SYNC-NET: Distributed Time Synchronization in Clustered Sensor Networks

Ossama Younis
Department of Electrical and Computer Engineering, University of Arizona
E-mail: younis@ece.arizona.edu
Sonia Fahmy
Department of Computer Science, Purdue University
E-mail: fahmy@cs.purdue.edu

***Abstract*— Time synchronization is essential for several ad-hoc network protocols and applications, such as TDMA scheduling and data aggregation. In this paper, we propose a time synchronization framework for clustered, multi-hop sensor networks. We assume that relative node synchronization is sufficient, i.e., consensus on one time value is not required. Our goal is to divide the network into connected synchronization regions (nodes within 2-hops) and perform inter-regional synchronization in $O(LLSync) \times N_{iter}$ time, where $O(LLSync)$ denotes the complexity of the underlying low-level synchronization technique (used for single hop synchronization), and N_{iter} denotes the number of iterations where the low-level synchronization protocol is invoked. Thus, our main objective is *rapid convergence*. We propose novel fully-distributed protocols, SYNC-IN and SYNC-NET, for regional and network synchronization, respectively, and prove that N_{iter} is $O(1)$ for all protocols. Our framework does not require any special node capabilities (e.g., being GPS-enabled), or the presence of reference nodes in the network. Our framework is also independent of the particular clustering, inter-cluster routing, and low-level synchronization protocols. We formulate a density model for analyzing inter-regional synchronization, and evaluate our protocols via extensive simulations.**

I. Introduction

Time synchronization is critical for several ad-hoc and sensor network applications. Data aggregation in sensor networks requires timestamps to combine events occurring within specified time frames. Applications that exploit caching need timestamps to avoid adding stale (or duplicate) information to the cache tables. Time Division Multiple Access (TDMA) scheduling requires accurate knowledge of time lags and continuous synchronization among participating nodes to avoid interference. Time synchronization is also essential for coordinating the sleep and wakeup schedules (duty cycles) of sensors.Several cryptography schemes for ad-hoc networks also require that timestamps be included as part of the digital signature, e.g., in μTESLA [1].

Time synchronization in sensor networks faces unique challenges, most importantly (i) energy-scarcity, (ii) hardware cost, and (iii) dense sensor deployment. The foremost challenge is energy-scarcity, which renders the use of energy-consuming devices, such as Global Positioning Systems (GPS), uneconomical (a GPS-enabled node synchronizes its clock with a satellite). Energy-efficiency also dictates using low overhead protocols, which may trade off accuracy for reduced message exchange. Another challenge is the high cost of adding hardware devices for clock synchronization (such as GPS). The efficacy of approaches such as [2] depends on the distribution of the reference nodes in the network. In addition, in environments with malicious users, attacks can target such highly equipped reference nodes. Finally, dense deployment of sensor nodes necessitates the design of scalable solutions.

A. Synchronization Approaches

Network time synchronization can be classified as low-level synchronization or high-level synchronization (that uses low-level methods). High-level synchronization gives methods for an entire multi-hop network to be synchronized [3], [4], [5], [2], regardless of the underlying protocol used to synchronize the clocks. Low-level synchronization involves synchronizing two or more clocks [2], [6], [7], [8],

– This research has been sponsored in part by NSF grant ANI-0238294 (CAREER).

[5]. Low-level synchronization can be further classified into sender-receiver (SR) and receiver-receiver (RR) approaches. In SR approaches, e.g., [2], a receiver adjusts its clock according to the timestamp of a reference node. In RR approaches, e.g., [6], [9], receivers within 1 hop use a number of synchronization pulses initiated by a "synchronization initiator" to synchronize among themselves. The received pulses are timestamped at every reachable sensor, and these timestamps are exchanged. Every sensor (other than the initiator) can thus compute the time offset and clock skewness with every other sensor in this single-hop region. Fig. 1 depicts a single-hop (i.e., 1-hop) region synchronization using [6], where an initiator sends synchronization pulses in step 1 and nodes v_1, v_2, v_3, and v_4 exchange the timestamps in step 2.

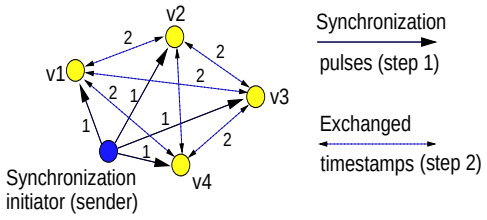


Fig. 1. Receiver-receiver time synchronization.

RR synchronization has two primary advantages: (1) it does not require the presence of GPS-enabled nodes in the network to act as reference nodes, and (2) it gives higher accuracy than SR approaches if timestamping is not possible at the MAC layer. Even if MAC layer timestamping is possible, it is not preferable for a node to follow the clock of another node which does not have a reference clock. For these reasons, we use RR synchronization for low-level synchronization in our work.

B. Application Scenarios

Sensor network applications also have different synchronization requirements according to their traffic patterns. Consider an application where the sensors send timestamped measurements of field temperature to a number of observers. Assume an observer queries the network for temperatures exceeding 150 degrees. An SQL-like query will be in the form: *SELECT Time* T_i*, Temperature* T_e *FROM SENSORS S WHERE* $T_e > 150$. The query may be pre-defined, or a sensor may possess simple query processing capabilities [10]. Two cases may arise: (1) the network is lightly-loaded, i.e., the expected number of replies is small. For energy-efficiency, reactive routing techniques, e.g., Directed Diffusion [10], are used to construct paths between the observer and the responding nodes. Thus, synchronization on the routing paths is sufficient to handle possible data aggregation, and (2) the network is heavily-loaded, e.g., in data streaming applications. The observer divides the data stream into time frames (windows) according to their source timestamps for further analysis. In-network aggregation may be performed using a query processor or simple aggregation operations for pre-defined queries. Therefore, routing paths in the entire network must be pro-actively synchronized. This second case is the primary focus of our work.

Prior approaches have not considered *rapid convergence* of multi-hop network synchronization, especially when observers may query the network from various locations. The presence of multiple mobile unsynchronized observers necessitates separating sensor synchronization from the observers, since multiple (possibly different) reference timestamps may be available.

C. Our Contributions

In this work, we propose a novel framework for high-level time synchronization in clustered, multi-hop sensor networks. Clustering is typically used in applications to facilitate data aggregation and reduce the communication overhead (data aggregation applications typically require timestamping as mentioned above). We will consider the more challenging scenario of using a RR low-level synchronization approach, since it provides *fine-grained* synchronization and does *not* assume the presence of any specially-equipped reference nodes in the network. In contrast to prior work, our primary goal is to achieve *rapid network synchronization* (i.e., in only $N_{iter} = O(1)$ iterations). In addition, our proposed techniques have low message overhead, which is essential for energy-efficiency.

