
14:   if using $minpower(i)$ or heard any other node's similar broadcast then
15:     Announce minimal power assignment usage at maximum power level $P^{max}_i$
16:   if decided on $maxpower(i)$ and hear a minimal power usage announcement then
17:     $p(i) = minpower(i)$
```

Each node runs a localized algorithm *DistMinPA* that assigns power levels such that the network is connected with symmetric neighbor relations and uses the minimal power possible. This is the *minpower* level of the node. Each node calculates its energy consumption per round $E^{min}_{pr(i)}$. All nodes then execute the global synchronization algorithm and note down the number of rounds k^{min}_i they require to be synchronized for this power level. Each node obtains a local estimate of the energy metric $E^{min}_{metric(i)} = E^{min}_{pr(i)}.k^{min}_i$. Next, each node calculates the local estimate of energy metric for a maximal power (termed *maxpower* of the node) assignment found using *DistMaxPA* similarly, i.e. $E^{max}_{metric(i)} = E^{max}_{pr(i)}.k^{max}_i$. If the maximum power of nodes is unlimited or enough to cover while network, $k^{max}_i = 1$ and there is no need to execute the synchronization algorithm for the maximal power assignment. Based on these local estimates, each node decides on using the maximum power or minimum power assignment. If $E^{max}_{metric(i)} < E^{min}_{metric(i)}$, node i chooses $maxpower(i)$ as its power level else $minpower(i)$. At the end of this stage, any node having a *minpower* assignment broadcasts a packet at it's *maxpower* level to make nodes that had decided on *maxpower* levels to switch to their *minpower* levels. To prevent flooding of packets, once a node has sent an announcement, it can suppress future announcements.

The key aspect of the distributed algorithm as described above is the switching of all nodes to their *minpower* levels if there is no consensus to use *maxpower* levels. If some nodes decide on a *maxpower* level, it forces some other nodes who had decided on their *minpower* level to use up larger powers to reach these nodes to maintain symmetry in communication links. This leads to a much higher power consumption, specially in environments with high path loss exponents.

The algorithm *DistMinPA* used to construct a minimal power assignment in Algorithm 3 should output a connected topology with symmetric links. Moreover, it must be localized and should try to minimize the maximum power used by any node. Such a distributed topology control algorithm, Localized Minimum Spanning Tree (LMST), was proposed by [19]. The algorithm begins with nodes exchanging their neighbor lists with their neighbors, with a neighbor defined as any node which is within the maximum communication range of the node. Once the neighbor lists are exchanged, each node has knowledge of it's neighbors and their neighbors, i.e a local topology. A minimum spanning tree (MST) is constructed on the induced graph of this local topology by each node. The power level of a node is assigned as the power corresponding to the maximum adjacent edge of a node in its local MST (LMST). Asymmetric neighbor relations can then be made symmetric by either removing such links or augmenting powers to make those links symmetric (we use that latter option in our evaluations later). A power assignment derived from a LMST, like a MST, also tends to minimize the maximum power, sharing the energy burden as required by our energy metric [17].

During the maximal power assignment algorithm *DistMaxPA*, if the locations of all possible neighbors were known to each node, then the maximal power assignment for each node, for the case of unlimited maximum power, would be the power required to reach the node which was farthest from it. For the case of limited maximum power, the maximal power level would just be this power limit. In our case, the locations are not known, but by storing the neighbor information gathered during *DistMinPA*, a node can easily set the power level to reach the farthest possible neighbor.

IX. Evaluation

We use simulations to demonstrate the effectiveness of the centralized *CPA* algorithm as well as the distributed *MaxMin* algorithm to find energy efficient power assignments for clock synchronization. A network layout of size 20×20 with varying number of static nodes was used, with each data point shown the average of 20 runs. The power to receive a packet, P_{rec}, was taken as 35 mW which is the approximate value for the IEEE 802.15.4 radio [20]. The constant of proportionality, c, in our propagation model was taken as the value which made $P_{tx} = P_{rec}$ for a range of 10 with a pathloss exponent of 2. 802.15.4 radio data sheets show that these power values are roughly equal for the standard transmission range of the radio which we take as 10 here. Thus for a smaller range than 10, P_{tx} will be smaller than P_{rec}. The time to transmit a packet t_{tx} was taken equal to the time to send a 30 byte packet over a 802.15.4 link which has a data rate of 250 kbps. Time to receive a packet, t_{rec}, was set equal to the value of t_{tx}, giving $E_{rec} = P_{rec}.t_{rec}$. The clock values of all nodes were randomly assigned between 0 and 1 and, based on the topology induced by the derived power assignment, the synchronization algorithm was run till all nodes achieved a application specified target error rate. The common diffusion rate r for the rate diffusion matrix R was taken equal to $1/(d_{max} + 1)$. We start by describing the evaluation of our centralized algorithm *CPA* which is followed by the evaluation of our distributed algorithm *MaxMin* . The maximum power levels (and thus corresponding range) for nodes was considered unlimited for the *CPA* evaluation while a limit corresponding to a range of 10 was used for the *MaxMin* evaluation to simulate deployments where the scale is such that power limits are smaller than what is required to cover the whole network.

