A Scalable Framework for Distributed Time Synchronization in Multi-hop Sensor Networks

Ossama Younis and Sonia Fahmy
Department of Computer Science, Purdue University
250 N. University Street, West Lafayette, IN 47907–2066, USA
E-mail: younis@ece.arizona.edu, 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 clustering-based time synchronization framework for 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. We propose two novel fully-distributed protocols, SYNC-IN and SYNC-NET, for regional and network synchronization, respectively, and prove that N_{iter} is $O(1)$ for both protocols. We exploit the tradeoff between rapid convergence (and consequently energy-efficiency) and perceived accuracy. 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.**

Index Terms— **Sensor networks, time synchronization, node clustering**

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. 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 to reduce energy consumption at idle times, e.g., in TinyDB [1]. Several cryptography schemes for ad-hoc networks also require that timestamps be included as part of the digital signature, e.g., in μTESLA [2]. Other time-sensitive applications include object tracking and navigation.

Although standard time beaconing using WWV [3] is deployed across North America and Europe, sensors may be deployed in areas where there is no coverage. In addition, the sensors have to be able to tune to the specific frequencies of the generated beacons, which may not always be feasible. 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 GPS, uneconomical[1]. Energy-efficiency also dictates using low overhead protocols, which may trade off accuracy for reduced message exchange. Another challenge is the cost of adding hardware devices for clock synchronization (such as GPS). This cost is typically high compared to the price of the sensor itself. The efficacy of approaches such as TPSN 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. This is because node synchronization must be frequently invoked in cases where the application requires fine granularity, or the clock frequencies of nodes are significantly different.

A. Synchronization Approaches

Network time synchronization can be classified as low-level synchronization or high-level synchronization (that uses low-level methods). Low-level synchronization involves the process of synchronizing two or more clocks [4], [5], [6], [7], [8]. Low-level synchronization can be further classified into sender-receiver (SR) and receiver-receiver (RR) approaches. In sender-receiver approaches, such as TPSN [4], a receiver adjusts its clock according to the timestamp received from a reference node. This node is referred to as the synchronization initiator. In receiver-receiver approaches, such as RBS [5] or LTS [9], receivers within 1 hop use a number of synchronization pulses initiated by a sender to synchronize among themselves. The received pulses are timestamped at every reachable sensor, and these timestamps are exchanged. Every sensor (other than the sender) can thus compute the time offset and clock skewness with every other sensor in this single-hop region. The sender is not synchronized with the receivers in this case. Fig. 1 depicts a single-hop (i.e., 1-hop) region synchronization using RBS [5], 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. The time complexity of

– This research has been sponsored in part by NSF grant ANI-0238294 (CAREER), and the Schlumberger Foundation technical merit award.

[1]A node equipped with a GPS (Global Positioning System) antenna synchronizes its clock with a satellite.

this low-level synchronization approach depends on how fast all the nodes in the 1-hop region can access the channel to broadcast their measured timestamps.

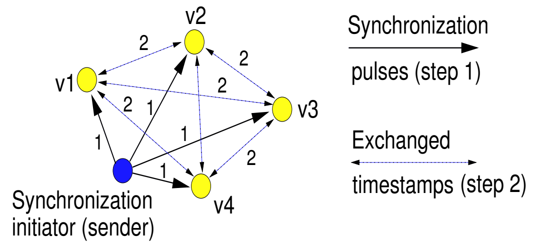

v1
v2
v3
v4
1
2
Synchronization initiator (sender)
Synchronization pulses (step 1)
Exchanged timestamps (step 2)

Fig. 1. Time synchronization of a 1-hop region using the receiver-receiver approach

Receiver-receiver synchronization has two primary advantages: (1) it does not require/prefer 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 receiver-receiver synchronization for low-level synchronization in our work.

In contrast to these low-level approaches, high-level synchronization gives methods for an entire multi-hop network to be synchronized [10], [11], [8], [4], regardless of the underlying protocol used to synchronize the clocks. Multi-hop RBS [5] also belongs to this category, but does not give practical methods to organize the network into synchronized connected regions, which is the focus of our work.