It is important to note that, although node clustering facilitates collaboration for aggregating data and reducing communication overhead, it does *not* solve the network synchronization problem. To the best of our knowledge, our proposed framework for high-level time synchronization is unique in accomplishing rapid multi-hop network synchronization

(without reference nodes in the network). Our goal is end-to-end synchronization of *communicating nodes*, and not common time consensus among *all* network nodes. Our synchronization framework is independent of the particular clustering, routing, and low-level synchronization protocols.

The rest of the paper is organized as follows. Section II briefly surveys related work. Section III defines the problem and objectives. Section IV gives the design rationale and synchronization algorithms. Section V evaluates the proposed algorithms via simulations. Finally, Section VI concludes our work.

II. Related Work

Several protocols have been proposed for network time synchronization. The Reference-Broadcast Synchronization (RBS) [6] is a *low-level* RR protocol that computes the *relative* clock skewness between two neighbors. CesiumSpray [8] also uses RR synchronization and applies a GPS-based hierarchical structure to achieve scalable synchronization. Romer's synchronization mechanism [5] for ad-hoc networks assumes uni-directional links and achieves 1 ms accuracy. Recent work [11] extends Romer's mechanism for higher synchronization accuracy. Cristian [12] proposes a probabilistic approach where synchronization is achieved by sending multiple packets until the error is bound by a pre-defined constant. The Timing-sync Protocol for Sensor Networks (TPSN) [2] and Ping's technique [7] use SR synchronization to achieve high accuracy, *assuming* that timestamping can be done at the MAC layer. The Automatic Self-time Correcting Procedure (ASP) [13] assigns higher probability to nodes with faster clocks to act as beacons. The Lightweight Time Synchronization protocol (LTS) [9] uses a simple RR mechanism, where only 3 packets are exchanged. Biaz and Welch [14] proved that the lower bound on the achievable synchronization under uncertainties in an arbitrary graph is equal to half the graph diameter.

Several protocols were proposed for *high-level* synchronization. Lamport [15] introduced the notion of virtual clocks for event ordering. The post-facto synchronization mechanism [4] was proposed for systems where events do not occur too often, and thus synchronization is performed only when necessary. A high-level synchronization technique was proposed in [2] (which we refer to as multi-hop TPSN) to build a tree hierarchy using message flooding. In [6], high-level synchronization (which we refer to as multi-hop RBS) is achieved by assuming that intersecting regions have nodes that perform inter-regional synchronization. The multi-hop LTS protocol [9] constructs a spanning tree and synchronizes only among neighboring tree levels. The FTSP protocol [16] exploits message flooding, MAC layer timestamping, and clock skew estimation to improve the achieved accuracy over RBS and TPSN. Li and Rus [3] assume that all network nodes need to agree on a clock value, which is different from our goal. Their distributed (diffusion-based) approach requires a time complexity that is *linear* in the number of nodes. In [17], we classify the key research in time synchronization in ad-hoc and sensor networks according to goals and approach.

Clustering ad-hoc networks has been employed for efficient routing, increasing network capacity, supporting data aggregation, and prolonging network lifetime. The reader is referred to [18] for a classification of recent clustering protocols and their deployment challenges.

III. Problem Definition

In this section, we define new terms and functions that will be used throughout this paper, and formulate our problem.

Definition 1: For any two nodes u and v, the function *SYNC(u,v)* = 1 if v is synchronized with u; and SYNC(u,v) = 0 otherwise. SYNC(u,v) is transitive, i.e., if SYNC(u,v) = 1 and SYNC(v,w) = 1, then SYNC(u,w) = 1.

Definition 2: Nodes u and v are said to be *relatively synchronized* if one of them (or both) is aware of the difference $|clock(u) - clock(v)|$. This type of synchronization is asymmetric.

Definition 3: A *strictly synchronized path* $P(v_1, v_{|P|})$ is an ordered set of nodes between a source v_1 and a destination $v_{|P|}$, such that SYNC($v_1, v_{|P|}$) = 1 if $|P| = 2$; otherwise $\forall v_i \in P$, SYNC(v_{i-1}, v_i) = SYNC(v_i, v_{i+1}) = 1, where $1 < i < |P|$.

Definition 4: A *loosely synchronized path* $P(v_1, v_{|P|})$ is an ordered set of nodes between a source v_1 and a destination $v_{|P|}$, such that either SYNC($v_1, v_{|P|}$) = 1, if $|P| = 2$, or $\exists i, j : 1 < i, j < |P|$, such that $v_i, v_j \in P(v_1, v_{|P|})$, $j \geq i$, SYNC(v_1, v_i) = 1, SYNC($v_j, v_{|P|}$) = 1, and the path $P(v_i, v_j)$ is loosely synchronized.

In other words, a strictly synchronized path is one in which every two adjacent nodes on the path are synchronized. On the other hand, a loosely synchronized path is one in which not every two adjacent nodes on the path are synchronized. Loose synchronization requires that a node on the path is synchronized with the source and another node on the path is synchronized with the receiver, and the path among these two nodes is loosely synchronized. For example, in a military field, a soldier may ask if and when any node has sensed a moving tank. One or more nodes (senders) can reply positively and report their timestamps. The soldier should be able to interpret these timestamps according to his clock, regardless of whether *all* the nodes on each path from a sender to the soldier are synchronized. However, if data aggregation occurs on different paths to the soldier, then strict path synchronization is required. Fig. 2 gives an example of strictly versus loosely synchronized paths. Our main focus in this work is on strict synchronization since. In [17], we exploit loose synchronization to synchronize a path in a lightly-loaded network.

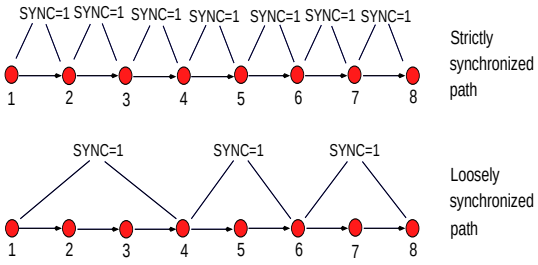


Fig. 2. An example of strictly versus loosely synchronized paths

A. System Model

Assume that n sensors are randomly and independently dispersed in a field. We assume the nodes are quasi-stationary and links are symmetric, but do not assume any infrastructure support. Each node is assumed to have a unique identifier and its transmission power can be tuned (as in Berkeley motes). Nodes are left unattended after deployment and are location unaware. We assume that the sensor radio has an omni-directional antenna that covers a circular range. This range is smaller than the theoretical range to account for signal fading and irregularities [19].