A. Effectiveness of CPA Algorithm

The goal of the *CPA* algorithm is to find a power assignment for all nodes such that the energy metric E_{metric} is minimized. To see how well the algorithm does, we compare it against other possible algorithms which are quite intuitive. This will give a fair idea of where the performance of our algorithm stands. Thus, next we describe these algorithms

1) Comparison Algorithms: *MinPA* is a power assignment that assigns the minimum possible powers to all nodes such that the maximum energy consumed by any single node is minimized, keeping the network connected. *MaxPA* is a power assignment that assigns the maximum required power for each node to reach all other nodes. *PA12* and *PA16* are power assignments which assign power levels to achieve communication ranges of 12 and 16 respectively given the 20×20 layout[1]. *PA12* and *PA16* serve the purpose of showing how effective constant power levels between those of *MaxPA* and *MinPA* can be.

We also provide a reference algorithm termed *RefPA* . *RefPA* also looks at the same power assignments (across increasing mincut sizes) as *CPA* , but instead of relying on an estimation of number of rounds, it executes the synchronization algorithm and finds out the exact number of rounds required for each assignment. This enables *RefPA* to find the optimal power assignment from among those considered. Note that *RefPA* does not necessarily return the optimum power assignment as it does not do an exhaustive search of all possible power assignments (which is computationally intractable). However, we did compare *RefPA* to an optimum algorithm for a 7 node network and found the energy metric values to be the same for both. It is practically infeasible to implement *RefPA*

[1]These values were chosen large enough to ensure connectivity of the network at all times. They are simpler than *MinPA* and *MaxPA* in that they used fixed power levels and are completely independent of the topology and node density.

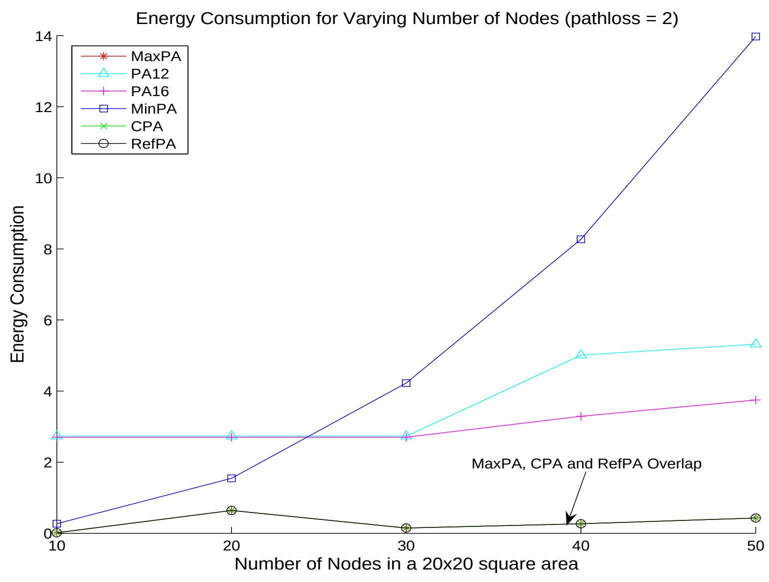


Fig. 2. Comparison of energy used by all schemes for pathloss = 2

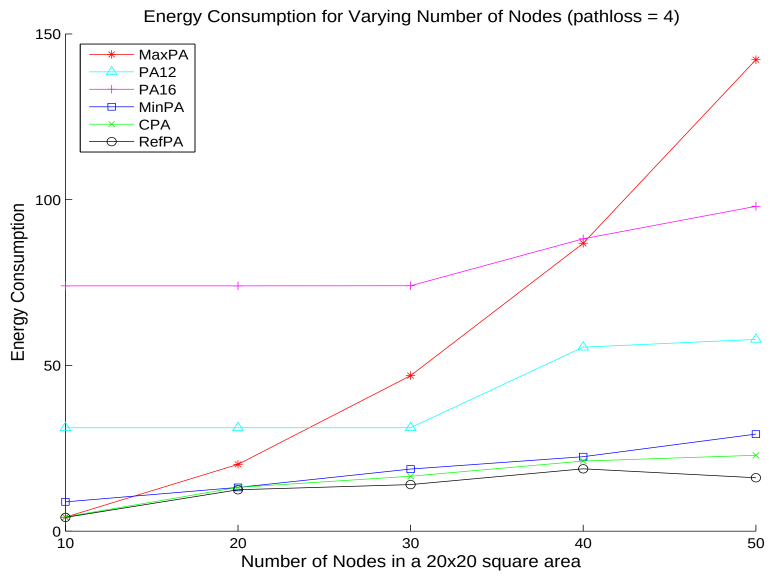


Fig. 3. Comparison of energy used by all schemes for pathloss = 4

algorithm as it uses knowledge of the clock values of all nodes at all times. *RefPA* is intended as a comparison tool only to quantify the magnitude of error of the *CPA* algorithm. An error in the estimate of the number of rounds to achieve convergence by *CPA* could result in both the *CPA* and *RefPA* power assignments to end up different, which is what we are interested in looking out for in the evaluation of this centralized algorithm.