B. Application Scenarios

Sensor network applications have different synchronization requirements according to their types (or traffic patterns). In source-driven applications, nodes periodically send reports to an observer about a measured parameter(s), whereas in data-driven (query-driven) applications, an observer queries the network about the occurrence of an event[2]. 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 using systems such as TinyDB [1]. Based on the network load, two cases arise.

Case 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 [12], 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.

Case 2. The network is heavily-loaded, e.g., a large percentage of the sensors is expected to report their responses to the observer, or responses are periodically reported. For example, in 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 [1], 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 new framework for high-level time synchronization in multi-hop sensor networks. Our framework integrates synchronization with node clustering to construct two-tiered, synchronized networks. We will consider the more challenging scenario of using a receiver-receiver 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). Such a fast response is especially important if the network is dense and the synchronization algorithm must be frequently invoked (whenever the network goes out-of-sync). In addition to rapid convergence, 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.

D. Organization of the Paper

The remainder of this paper is organized as follows. Section II briefly surveys related work. Section III defines the terms used throughout the paper, and defines the problem. Section IV gives the design rationale and synchronization algorithms. Section V evaluates the proposed algorithms via simulations. Section VI outlines deployment issues for our framework. Finally, Section VII summarizes our work and suggests future research directions.

[2]A source-driven network can be viewed as a data-driven one, where the sensors respond to a pre-defined periodic query.

II. Related Work

Several protocols have been proposed for network time synchronization. The Reference Broadcast Synchronization (RBS) [5] is a *low-level* receiver-receiver protocol that computes the *relative* clock skewness between two neighbors and does not need any infrastructure support. CesiumSpray [7] also uses receiver-receiver synchronization and applies a GPS-based hierarchical structure to achieve scalable synchronization. Romer's synchronization mechanism [8] for ad-hoc networks assumes uni-directional links and achieves 1 ms accuracy. Recent work [13] extends Romer's mechanism for higher synchronization accuracy. Cristian [14] proposes a probabilistic approach where synchronization is achieved by sending multiple packets until the error is bound by a pre-defined constant. Basic TPSN [4] and Ping's technique [6] use sender-receiver synchronization to achieve higher accuracy than RBS, *assuming* that timestamping can be done at the MAC layer. The Automatic Self-time Correcting Procedure (ASP) [15] assigns higher probability to nodes with faster clocks to act as beacons. The Network Time Protocol (NTP) [16] is a sender-receiver synchronization approach widely deployed in the Internet, which has proved to be scalable and robust. The Lightweight Time Synchronization protocol (LTS) [9] uses a simple receiver-receiver mechanism, where only 3 packets are exchanged. *tiny-sync* and *mini-sync* [17] use sender-receiver synchronization and assume that the nodes are organized in a tree topology for data aggregation. Biaz and Welch [18] 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 [19] introduced the notion of virtual clocks for event ordering. The post-facto synchronization mechanism [11] 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 [4] (which we refer to as multi-hop TPSN) to build a tree hierarchy using message flooding. In [5], 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. *mini-sync* [17] also assumes a tree topology of sensors and uses relative synchronization. Convergence of this approach is dependent on the tree depth. Li and Rus [10] 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. We give a taxonomy of time synchronization approaches for wireless ad-hoc networks in [20].

Clustering ad-hoc networks has been employed for efficient routing [21], [22], increasing network capacity [23], supporting data aggregation, and prolonging network lifetime [24], [25]. Several clustering approaches have been proposed – most notable are weight-based approaches, which cluster based upon a certain parameter (weight) or a number of parameters, such as node degree or residual energy [26], [27], [24], [28], [29], [25]. Other approaches cluster the network by selecting a dominating set, such as in [30], [31], [32]. A third class of approaches is heuristic-based, e.g., cluster the network using node identifiers [21].

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 $1 < i < |P|$.

In other words, a strictly synchronized path is one in which every two adjacent nodes on the path are synchronized.

A. System Model

Assume that n sensors are randomly 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 range of the sensor omni-directional antenna covers a circular range. This range is smaller than the actual range to account for signal fading and obstacles.

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 [14]; and (2) a synchronization initiator node (one that generates synchronization pulses) can synchronize its neighbors, but will not be synchronized with them (as in receiver-receiver low-level synchronization, e.g., RBS [5]). This is because sender-receiver mechanisms make strong assumptions about reference nodes and MAC layer capabilities, as discussed above.