We assume that the network is clustered (e.g., for data aggregation purposes) using distributed protocols that do not require knowledge of node locations, such as [20]. A node affiliates itself with only one cluster. Clusters can be of different sizes (number of nodes). We assume that the application selects a power level, which corresponds to a cluster range R_c, for cluster formation and intra-cluster communication, and reserves a higher level corresponding to range R_t ($R_t > R_c$) for inter-cluster communication. The selection of the best cluster power level is beyond the scope of this work. Our main concern is that the cluster head overlay (i.e., the network of CHs) is connected. This can be achieved if the relation between the number of nodes in this overlay, n_0, and the inter-cluster transmission range R_t satisfies the connectivity conditions specified in [21]. That is, assuming that a node is active with probability p, the necessary condition for connectivity and coverage is that $R_t^2 \geq \frac{c\ log\ n_0}{p\ n_0}$, where $c = \frac{1}{\pi\beta^2}$, and $\beta \leq 0.5$ (this is a generalization of the result in [22]).

We make two assumptions related to the synchronization process: (1) any two neighboring nodes can be synchronized in $O(1)$ time, which we call *direct* synchronization. This is reasonable since two nodes can typically be synchronized by exchanging a fixed number of messages and averaging the delay [12]; and (2) a synchronization initiator node (one that generates synchronization pulses) can synchronize its neighbors, but will not be synchronized with them (as in RR low-level synchronization, e.g., RBS [6]).

B. Goals

The goal of this work is to provide a framework for time synchronization in clustered networks with complexity $N_{iter} \times O(LLSync)$, where N_{iter} is the number of iterations in which a low-level synchronization protocol of complexity $O(LLSync)$ is invoked. We consider relative synchronization, which is sufficient for most sensor networking applications. Our framework will provide mechanisms for synchronizing a 2-hop region or the entire network. We define a 2-hop *region* R in the network as follows. Any two nodes $u, v \in R$ can reach each other in either: (1) one hop, or (2) two hops through a node w, such that $w \in R$. We design mechanisms to support the following requirements:

1) **Regional (intra-cluster) synchronization:** Assume that $\exists$ a node $w \in R$, such that $\forall v \in$

R, distance(v, w)=1. Then, SYNC(v, w)=1 (R is a region in the network).

2) **Relative network synchronization:** For a multi-hop network with a set V of nodes, $\exists$ at least one strictly synchronized routing path P from any $v_i \in V$ to the observer(s) (v_i can be a cluster head (CH) or a non-cluster head).

Relative synchronization means that one node's clock does not have to follow the clock of another node. However, adjacent nodes know the time differences among them. This helps a node v to *interpret* the timestamp associated with an event reported by a neighboring node u before forwarding u's message to the observer. Since our approach enforces relative synchronization between every pair of neighboring nodes, the network is globally synchronized. *Our focus is thus on rapid relative synchronization to the best that the underlying low-level synchronization mechanism can provide.*

IV. A Time Synchronization Framework

In lightly loaded networks, synchronization can be performed "reactively". In contrast, network synchronization must be periodically performed in heavily-loaded networks because queries do not follow a distinct locality pattern. This type of synchronization is "pro-active" (we borrow these terms from the routing literature). Data-driven networks can typically exploit locality of requests more than source-driven networks, unless the observer is mobile and its location significantly changes between the issuance of queries.

The notion of a "region", where single-hop communication is possible between every pair of nodes, is central to many synchronization protocols. For example, in [6] the network is assumed to be divided into regions. The protocol relies on nodes in region intersection areas to propagate synchronization information as data is forwarded. Consider the scenario in Fig. 3 where the application of RBS may fail. In this scenario, the network is divided into three regions around nodes A, B, and C. These regions have no nodes in the intersection areas. Therefore, a packet sent from node 1 to node 8 will not find a synchronized path, although this would have been possible if nodes 2, 5, and 7 were the synchronization initiators. Therefore, the network must be organized such that regions are clearly defined and inter-regional communication is possible, even if regions are non-intersecting.

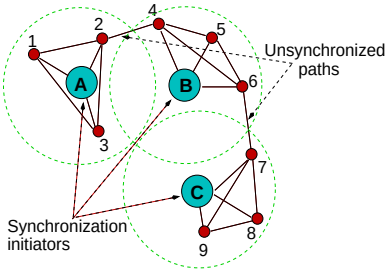


Fig. 3. Failure to find inter-regional synchronized paths

Grid network organization (by node clustering) and intelligent communication power level selection can alleviate the above problem. In a clustered network, a number of nodes act as CHs and communicate with their cluster members, their neighboring CHs, and any close-by observer(s). For node synchronization, clustering can play an important role in: (1) defining synchronization *regions* to be *clusters*; (2) selecting the synchronization initiators in the network (e.g., to be the CHs); (3) adapting to application requirements by expanding or contracting the synchronization regions (cluster sizes); (4) synchronizing 2-hop neighbors through CHs; and (5) enabling scalable and efficient multi-hop synchronization by synchronizing each cluster independently and only relying on the cluster head overlay for synchronizing the network and propagating time information.

A. Intra-cluster Synchronization (SYNC-IN)

For intra-cluster synchronization, all nodes within a cluster need to be synchronized with the cluster head. Fig. 4 gives the pseudo-code for the intra-cluster synchronization algorithm executed at each CH. Let set S hold cluster members that are already synchronized with CH and set V_c hold all cluster members. CH randomly elects an unsynchronized cluster member u to act as an initiator (line 3). If u has any neighbor v that is not in S, it initiates RR synchronization to synchronize v with CH (line 9). Otherwise, u synchronizes itself directly with CH (line 6). The "Synchronize" function (line 6) can use techniques in [12] to directly synchronize the last initiator with CH. Any newly synchronized node with CH is added to S. CH repeats the same process until all the nodes subscribed to its cluster are synchronized with it. Since this is an intra-cluster operation pulses (messages) for intra-cluster

```
// Let S = φ, Cluster = C, Cluster head = CH
// Range is given as an input parameter
1.    V_c ← {v : v ∈ C, v ≠ CH}
2.    WHILE |S| < |V_c|
3.       Pick u ∈ (V_c − S) as synchronization initiator
4.       Send S to u
5.       IF (∄v ∈ S, s.t. v ∈ neighbor(u) and v ≠ CH)
6.          Synchronize(u, CH) // last node
7.          S ← S ∪ {u}
8.       ELSE
9.          RR synchronization with initiator u
10.         S ← S ∪ {v: SYNC(v,CH) = 1}
```