2) Effect of Node Density with different pathloss exponents: Here the number of nodes are varied from 10 to 50 with a desired error of 10^{-04} between the clocks.

Figure 2 shows the energy usage for all schemes with a pathloss exponent, $\alpha = 2$. The energy consumption of *CPA* scheme is identical to the *MaxPA* and *RefPA* schemes. This shows that for a small α, maximal power assignments are most energy efficient. The rapid convergence seems to outweigh any gains by using smaller transmit powers. The *MinPA* scheme uses up a lot more energy due to the large number of rounds required to achieve convergence. The *PA12* and *PA16* are also inefficient, but better than *MinPA* due to their larger power levels resulting in faster convergence. *PA12* and *PA16* power levels do not always achieve a connected network when the number of nodes are less than 30, and hence we have assigned the same energy consumption as with 30 nodes to the case of 10 and 20 nodes (the same holds for $\alpha = 4$ below).

With $\alpha = 4$ (Figure 3), there is a big jump in the energy consumption of the *MaxPA* scheme. The large power

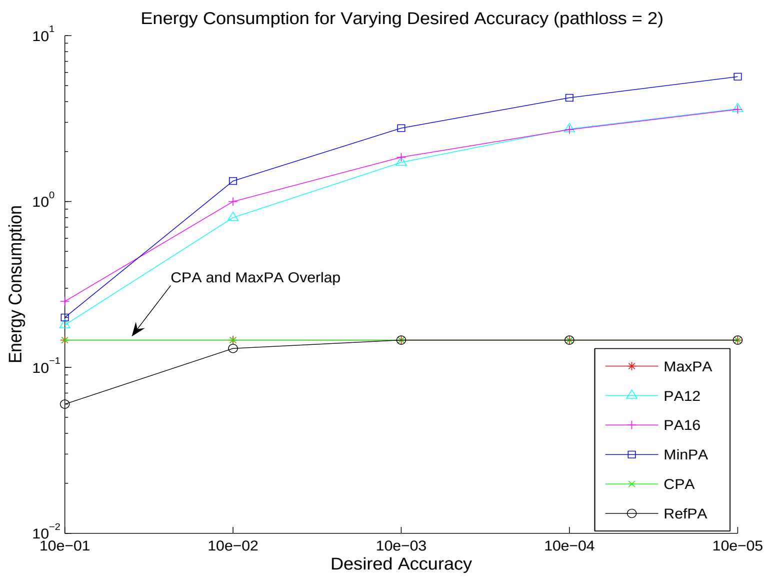


Fig. 4. Comparison of energy used by schemes for various desired accuracy (pathloss = 2)

required to cover all other nodes now pushes the energy consumption higher, in spite of only a single round of synchronization required. With higher emphasis on smaller powers now, the *MinPA* scheme performs much more efficiently than *MaxPA* except at very low densities. The *CPA* scheme does slightly better than *MinPA* as it seems to find power assignments which converge faster than those of the *MinPA* without using too much extra energy. *CPA* and *MinPA* schemes are comparable to *RefPA* at all densities because the emphasis now is on choosing lower powers that have good convergence properties. The higher pathloss exponent has made *PA12* and *PA16* energy inefficient compared to *MinPA* , in spite of their faster convergence.

The estimate of number of rounds to converge through the notion of path congestion in the *CPA* scheme is seen to be a good one because the power consumption of the topologies derived from the *CPA* and *RefPA* assignments are comparable for both pathloss exponents considered. That the *RefPA* algorithm does better than all other schemes validates its use as a reference algorithm. The *CPA* scheme finds energy efficient power assignments regardless of what the value of α is. In fact, the *CPA* scheme would also work for environments with mixed pathloss exponents for different nodes as long as these exponents are known to the algorithm.