B. Goals

The goal of this work is to provide a framework for time synchronization 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 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).

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

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. SEAD [33] proposed fixing the aggregation points in the network so that a mobile observer can access aggregated values by querying any nearby node (access point) on the aggregation tree. Our synchronization framework can be quite useful in this case, since it synchronizes the network independently from the observer.

The notion of a "region" is central to many synchronization protocols. For example, in [5] 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. A region in this context is an area in which single-hop communication is possible between every pair of nodes. To understand the problem caused by non-intersecting regions in the network, consider the scenario in Fig. 2 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. This problem depends on node density, node distribution, and transmission range. Therefore, the network must be organized such that regions are clearly defined and inter-regional communication is possible, even if regions are non-intersecting.

Node clustering and intelligent communication power level selection can alleviate the above problem. In a clustered network, a number of nodes act as cluster heads and communicate with their cluster nodes, their neighboring cluster heads, 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 cluster heads); (3) adapting to application requirements by expanding or contracting the synchronization regions (cluster sizes); (4) synchronizing 2-hop neighbors through cluster heads; and (5) enabling scalable

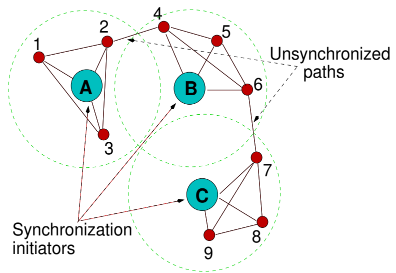


Fig. 2. Failure to find inter-regional synchronized paths

and efficient multi-hop synchronization by synchronizing each cluster independently and only relying on the cluster head overlay (i.e., the network of cluster heads) for synchronizing the network and propagating time information. This reduces message overhead and increases scalability.

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 power level choices depend on the node capabilities, MAC protocol, node density, and transmission patterns, and aim at maximizing spatial reuse and reducing energy consumption. The selection of the best cluster power level is beyond the scope of this work. Our main concern is that the cluster head overlay 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 [34]. 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 [35]). We will define a density model in Section IV-B.1 to give the necessary conditions for connectivity in our network synchronization algorithm.

We now discuss algorithms for intra-cluster and inter-cluster synchronization. For intra-cluster synchronization, cluster members are synchronized with their cluster heads. For inter-cluster synchronization, nodes in the cluster head overlay are synchronized.

A. *Intra-cluster Synchronization (SYNC-IN)*

For intra-cluster synchronization, all nodes within a cluster need to be synchronized with the cluster head, since communication to the observer(s) always takes place through the cluster head. A cluster head in the receiver-receiver approach cannot synchronize itself with the cluster nodes if it acts as a synchronization initiator, as stated in Section III-A. In this case, the cluster head elects nodes from within its cluster to act as initiators. It continues doing so until all the nodes subscribed to its cluster are synchronized with the cluster head. Since this is an intra-cluster operation, and to avoid interference with neighboring clusters regardless of the MAC protocol, pulses (messages) for intra-cluster synchronization are sent with the cluster range R_c (i.e., using the power level used for cluster formation and *intra-cluster* communication).

```
// Let S = $\phi$, Cluster = C, Cluster head = CH
// Range is given as an input parameter
1.    $V_c \leftarrow \{v : v \in C,\ v \neq CH\}$
2.    WHILE $|S| < |V_c|$
3.        Pick $u \in (V_c - S)$ as synchronization initiator
4.        Send $S$ to $u$
5.        IF ($\nexists v \in S$, s.t. $v \in$ neighbor($u$) and $v \neq CH$)
6.            Synchronize($u$, $CH$) // last node
7.        ELSE
8.            Receiver-receiver sync. with initiator $u$
9.        $S \leftarrow$ S $\cup$ {$v$: SYNC($v$,$CH$) = 1}
```

Fig. 3. SYNC-IN: Intra-cluster Synchronization Algorithm

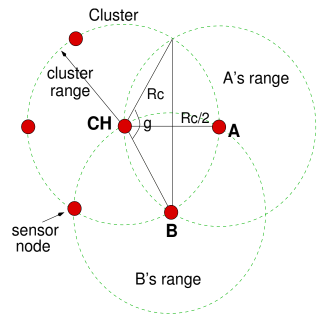


Fig. 4. Worst case scenario for electing initiators to perform intra-cluster synchronization

This also increases energy efficiency. Fig. 3 gives the pseudo-code for the intra-cluster synchronization algorithm executed at each cluster head. The "Synchronize" function (line 6) can use techniques in [14] to directly synchronize the last initiator with the cluster head.