Fig. 4. SYNC-IN: Intra-cluster Synchronization Algorithm

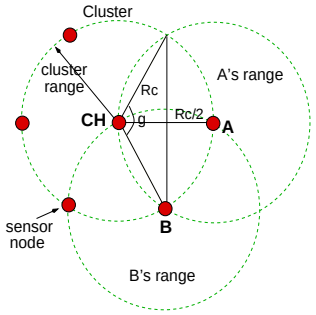


Fig. 5. Worst case scenario for electing initiators to perform intra-cluster synchronization

synchronization are sent using the power level used for intra-cluster communication.

The best synchronization initiator to select is the node closest to the CH because it is likely able to cover most of the nodes in the cluster using range R_c. A CH can maintain lists of neighbors using each of its available power levels, so that neighbors in the smallest level are identified as closest. If the CH cannot deduce the proximity of its cluster members, random selection can be employed.

Correctness: It is easy to see that when the SYNC-IN algorithm terminates, all nodes in the cluster are synchronized with the cluster head, CH. Assume that S_i and S_{i+1} are the sets of nodes synchronized with CH at the beginning of iterations i and $i+1$, respectively. At iteration i, CH picks a node $u \notin S_i$ to act as a synchronization initiator. This results in at least one new synchronized node(s) that was not in S_i. Thus, $|S_{i+1}| > |S_i|$. The algorithm only terminates when $|S| = |V_c|$.

Proposition 1: The SYNC-IN algorithm terminates in N_{iter} = $O(1)$ iterations, where an iteration is $O(LLSync)$ time.

Proof. The number of iterations depends on the part of the cluster that is covered each time a node is elected to act as an initiator. A worst case scenario is demonstrated in Fig. 5 where the elected nodes are very close to the boundary of the cluster, i.e., on the perimeter of the virtual transmission circle of the cluster head.

Assuming that the cluster circle has a perimeter p, the length of the arc covered in circle CH by circle A is $p/3$. In the worst case, the next elected node B is also on the perimeter of CH and A. This covers another arc of CH of length $\pi/3$. We can add at most three other nodes on the perimeter of CH to cover the entire area of CH. Therefore, SYNC-IN requires at most 5 iterations to visit all "non-initiator" nodes and at most another 5 iterations to visit each of the initiators again (we will validate this result in Section V). An "iteration" in this context denotes a low-level synchronization process of a group of nodes in the cluster (as shown in Fig. 1). Interference is avoided since only one node transmits synchronization pulses at any time. □

Proposition 2: The SYNC-IN algorithm requires $O(1)$ messages per node in the cluster (proof can be found in [17]).

B. Inter-cluster Network Synchronization (SYNC-NET)

We now turn to the main focus of this work: *global network synchronization*. We design an algorithm, SYNC-NET, for pro-active time synchronization of the hierarchical network. Since pro-active network synchronization is typically carried out in a heavily-loaded network, our goal is to construct strictly synchronized routing paths among every pair of nodes, and consequently between any node and the observer. If the observer is not included in the clustered network, it can be synchronized with the last node(s) on its routing path(s). SYNC-NET synchronizes the CH overlay and uses SYNC-IN to independently synchronize each cluster. We assume the network has been clustered using any clustering approach [18]. Note that the specific details of how the network is clustered does not affect our synchronization framework. For illustration, we assume that the network is clustered using HEED [20]. HEED uses a probabilistic approach for electing CHs in $O(1)$ time. It assumes that intra-cluster

communication is done at range R_c which is shorter than that used for inter-cluster communications (R_t). Non CH nodes only communicate with their CHs and the CH overlay routes packets to the observer.

Fig. 6 gives the pseudo-code for Algorithm SYNC-NET. Let C_{comm} be the set of nodes in the CH overlay, determined by the execution of the clustering protocol. SYNC-NET will re-cluster the network by applying the same clustering protocol on the set $V - C_{comm}$. This results in another CH overlay with a disjoint set of CHs C_{sync}, i.e., $C_{comm} \cap C_{sync} = \phi$. Since sensor networks are usually dense, we assume that this is possible (asymptotic conditions are given in Section IV-B.1). The two CH overlays have different roles. The first overlay, C_{comm}, is the overlay that will later be used for "time-aware" forwarding. CHs in C_{comm} are also responsible for applying SYNC-IN for intra-cluster synchronization. In contrast, CHs in the overlay C_{sync} are only used to synchronize the set C_{comm}, and therefore, other nodes in the network do not need to register themselves with CHs in C_{sync}. CHs in C_{sync} discover their neighboring heads in C_{comm} (within range R_t) by overhearing the exchanged messages of C_{comm}, e.g., routing updates (line 1). A "neighbor" in the remainder of this section refers to a node within a range R_t. CHs in C_{comm} do not need to discover their neighbors in C_{sync}[1].

Network synchronization proceeds as follows. In the first iteration of SYNC-NET, a node $v \in C_{sync}$ elects to become a synchronization initiator for its neighbors in C_{comm} with a small probability P_s, $0 < P_s \leq 1$ (line 4). The elected initiator v synchronizes a CH $u \in C_{comm}$ that covers an intersecting region with that of v, with all the CH neighbors of u in C_{comm} (line 6). This probabilistic election reduces redundant message exchange. In addition, starting with a small value of P_s allows gradual network synchronization and thus reduces interference. We will study the number of messages exchanged via simulations in Section V.