3) Effect of Varying Desired Accuracy: Here, we look at how the schemes compared for varying error thresholds desired. 30 nodes were randomly placed in a 20×20 square area.

For $\alpha = 2$, the plot is shown in Figure 4. The *MaxPA* energy consumption remains the same regardless of desired accuracy because it synchronizes the clocks in just one round. The *RefPA* scheme uses lower powers for lower accuracies and finds the maximum power optimal for higher accuracies. The *CPA* scheme finds maximal powers more energy-efficient and has the same assignments as *MaxPA* . The *PA12* and *PA16* assignments are again found to be generally inefficient.

A similar observation can be made from Figure 5 for $\alpha = 4$. The main difference is the inefficiency of *MaxPA* at higher pathloss exponents as observed earlier. The plots of *MinPA* , *CPA* and *RefPA* are all comparable.

B. Effectiveness of MaxMin Algorithm

Based on the evaluation above, it is clear that depending on the value of α, either the maximal or minimal power assignment is energy efficient for global clock synchronization. Here we evaluate how energy-efficient the power assignments derived from *MaxMin* are by comparing them with those of the better of minimal and maximal power assignments which were generated as explained in Section VIII. Our goal here is to see whether *MaxMin* is able to select the more energy efficient of the two, frequently enough. We do not compare against the *CPA* algorithm due to its large running time which requires considerable amount of time to generate results for greater than 50 nodes. Our previous experiments show that the better of maximal and minimal power assignments compares very favorably with the *CPA* algorithm, thus validating its use for gauging the effectiveness of the *MaxMin* algorithm. We vary the pathloss exponent as well as number of nodes for this experiment.

Figure 6 shows that the effectiveness of *MaxMin* also depends on the value of pathloss exponent. For low values of α, the maximal power assignment is very energy efficient, but the *MaxMin* algorithm does not always choose it due to the fact that there are some nodes who still find a minimal assignment more energy efficient for themselves.

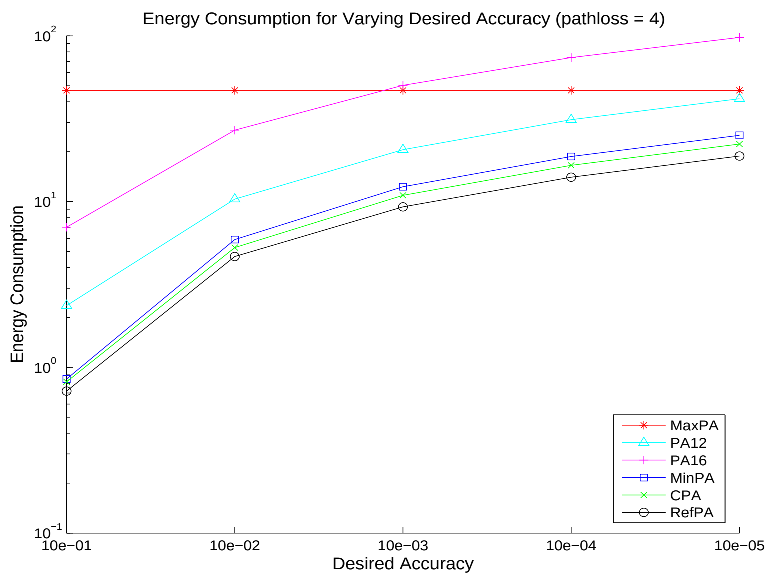


Fig. 5. Comparison of energy used by schemes for various desired accuracy (pathloss = 4)

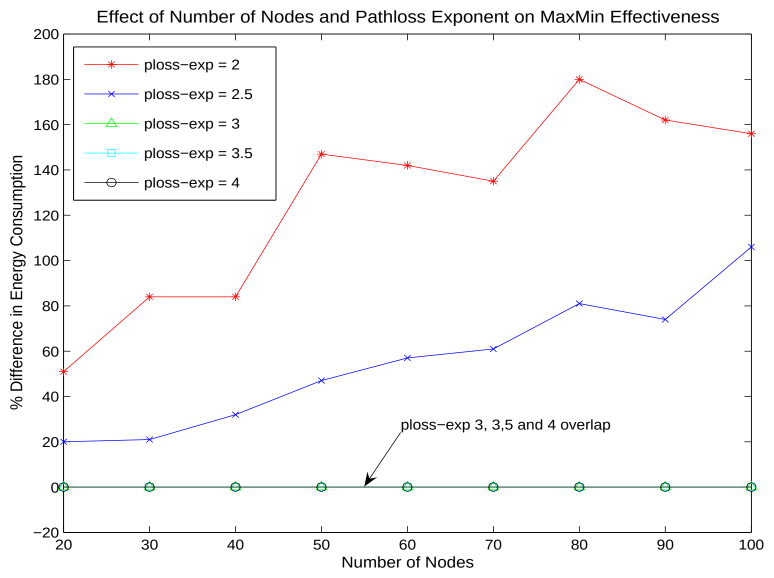


Fig. 6. Effectiveness of *MaxMin* algorithm as compared to the better of maximal and minimal power assignments for various pathloss exponent values for different number of nodes

This forces all nodes to use their minimal power assignment which results in more energy being used by *MaxMin* for lower values of α, especially with larger number of nodes. For larger values of α, above 2.5, *MaxMin* is able to correctly choose the energy efficient minimal power assignment.