The best candidate to select as a synchronization initiator is the node closest to the cluster head. This is because such a close neighbor is likely able to cover most of the nodes in the cluster using the cluster range R_c. A cluster head can maintain lists of neighbors using each of its available power levels, so that neighbors in the smallest level are identified as closest (references [23] and [36] similarly suggest using variable power levels, but with the goal of increasing the network capacity). If the cluster head cannot deduce the proximity of its cluster members, random selection can be employed. During the operation of this protocol, nodes are synchronized with the cluster head and removed from the candidate set of synchronization initiators.

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. 4 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 $p/3$. This is because the opposite angle g is $2\pi/3$ (since $cos(g/2) = 0.5$). 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, the SYNC-IN algorithm 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 $O(1)$ message transmissions per node in the cluster.

Proof. A node participates in the synchronization process in only one iteration and sends only one message. A node may act as an initiator only once and sends $O(1)$ pulses. Thus, each node (other than the cluster head) sends $O(1)$ messages during the entire synchronization process. The cluster head sends two messages at each iteration (one to elect an initiator and one for synchronization). Since the number of iterations is $O(1)$, the cluster head also sends $O(1)$ messages. □

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

We now design an algorithm, SYNC-NET, for pro-active time synchronization of the entire hierarchical network. Since pro-active network synchronization will be carried out in a heavily-loaded network (otherwise reactive synchronization of a routing path suffices), 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 strictly synchronizes the cluster head overlay and uses SYNC-IN to synchronize each cluster. We assume the network has been clustered using any clustering approach, e.g., [26], [24], [28], [32], [22]. Approaches that result in well-distributed cluster heads in the network are ideal because interference is reduced, and intra-cluster communication can proceed in parallel using the cluster range R_c. Fig. 5 illustrates an organization tree rooted at the *observer* to aggregate data from the entire network.

SYNC-NET operates on a clustered network. Let C_{comm} be the set of nodes in the cluster head overlay. SYNC-NET will re-cluster the network using the set $V - C_{comm}$. This results in another cluster head overlay with a disjoint set of cluster heads 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 cluster head overlays have different roles. The first overlay, C_{comm}, is the overlay that will later be used for "time-aware" forwarding.

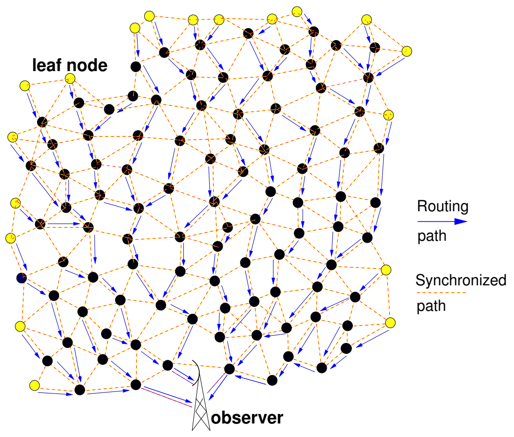


Fig. 5. An observer collecting data from a synchronized network using tree routing. Light-colored nodes are the tree leaves. Only cluster heads are shown.

```
// The following is executed at every node $v \in C_{sync}$
1.  $S_{nbrs}[v] \leftarrow \{u : u \in C_{comm},\ distance(u,v) \leq R_t\}$
2.  $Max\_iter \leftarrow \lceil log_2 \frac{1}{P_s} \rceil + 1,\ iter \leftarrow 0$
3.  REPEAT
4.      $iter \leftarrow iter + 1,\ r \leftarrow$ Uniform(0,1)
5.      IF $r < P_s$
6.          Send SYNC beacons with range $R_t$
7.          EXIT SYNC-NET
8.      $S_{covered} = \{u : u \in C_{comm},$
                    $u$ has sent message "SYNC-DONE"$\}$
9.      IF $S_{covered} \neq S_{nbrs}$
10.         $P_s \leftarrow min(P_s \times 2, 1)$
11. UNTIL ($iter = Max\_iter$ OR $S_{covered} = S_{nbrs}$)
```

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

Cluster heads in C_{comm} are also responsible for applying SYNC-IN for intra-cluster synchronization. In contrast, cluster heads in the overlay C_{sync} are only used to synchronize the set C_{comm}.