At the end of the first iteration, a CH that has elected to act as a synchronization initiator exits SYNC-NET (line 7). A node $u \in C_{comm}$ that detects that it is currently synchronized with all its neighbors in C_{comm} broadcasts a "SYNC-DONE" message, and exits SYNC-NET. If all neighbors

[1]We assume that the inter-cluster routing protocol will exploit a neighbor as the next hop in the inter-cluster routing path.

```
// Two CH overlays are used C_comm ∩ C_sync = φ
// The following is executed at every node v ∈ C_sync
1.  S_nbrs[v] ← {u : u ∈ C_comm, distance(u,v) ≤ R_t}
2.  Max_iter ← ⌈log_2 1/P_s⌉ + 1, iter ← 0
3.  REPEAT
4.    iter ← iter + 1, r ← Uniform(0,1)
5.    IF r < P_s
6.      Send SYNC beacons with range R_t
7.      EXIT SYNC-NET
8.    S_covered = {u : u ∈ C_comm,
                    u has sent message "SYNC-DONE"}
9.    IF S_covered ≠ S_nbrs
10.     P_s ← min(P_s × 2, 1)
11. UNTIL (iter = Max_iter OR S_covered = S_nbrs)
```

Fig. 6. SYNC-NET: Inter-cluster Synchronization at $v \in C_{sync}$

in C_{comm} of CH $v \in C_{sync}$ have sent "SYNC-DONE" messages, v exits SYNC-NET. Otherwise, v doubles its P_s value (line 10), and proceeds to the next iteration. This process is repeated until P_s reaches 1. Note that when a node exits SYNC-NET, it ignores any newly received synchronization pulses. SYNC-NET is asynchronous, i.e., all nodes need not start executing it simultaneously. Observe that using SYNC-IN and SYNC-NET, non-cluster head nodes need not maintain any synchronization information, while a CH in C_{comm} only maintains relative synchronization information with its cluster members and its neighboring CHs in C_{comm}.

1) **Density Model:** Since SYNC-NET requires two independent CH overlays (C_{comm} and C_{sync}), we specify what node density is required to be able to form such overlays. Assume that n nodes are uniformly and independently dispersed at random in an area $R = [0, L]^2$. Assume that R is divided into N square cells of size $\frac{R_c}{\sqrt{2}} \times \frac{R_c}{\sqrt{2}}$ (thus $N = \frac{2L^2}{R_c^2}$), where a cell is an approximation of a cluster. This implies that every node in each cell can reach every other node residing in the same cell using a transmission range R_c. We have formulated a general density model in [23] that allows forming k connected CH overlays. This requires a minimum cell occupancy of at least $k > 1$ nodes asymptotically almost surely (a.a.s.).[2] We can simply use the special case $k = 2$ of the following theorem (which we proved in [23]):

Theorem 1: Let $\eta(n, N)$ be a random variable

[2]A cell is an approximation of a cluster, and thus R_c defines the required density, and R_t is used to define connectivity.

denoting the minimum number of nodes in a cell. For a fixed arbitrary $k > 0$, assume that n nodes are uniformly and independently distributed at random in an area $R = [0, L]^2$. Assume R is divided into N square cells, each of side $R_c\sqrt{2}$. If $R_c^2 n \geq aL^2 ln\ N$ for some constant $a \geq 2$, $R_c \ll L$, and $n \gg 1$, then $lim_{n,N\to\infty} E[\eta(n, N)] = k$ iff $k \sim ln\ N$.

Corollary 1: Each cell will a.a.s. have two distinct CHs, one in C_{comm} and the other in C_{sync}.

The proof is not included due to space limitation and can be found in [17].

2) **Protocol Analysis:** *Correctness:* When all nodes in C_{sync} terminate SYNC-NET, every node $u \in C_{comm}$ is synchronized with all its neighbors in C_{comm}. To prove this, assume that R_t is selected such that it covers every CH in the complete neighborhood of cells around any cell A. The complete neighborhood around A constitutes all the eight cells surrounding A (this can be ensured by enforcing a relation between R_t and R_c). Assume that $\exists a_1 \in C_{comm}$, such that $S_{nbr}(a_1)$ is the set of neighbor CHs of a_1 in C_{comm}. We can prove synchronization by contradiction. Assume that $\exists u \in S_{nbr}(a_1)$, such that $SYNC(a_1, u) = 0$. We assume that Theorem 1 holds (where $k = 2$), and therefore every cell contains two CHs (one in C_{comm} and the other in C_{sync}). There are two cases for u:

Case 1. The cell of node u is within the complete neighborhood of the cell of a_1. For example, as depicted in Fig. 7, a_1 is in cell A, u can be one of $\{b_1, d_1, e_1, f_1\}$. In this case, the CH $a_2 \in C_{sync}$ can reach all of these nodes, and therefore can synchronize them with a_1, which is a contradiction.

Case 2. The cell of node u is not in the neighborhood of the cell of a_1 (cell A). For example, cell G in Fig. 7 is one such case. Assume that a_1 and g_1 are neighbors, while a_2 and g_1 are not. However, there must exist another CH in a neighbor cell that belongs to C_{sync} (node d_2 in this example) which will not exit SYNC-NET until a_1 and g_1 are synchronized and send "SYNC-DONE" messages. This means that a_1 and g_1 will be synchronized, which is a contradiction.

Proposition 3: $\forall v \in C_{sync}$, SYNC-NET terminates in $N_{iter} = O(1)$ iterations, assuming that the clustering protocol takes $O(1)$ time.

Proof. Since SYNC-NET continues until P_s reaches 1, the number of iterations, N_{iter} can be computed as: $N_{iter} \leq \lceil log_2 \frac{1}{P_s} \rceil + 1$, which is $O(1)$. We assume that the underlying clustering protocol ensures that the number of CH neighbors of each CH is constant. Thus, neighborhood discovery also takes $O(1)$. □

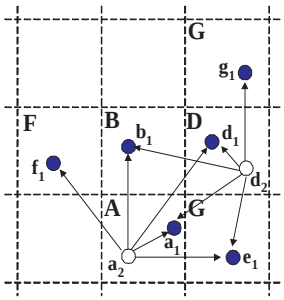


Fig. 7. Example of node synchronization using SYNC-NET. $\{a_1, b_1, c_1, d_1, e_1, f_1\} \subset C_{comm}$ *and* $\{a_2, d_2\} \subset C_{sync}$

Proposition 4: SYNC-NET has an $O(1)$ message overhead per node in each CH overlay.