It could be argued that *MaxMin* is not energy efficient at low pathloss exponents by being conservative in selecting a maximal power assignment. However, we found that it was always better to bias towards the minimal power assignment as it does not result in large values of energy consumption even for low values of α, while maximal power assignments are very inefficient for large values of α. *MaxMin* does much better than a strategy that chooses a minimal power assignment always by selecting a maximal power assignment when all nodes agree on using it. For comparison purposes Figure 7 shows the corresponding plot for a strategy that always uses a minimal power assignment. It can be clearly seen that *MaxMin* performs much better for the pathloss exponents 2 and 2.5. A further extension to *MaxMin* forcing all nodes to use the maximal power assignment for very low pathloss exponent (around the value 2) can be used if the scale of the network is much larger than those considered in our experiments. This will prevent further energy inefficiency for such low path loss exponents.

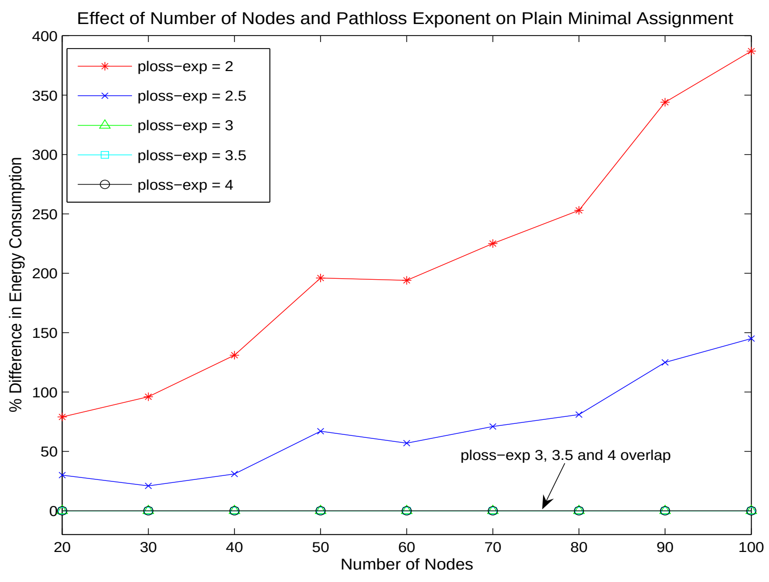


Fig. 7. Effectiveness of always using the minimal power assignment as compared to the better of maximal and minimal power assignments for various pathloss exponent values for different number of nodes

X. Feasibility of Implementing a Global Clock Synchronization Algorithm

We felt it was important to test the feasibility of implementing any algorithm based on periodic, distributed message passing. To the best of our knowledge, this has not been done by any previous work[2]. We mainly test for the practicality of being able to synchronize clocks based on clock value broadcasts by nodes distributedly. We do not test our *CPA* and *MaxMin* algorithms due to the small-scale testbed we had at our disposal; this study was mainly done to understand the challenges that would be faced when global clock synchronization algorithms (including our *CPA* and *MaxMin*) are implemented.

A. Target Platform and Setup

We have considered Berkeley Micaz motes as the WSN platform. These motes have a MPR2400CA Processor/Radio board, which is a 2.4GHz mote module capable of setting up low-power wireless sensor networks. The micaz motes have 128K bytes of programmable flash memory, 512K bytes of serial flash, 2400MHz wireless RF transceiver, which is capable of 250 Kbps, transmit data rate-with 75m to 100m outdoor range. The Micaz motes work on TinyOS operating system, which is event driven and modular OS, specially designed for wireless sensor devices.

Our experimentation setup consisted of 5 Micaz motes, 4 of which are configured with required application and one with Base station application. The base station node is connected to a PC. The base station node promiscuously listens to the packets exchanged between the nodes and pushes the raw data on to the PC. The PC is configured with a front end DotNET application that is capable of displaying the clock differences and local clock values of all the nodes. By this we can infer after how many cycles the synchronization is achieved and the accuracy of synchronization. The front end DotNET application basically acts as a tool to dissect the raw data into user understandable information.