Network synchronization proceeds as follows. Each cluster head $v \in C_{sync}$ discovers its neighbor heads in C_{comm} using the inter-cluster transmission range R_t. A "neighbor" in the remainder of this section refers to a node within a range R_t.[3] 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 probability P_s, $0 < P_s \leq 1$ (say 5%). The elected initiator v synchronizes a cluster head $u \in C_{comm}$ that covers an intersecting region with that of v, with all the cluster head neighbors of u in C_{comm}. 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 cluster head that has elected to act as a synchronization initiator exits SYNC-NET. 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 in C_{comm} of cluster head $v \in C_{sync}$ have sent "SYNC-DONE" messages, v exits SYNC-NET. Otherwise, v doubles its P_s value, 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 received synchronization pulses. Fig. 6 gives the pseudo-code for Algorithm SYNC-NET. The algorithm 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 cluster head in C_{comm} only maintains relative synchronization information with its cluster members and its neighboring cluster heads in C_{comm}.

1) **Density Model:** Since SYNC-NET requires two independent cluster head overlays (C_{comm} and C_{sync}), we must 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 [37] that allows forming k connected cluster head overlays. This requires a minimum cell occupancy of at least $k > 1$ nodes asymptotically almost surely (a.a.s.).[4] We can simply use the special case $k = 2$ of the following theorem (which we proved in [37]):

Theorem 1: Let $\eta(n, N)$ be a random variable that denotes the minimum number of nodes in a cell. For any 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 cluster heads, one in C_{comm} and the other in C_{sync}.

Proof. Assuming that Theorem 1 holds for $k = 2$, then every cell contains at least two nodes a.a.s., and consequently must contain two cluster heads, one for each cluster head overlay. The property holds a.a.s. because the nodes considered in constructing C_{sync} do not include the ones previously selected in C_{comm}. □

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 cluster head 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 cluster heads 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

[3]We assume that the multi-hop inter-cluster routing protocol will exploit a neighbor as the next hop in the inter-cluster routing path, which must be the case if R_t is the inter-cluster communication range.

[4]We view a cell as an approximation of a cluster, and thus R_c is used to define the required density, and R_t is used to define connectivity.

every cell contains two cluster heads (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 cluster head $a_2 \in C_{sync}$ 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 cluster head 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.

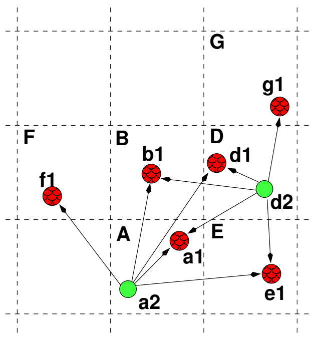


Fig. 7. Example of node synchronization using SYNC-NET. $\{a_1, b_1, c_1, d_1, e_1, f_1\} \subset C_{comm}$ *and* $\{a_2, d_2\} \subset C_{sync}$

Proposition 3: At every node $v \in C_{sync}$, SYNC-NET terminates in $N_{iter} = O(1)$ iterations, assuming that the clustering protocol takes $O(1)$ time (example $O(1)$ clustering protocols include [24], [28], [31]).

Proof. Since SYNC-NET continues until P_s reaches 1, the number of iterations, N_{iter} can be computed as:

$$N_{iter} \leq \lceil log_2 \frac{1}{P_s} \rceil + 1, \tag{1}$$

which is $O(1)$. For example, if P_s is 10%, then N_{iter}, is 5. We assume that the underlying clustering protocol ensures that the number of cluster head neighbors of each cluster head is constant. Thus, neighborhood discovery is also an $O(1)$ operation. □

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

Proof. A node may elect to become a synchronization 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 $O(1)$ [24], [28], [31]). □

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 cluster head overlay, since (except for the first/last hop), communication proceeds through it. We assume cluster heads are non-neighbors, and thus the cluster head 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 [5] 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 [4]. Therefore, according to the above discussion, SYNC-NET achieves an accuracy of $10 \times 29 \times 10^{-6} = 290$ μs on the longest expected path, in the worst case when errors add up.