Proof. A node may elect to become an initiator in C_{sync} only once, and sends $O(1)$ synchronization pulses. A node to be synchronized in C_{comm} replies to synchronization pulses until all its neighbors are synchronized with it. The number of neighbors is $O(1)$ (depends on the ratio R_t/R_c). Thus, every node in C_{comm} also sends $O(1)$ messages (clustering message overhead per node is $O(1)$ [20]). □

The worst case synchronization accuracy of SYNC-NET is approximately $O(\sqrt{N} \times q)$, where N is the number of cells in the network, and q is the accuracy of the low-level synchronization mechanism. We consider only the CH overlay, since (except for the first/last hop), communication proceeds through it. We assume CHs are non-neighbors, and thus the CH overlay can be approximated by a 2-D mesh network. Synchronization accuracy depends on the length of the path from the source to the destination, L_p, and the underlying low-level synchronization mechanism. In the worst case, L_p can be as long as the network diameter, which is $O(\sqrt{N})$. Therefore, the accuracy provided by SYNC-NET is $O(\sqrt{N} \times q)$. For example, consider a sensor network with $n = 10,000$, $N = 100$ and RBS [6] as the underlying low-level synchronization scheme. RBS achieves an absolute accuracy per hop in the order of $q \approx 29\mu$s on Berkeley sensor motes, as measured in [2]. Therefore, according to the above discussion, SYNC-NET achieves an accuracy of $10 \times 29 \times 10^{-6} = 290\ \mu$s on the longest expected path, in the worst case when errors add up.

C. Secure Synchronization

We comment on the security features of our framework since secure communication is essential in security-sensitive applications. Clustering is inherently resistant to several types of attacks [24], such as sinkhole (all traffic directed through one node), Sybil (node claiming multiple identities), and bogus routing updates. This is due to the continuous topology update and rotation of node roles. Consequently, our synchronization approach is also robust to these attacks in clustered networks. Our framework can resist outsider attacks (e.g., eavesdropping) by link-layer encryption of timestamps and identity authentication. However, both clustering and synchronization are vulnerable to insider attacks in which one or more nodes are compromised. If compromised nodes are well situated in the network such that they frequently lie on the path of data to the observer, they can tamper with the included timestamps. One possible approach to mitigate this effect is to transmit reports redundantly using multiple node-disjoint CH overlays for routing.

V. Performance Evaluation

We verify via simulations the properties of our proposed approaches for intra-cluster and inter-cluster synchronization. We developed a C-based simulator that is scalable to thousands of nodes. Our simulator assumes a MAC layer that ensures no packet losses. This is reasonable for three reasons. First, all the presented results are comparative and use the same simplifications for all scenarios. Second, we assume that the MAC layer uses orthogonal channels to allow simultaneous intra-cluster and inter-cluster transmissions. Imperfections, such as collisions, have little impact on the performance of SYNC-NET. The occurrence of collisions can be significantly reduced by using TDMA among cluster members. Third, the typical packet sizes in current systems are small (the default is 36 bytes for TinyOS [25]), which reduces the probability of collisions. We assume that nodes are deployed independently at random in the field. Packets are routed through C_{comm} using a greedy geographic routing mechanism. In this mechanism, the next hop of a packet is the one that is geographically closest to the destination. Every reported result is the average of 100 experiments on different topologies.

A. Intra-cluster Synchronization

We explore the two possibilities for selecting synchronization initiators that were discussed in Section IV-A: (1) randomly, and (2) closest to the cluster head. We vary the number of nodes per cluster from 10 to 1000 to study how fast the algorithm terminates for different node densities: node density ranges from 0.1 nodes/m^2 to 10 nodes/m^2. The transmission range ($R_c = 10$ m). Fig. 8 illustrates that: (1) the number of iterations until SYNC-IN converges is less than 8 for different densities, which agrees with the result in Proposition 1, and (2) the number of iterations when the closest neighbors are selected as initiators is lower than that when random initiators are selected, as expected. Selecting the closest neighbors as initiators, however, adds overhead on the CH for discovering neighbors at each power level smaller than the cluster power level.

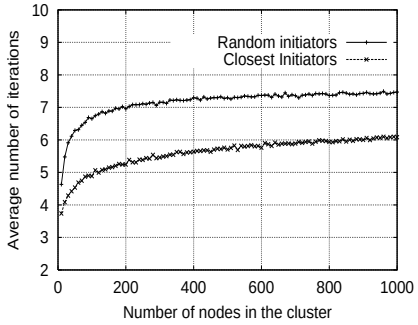


Fig. 8. Convergence of the SYNC-IN protocol

B. Inter-cluster Synchronization

We assume that nodes are dispersed uniformly and independently in a 100×100 m^2 area. We use 2500 nodes, unless otherwise specified. Throughout this section, we use the term "neighbors" in SYNC-NET to refer to two CHs which can communicate using a transmission range R_t. Two neighbor CHs can belong to the same CH overlay, or belong to different overlays. As defined in Section IV-B, we use C_{comm} and C_{sync} to refer to the forwarding and synchronizing overlays, respectively, and use "node density" to refer to the number of nodes per cluster.

1) **Effect of varying SYNC-NET parameters:** We investigate SYNC-NET parameters with respect to: (i) the average number of neighbors as the transmission range grows; (ii) the convergence speed as the node density increases; and (iii) how probabilistic synchronization initiation reduces the number of

messages exchanged in the network. We keep R_c fixed in most experiments. Changing R_c only results in changing the average number of CHs in C_{comm} and C_{sync}, and has no impact on the performance of SYNC-NET.

To verify that the nodes in C_{sync} can synchronize the C_{comm} overlay, we conduct an experiment where the inter-cluster range R_t is varied from double to four times the cluster range R_c (R_c = 10 m). We use average node densities of 2.5 nodes/cell, 5 nodes/cell, and 10 nodes/cell, for 500, 1000, and 2000 nodes, respectively. Results (given in our technical report [17]) illustrate that the average number of neighbors in C_{comm} for each node in C_{sync} exceeds five, for all values of R_t. This number gives an indication of number of nodes typically synchronized when a node $v \in C_{sync}$ acts as a synchronization initiator. Node density, as long as it satisfies Theorem 1, does not appear to have as significant an impact on the results in this case.