B. Time Synchronization Methodology

TinyOS provides with three different Timer interfaces based on precision. The three different timers based on the increasing order of their precision are as follows, 1) TMilli (1024 ticks per second 2) T32khz (32768 ticks per second) 3) TMicro (1048576 ticks per second). The accuracy of the clock is 0.03 milli seconds. For all our experiments, we made use of the TMilli timer . Apart from the above mentioned timer interfaces, TinyOS also provided with multiple predefined clocks that make use of the above accuracy level . For our experiment we made use of the local time interface with T32khz accuracy. The local time interface provides with a 32 bit counter with out any overflow utilities. This counter does provide any specific trigger on overflow. One of the drawback on using these kind of pre-defined counters provided by TinyOS is that the local clock value cannot be adjusted or changed

[2]Note, as mentioned in Section II, most currently proposed time synchronization protocols for wireless sensor networks are based on centralized architectures, with the work in [1] being the sole exception that uses a diffusion based, distributed algorithm.

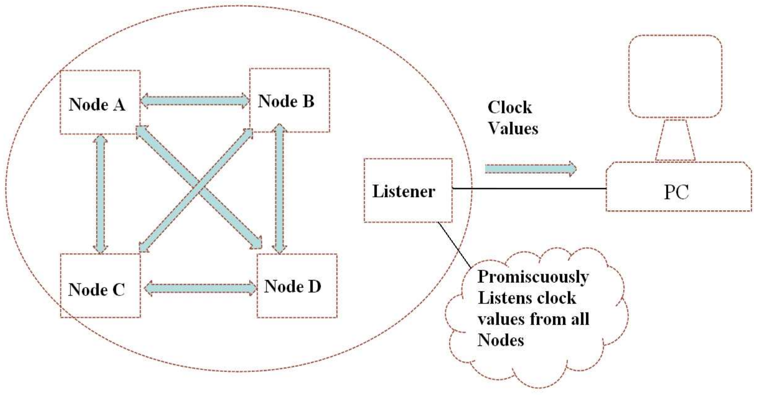
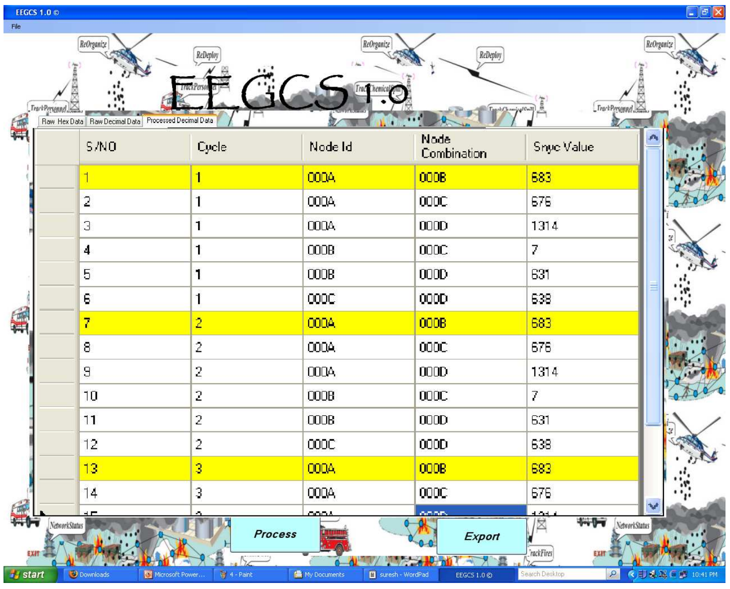


Fig. 8. Experimental setup to check feasibility of global clock synchronization

by the application code. The counter starts to run as soon as the application code gets booted up and keeps on running until it is switched off. For our application, we needed this clock value to be adjusted on the fly so we came up with a idea of introducing a variable named $OFFSET$. It's use is made based on the following equation: $GLOBALCLOCK = LOCALCLOCK(+/-)OFFSET$, where $LOCALCLOCK$ is the counter value from local time interface, $GLOBALCLOCK$ is clock value to be transmitted, and $OFFSET$ is the adjustment variable. The $OFFSET$ is a 32 bit unsigned integer, so by adding or subtracting the $OFFSET$ variable to/from the local clock we were able to modify the global clock. As the offset values are changed, apparently global clock value is changed.