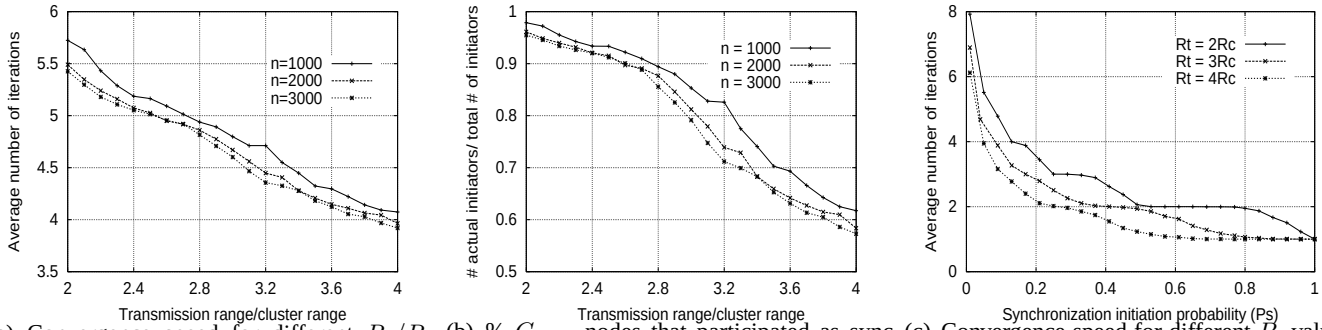
We now perform two experiments to verify Proposition 3. In the first experiment, we compute the actual average number of iterations in these experiments to compare with the analytical upper bound. In both experiments, the transmission range R_t varies from $2R_c$ to $4R_c$, and $R_c = 6$ m. Experiments are performed for n = 1000, 2000, and 3000. This results in node densities ranging from 2 nodes/cell to 6 nodes/cell. Fig. 9(a) shows that SYNC-NET terminates more rapidly as R_t grows relative to R_c. We also examine the number of exchanged messages associated with longer transmission ranges. Fig. 9(b) shows that the percentage of actual number of synchronization initiators out of the total number of viable initiators in C_{sync} is about 95% for $R_t = 2R_c$, and about 60% for $R_t = 3R_c$. This is a significant reduction in message exchange, compared to the simple approach of making every node in C_{sync} a synchronization initiator.

Finally, we study the effect of P_s on the convergence speed and message overhead of SYNC-NET. We let P_s range from 0.01 to 1 and set the number of nodes n to 2000. The cluster range R_c is 6 m, while the transmission range R_t varies from $2R_c$ to $4R_c$. Fig. 9(c) shows that (1) the average number of iterations until all the nodes in C_{comm} are synchronized with their neighbors is strictly less than the maximum specified by Proposition 3, and (2) as P_s increases, termination is faster, since the synchronization probability goes to 1 quickly. This is not a desirable behavior, however, since more nodes in C_{sync} send redundant synchronization pulses. Results in [17] also show that smaller values of P_s generally result in a lower average number of initiators, and hence lower message overhead. Therefore, we surmise that a small P_s (e.g., 5%) will help achieve two goals: fast termination and lower message exchange. Even for a small P_s, the convergence speed is within practical bounds.

2) **Comparisons to other approaches:** We compare the performance of SYNC-NET to another RR approach, the Diffusion-based protocol [3], and an SR approach, multi-hop TPSN [2]. Observe, however, that this comparison is only for demonstration, since protocols like TPSN and Diffusion-based assume that the application needs to achieve time *consensus* in the network, which is not our goal. In TPSN, a reference node (carrying GPS) initiates synchronization by forming a hierarchy using message flooding, while SYNC-NET does not rely on the presence of any infrastructure support. We will demonstrate that SYNC-NET provides comparable performance to multi-hop TPSN. We focus on three performance metrics: (i) convergence speed; (ii) message overhead, which is directly related to energy savings; and (iii) perceived accuracy, which is the goal of any synchronization protocol.

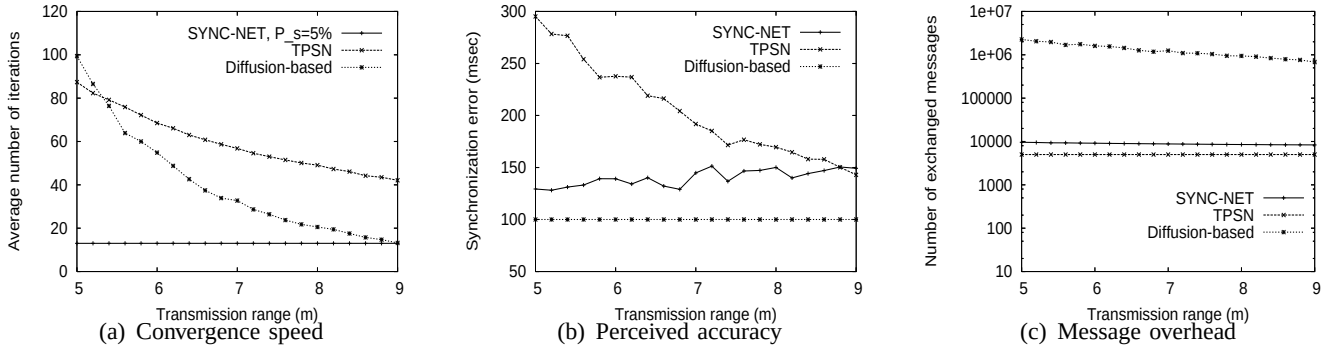
In our first experiment, the cluster range for SYNC-NET (R_c), TPSN neighbor discovery, and Diffusion-based communications, varies from 5 m to 9 m. We plot the average number of iterations for multi-hop TPSN and the Diffusion-based protocol. We also plot the maximum number of iterations of SYNC-NET for P_s = 0.05, (which gives 6 iterations). We assume that an $O(1)$ clustering protocol is used, and hence add 7 iterations to the SYNC-NET iterations to construct C_{sync}. Fig. 10(a) illustrates a significant difference in convergence speed between SYNC-NET and the other two protocols, especially for the more typical small transmission ranges. In fact, multi-hop TPSN and Diffusion-based protocols are expected to be even slower in a 1-dimensional space since the number of iterations is expected to be $O(n)$ in the average case.

In our second experiment, we compare the three protocols in terms of their perceived accuracy (or error propagation). We assume that the Diffusion-based target accuracy γ = 100 msec. The algorithm terminates only when this γ is achieved. We assume RBS low-level synchronization is employed



(a) Convergence speed for different R_t/R_c ratios (b) % C_{sync} nodes that participated as sync initiators for different R_t/R_c ratios (c) Convergence speed for different P_s values

Fig. 9. Convergence speed and number of synchronization initiators in SYNC-NET



(a) Convergence speed (b) Perceived accuracy (c) Message overhead

Fig. 10. SYNC-NET performance compared to TPSN [2] and the Diffusion-based approach [3]. Note that TPSN assumes the presence of reference nodes, in contrast to our model which assumes that nodes have minimum capabilities.

by SYNC-NET for an absolute RR synchronization error value of mean 29 μs/hop, while TPSN low-level synchronization achieves an absolute SR error value of mean 17 μs. These values were reported in [2] based on an implementation of RBS and TPSN, and experimental results on Berkeley sensor motes. We consider the synchronization error propagated across the network as reports are transmitted from a source closest to the bottom left corner of the network area to an observer that is closest to the upper right corner (the longest path). Fig. 10(b) illustrates that both SYNC-NET and the Diffusion-based approach provide comparable synchronization granularity for the network. Multi-hop TPSN has higher error for smaller ranges. The reason for SYNC-NET performing better than TPSN although it has a higher relative error propagation is that SYNC-NET uses the CH overlay for forwarding, and thus has a fewer number of hops than TPSN.

Finally, we compute the message overhead for the three approaches to demonstrate their energy efficiency. Fig. 10(c) shows the price paid by the Diffusion-based approach to achieve its target accuracy. Results (shown on a log scale) also demonstrate that multi-hop TPSN requires the least message overhead since timing information is only forwarded and copied by the nodes. SYNC-NET overhead is slightly higher than multi-hop TPSN but significantly lower than the Diffusion-based approach. The primary contributor to the overhead in SYNC-NET is the RBS low-level synchronization at both the intra-cluster and inter-cluster levels.

VI. Conclusions

In this work, we proposed a distributed, high-level time synchronization framework for clustered sensor networks that provides scalability and rapid convergence. We define synchronization regions as clusters, where two-hop communication can take place through a cluster head. We designed fully distributed protocols for intra-cluster synchronization (SYNC-IN), and inter-cluster synchronization (SYNC-NET). We showed in [17] how to adapt SYNC-NET for flat networks. Results show that by gradual network synchronization (through a probability P_s), message overhead can be significantly

reduced. Results also show that SYNC-NET can achieve a synchronization accuracy that is comparable to other approaches

REFERENCES

[1] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks," in *Proc. of the ACM MobiCom Conference*, 2001.

[2] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync protocol for sensor networks," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2003.

[3] Q. Li and D. Rus, "Global clock synchronization in sensor networks," in *Proc. of the IEEE INFOCOM Conference*, March 2004.

[4] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, April 2001.

[5] K. Romer, "Time synchronization in ad-hoc networks," in *Proc. of the ACM International Symposium on Mobile and Ad-Hoc Networking and Computing (MobiHoc)*, October 2001.

[6] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of OSDI*, 2002.

[7] S. Ping, "Delay measurement time synchronization for wireless sensor networks," Intel Research, IBR-TR-03-013, Tech. Rep., June 2003.

[8] P. Verissimo, L. Rodrigues, and A. Casimiro, "CesiumSpray: A precise and accurate global time service for large-scale systems," *Special Issue on the Global Time in Large-scale Distributed Real-time Systems, Journal of Real-time Systems*, vol. 12, no. 3, November 1997.

[9] J. V. Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *Proc. of the ACM Workshop on Sensor Networks and Applications (ACM WSNA)*, September 2003.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proc. of the ACM MobiCom Conference*, 2000.

[11] L. Meier, P. Blum, and L. Thiele, "Internal synchronization of drift-constraint clocks in ad-hoc sensor networks," in *Proc. of the ACM International Symposium on Mobile and Ad-Hoc Networking and Computing (MobiHoc)*, May 2004.

[12] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, pp. 146–158, 1989.

[13] J.-P. Sheu, C.-M. Chao, and C.-W. Sun, "A clock synchronization algorithm for multi-hop wireless ad-hoc networks," in *Proc. of the IEEE Int'l Conference on Distributed Computing Systems*, March 2004.

[14] S. Biaz and J. L. Welch, "Closed form bounds for clock synchronization under simple uncertainty assumptions," *Elsevier Information Processing Letters*, vol. 80, pp. 151–157, 2001.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[16] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2004.

[17] O. Younis and S. Fahmy, "On time synchronization in multi-hop sensor networks," Purdue University, Tech. Rep. CSD-TR-04-020, June 2004.

[18] O. Younis, M. Krunz, and S. Ramasubramanian, "Clustering in wireless sensor networks: Recent developments and deployment challenges," *IEEE Network Magazine*, vol. 20, no. 3, pp. 20–25, May 2006.

[19] G. Zhou, T. He, S. Krishnamurthy, and J. Stankovic, "Impact of radio irregularity on wireless sensor networks," in *Proc. of the ACM International Conference on Mobile Systems, Applications, and Services (ACM MobiSys)*, June 2004.

[20] O. Younis and S. Fahmy, "Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach," in *Proc. of the IEEE INFOCOM Conference*, Hong Kong, March 2004, an extended version appeared in IEEE Transactions on Mobile Computing, 3(4), pp. 366-379, Oct-Dec 2004.

[21] S. Shakkottai, R. Srikant, and N. Shroff, "Unreliable sensor grids: Coverage, connectivity and diameter," in *Proc. of the IEEE INFOCOM Conference*, March 2003.

[22] P. Gupta and P. R. Kumar, "Critical power for asymptotic connectivity in wireless networks," *Stochastic Analysis, Control, Optimizations, and Applications: A Volume in Honor of W.H. Fleming, W.M. McEneaney, G. Yin, and Q. Zhang (Eds.)*, Birkhauser, 1998.

[23] O. Younis, S. Fahmy, and P. Santi, "Robust communications for sensor networks in hostile environments," in *the Twelfth International Workshop on Quality of Service (IWQoS'04)*, Montreal, Canada, June 2004.

[24] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasures," in *IEEE Workshop on Sensor Network Protocols and Applications*, May 2003.

[25] TinyOS, http://www.tinyos.net, 2006.

PLACE PHOTO HERE

Ossama Younis is an assistant research scientist in computer engineering at the University of Arizona. He received B.S. and M.S. degrees in computer science from Alexandria University, Egypt, and his Ph.D. degree in computer science from Purdue University in August 2005. He has served on the Technical Program Committee for several international conferences. His research interests include wireless sensor network protocols and applications, Internet tomography, and cognitive-radio networks. He is a member of the ACM and the IEEE.

PLACE PHOTO HERE

Sonia Fahmy is an associate professor of Computer Science at Purdue University. She received her PhD from the Ohio State University in 1999. Her research interests are in the design and evaluation of network architectures and protocols. She is currently investigating Internet tomography, overlay networks, network security, and wireless sensor networks. She received the National Science Foundation CAREER award in 2003 and the Schlumberger foundation technical merit award in 2000 and 2001, and the OSU presidential fellowship for dissertation research in 1998. She is a member of the ACM, Phi Kappa Phi, Sigma Xi, and Upsilon Pi Epsilon, and a senior member of the IEEE.