At regular intervals, all nodes will broadcast their global clock value, on hearing this message all the nodes will compare their global clock value with the received global clock value. If the received global clock value is less than that of a node's global clock, it will adjust its offset so that the received global clock equal the nodes global clock value. To find out whether synchronization is achieved, we needed to compare the global clock value. However, these broadcasted global clock values cannot be used for comparison as they are not triggered at the same point of time, or in other words, the broadcasted clock values represent different instances of time.

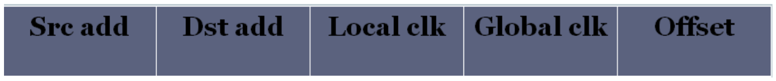
Thus, in order to check for synchronization, each node will transmit a Trigger packet at fixed time interval. On receiving this trigger packet all the node will capture their clock values and transmit them. The listener module will capture these trigger packets and forward them to the DotNET application, where the accuracy of Synchronization is calculated. Figure 9, gives a snap shot of the front end DOTNet application



S/NO	Cycle	Node Id	Node Combination	Snyc Value
1	1	000A	000B	683
2	1	000A	000C	676
3	1	000A	000D	1314
4	1	000B	000C	7
5	1	000B	000D	631
6	1	000C	000D	638
7	2	000A	000B	683
8	2	000A	000C	676
9	2	000A	000D	1314
10	2	000B	000C	7
11	2	000B	000D	631
12	2	000C	000D	638
13	3	000A	000B	683
14	3	000A	000C	676

Fig. 9. A snapshot of our front end application (DOTNet)

The payload of the the Synchronization message is 12 bytes. It is split up as follows: Global Clock (4 bytes), Local Clock (4 bytes), and Offset (4 bytes). It is obvious that the local clock and offset of the nodes need not be broadcasted as they are not used in any calculation on the other nodes; we broadcasted these extra packets only for the purpose of troubleshooting.



Src add	Dst add	Local clk	Global clk	Offset

Fig. 10. Message Format

C. *Results*

We checked clock synchronization accuracy both 15ms after a trigger, as well as 500 ms after a trigger. In the first case, the maximum error between the clocks of any pair of nodes was 0.03 ms. In the second case, the maximum error was 6ms. Thus, to achieve millisecond accuracy, the synchronization interval should be ideally no more than 100ms. The larger the network, and more multihop in nature, smaller should be the synchronization interval.

However, this result can be further improved upon by using linear regression to dynamically calculate clock skew as done in [7]. Further, in our experiments, we used the lowest resolution timer in TinyOS, TMilli (1024 ticks per second). Much better results can be obtained by using the TMicro timer (1048576 ticks per second). Thus, we feel that by using these suggested improvements, accurate global clock synchronization can be achieved with reasonably high synchronization intervals (100ms or more). And by further using our proposed power-assignment algorithms, clock synchronizatioachievede achived in an energy-efficient way as well.

XI. Conclusions

The problem of what is a good power assignment to achieve global clock synchronization in wireless sensor networks was studied. A centralized algorithm, *CPA* , based on path congestion of a graph was presented to estimate the number of rounds required to converge for a given power assignment and hence find one that is most energy-efficient. The *CPA* algorithm was found to consume less energy for synchronization than other reference algorithms considered. A simple distributed algorithm, *MaxMin* , which is more practical for large scale deployments was then presented that nodes could execute locally to derive energy efficient power levels for clock synchronization through distributed averaging. Through evaluations we showed the success of these algorithms to achieve accurate clock synchronization in an energy-efficient manner. Finally, we demonstrated the feasibility of achieving global clock synchronization in commodity wireless sensor nodes through a small-scale prototype implementation. This feasibility bolsters our argument that energy-efficient global clock synchronization is important in wireless sensor networks and is achievable.

This work throws open many new issues researchers can look at in the area of global clock synchronization in wireless sensor networks. One direction of future work is to implement our global clock synchronization algorithms practically in large sensor network deployments and demonstrate possible energy savings. Another direction of future work is to look at how our algorithms can be integrated with sleep schedules of wireless sensor nodes. Currently, our work only tries to be energy efficient with the clock synchronization process; much more can be done to tie up the synchronization algorithms with other energy saving approaches used in WSNs.

References

[1] Li, Q. and Rus, D. (2004) 'Global Clock Synchronization in Sensor Networks', *Proceedings of IEEE INFOCOM*, pages 564-574, Hong Kong, China.

[2] Santi, P. (2005) 'Topology control in wireless ad hoc and sensor networks', *Wiley*.

[3] Allan, D. W., Ashby, N. and Hodge. C. C. (1997) 'The Science of Timekeeping', *HP application note 1289*.

[4] Mills, D.L. 'Internet time synchronization: the Network Time Protocol' *IEEE Trans. Communications COM-39*, 10, October, 1482-1493.

[5] Elson, J., Girod, L. and Estrin, D. (2002) 'Fine-grained network time synchronization using reference broadcasts', In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 147-163, Boston, USA.

[6] Ganeriwal, S., Kumar, R. and Srivasatava, M.B. (2003) 'Timing-sync protocol for sensor networks', In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 138 - 149, Los Angeles, USA.

[7] Maroti, M., Kusy, B., Simon, G. and Ledeczi, A. 'The flooding time synchronization protocol', In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 39-49, Baltimore, USA.

[8] Boyd, S., Ghosh, A., Prabhakar B. and Shah, D. (2005a), 'Gossip Algorithms: Design, analysis and applications', In *Proceedings of IEEE INFOCOM*, pp. 1653- 1664, Miami, USA.

[9] Kempe, D., Dobra, A. and Gehrke, J. (2003) 'Gossip-based computation of aggregate information', In *Proceedings of IEEE Conference on Foundations of Computer Science (FOCS)*, pp. 482-491, Cambridge, USA.

[10] Boyd, S., Ghosh, A., Prabhakar, B. and Shah, D. (2005b) 'Mixing times for random walks on geometric random graphs', *Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, Vancouver, BC.

[11] Boyd, S., Diaconis, P. and Xiao, L. (2004) 'Fastest mixing Markov chain on a graph', In *SIAM Review*, 46 , no. 4, 667-689.

[12] Horn, R. A., Johnson, C. R. (1990) 'Matrix Analysis', *Cambridge University Press*.

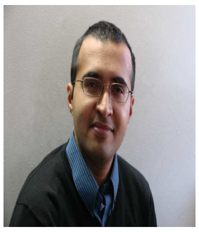
[13] Sinclair, A. 'Improved bounds for mixing rates of markov chains and multicommodity flow', *Combinatorics*, Probability and Computing, vol. 1, pp. 351-370.

[14] Rappaport, T. S. (1996) 'Principles and practice', *Prentice-Hall*.

[15] Avin, C. and Ercal, G. (2005) 'Bounds on the Mixing Time and Partial Cover of Ad-Hoc and Sensor Networks' In *Second European Workshop on Wireless Sensor Networks (EWSN)*, pp. 1-12, Istanbul, Turkey.

[16] Diaconis, P. and Stroock, D. 'Geometric bounds for eigenvalues of Markov chains', In *Annals of Applied Probability*, vol. 1, pp. 375-381.

[17] Ramanathan, R. and Rosales-Hain, R. (2000) 'Topology Control of Multihop Wireless Networks using Transmit Power Adjustment', *Proceedings of IEEE INFOCOM 2000*, vol. 2, pages 404-413.
[18] Sedgewick, R. (2001) 'Algorithms in C++: Part V. Graph Algorithms', *Addison-Wesley*.
[19] Li, N., Hou, J. and Sha, L (2003) 'Design and analysis of an mst-based topology control algorithm', *Proceedings of IEEE INFOCOM*, San Francisco, CA, pp. 275-286.
[20] Bougard, B., Catthoor, F., Daly, D. C., Chandrakasan, A., and Dehaene, W. (2005) 'Energy Efficiency of the IEEE 802.15.4 Standard in Dense Wireless Microsensor Networks: Modeling and Improvement Perspectives', In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, vol. 1, pp. 196-201, Washington, DC.



Vinod Namboodiri Vinod Namboodiri is currently an Assistant Professor at the Department of Electrical Engineering and Computer Science at Wichita State University, USA. He graduated with a Ph.D. in Electrical and Computer engineering from the University of Massachusetts Amherst in 2008. His research interests are in designing medium access control and routing protocols, and developing energy-efficient protocols and algorithms for different wireless technologies like Wireless LANs, RFID Systems, Wireless Sensor Networks, and Wireless Mesh Networks. His current research interests include designing communication techniques for smart electric grids, and integrating sustainability in mobile computing. He has served on the technical program committees of IEEE GLOBECOM, IEEE ICC, IEEE IPCCC, and IEEE GreenCOM, and is an active reviewer for numerous journals and conferences in the wireless networking and mobile computing areas.

Suresh Ramamoorthy Suresh Ramamoorthy is currently a Graduate Research Assistant at the Department of Electrical Engineering and Computer Science at Wichita State University, USA. He completed his B.S from Bharathiar University in 2003. His research interests are in understanding the energy consumption of wireless network interfaces of Wireless LANs, Wireless Sensor Networks, and RFID Systems. He has held a prior industrial appointment at Kone Elevators, India where he designed and developed micro-controller techniques for products relying on automation.