V. Performance Evaluation

In this section, we verify via simulations the properties of our proposed approaches for intra-cluster and inter-cluster synchronization. We developed our own simulator that is simpler than existing simulators, such as ns-2. This simplicity allows the simulator to scale to thousands of nodes. The simulator, however, does not model all the details of a wireless channel and MAC protocol, e.g., interference problems. This is a reasonable approach for three reasons. First, all the results presented below are comparative and use the same simplifications for all scenarios. Second, we assume that the MAC layer uses CDMA (or orthogonal channels) to allow simultaneous intra-cluster and inter-cluster transmissions. Intra-cluster collisions are further reduced by using TDMA among cluster members. Third, the typical packet sizes are small (the default is 29 bytes for TinyOS [38]), which reduces the probability of collisions, especially when aggregation is used.

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 cluster head for discovering neighbors at each power level smaller than the cluster power level. Note that we have assumed in this experiment that there is an infinite number of transmission ranges below the cluster range.

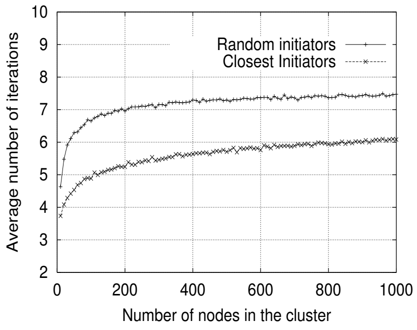


Fig. 8. Convergence of the SYNC-IN protocol

B. Inter-cluster Synchronization

We assume that nodes are dispersed uniformly and independently in a $100 \times 100\ m^2$ area. We use 2500 nodes, unless otherwise specified. Throughout this section, we use the term "neighbors" in SYNC-NET to refer to two cluster heads which can communicate using a transmission range R_t. Two neighbor cluster heads can belong to the same cluster head 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) **Comparisons to other approaches:** We compare the performance of SYNC-NET to the Diffusion-based protocol [10], which is a receiver-receiver approach, and multi-hop TPSN [4], which is a sender-receiver approach. 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 the goal of our work. In addition, in TPSN, a reference node initiates synchronization by forming a hierarchy using message flooding, while SYNC-NET does not rely on the presence of any infrastructure support in the network. We will demonstrate that SYNC-NET provides comparable performance to multi-hop TPSN. We focus on three performance metrics:

1) convergence speed, which is the primary focus of our work;
2) message overhead, which is directly related to energy savings; and
3) 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 for 100 different topologies. 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 [24], [28], [31] is used, and hence add 7 iterations to the SYNC-NET iterations to construct C_{sync}. Fig. 9(a) illustrates a significant difference in convergence speed between SYNC-NET and the other two protocols (which have linearly decreasing curves), 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 by SYNC-NET for an absolute receiver-receiver synchronization error value of mean 29 μs introduced at every hop, while TPSN low-level synchronization achieves an absolute sender-receiver error value of mean 17 μs. These values were reported in [4] based on an implementation of RBS and TPSN, and experimental results on Berkeley sensor motes. For simplicity, data is forwarded using greedy geographic routing. 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). We also assume worst case error propagation for both multi-hop TPSN and SYNC-NET. Fig. 9(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 cluster head overlay for forwarding, and thus has a fewer number of hops than TPSN.

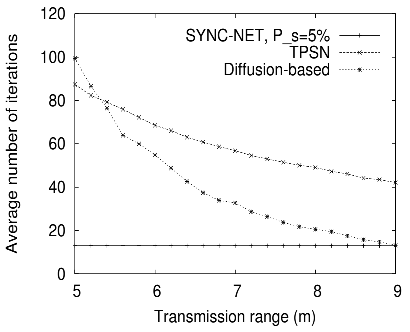
Finally, we compute the message overhead for each approach. Fig. 9(c) demonstrates the price paid by the Diffusion-based approach to achieve its target accuracy. Results (shown on a log scale) also demonstrate that multi-hop TPSN requires the least message exchange overhead since timing information is 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.

2) **Effect of varying SYNC-NET parameters:** We investigate SYNC-NET with respect to:

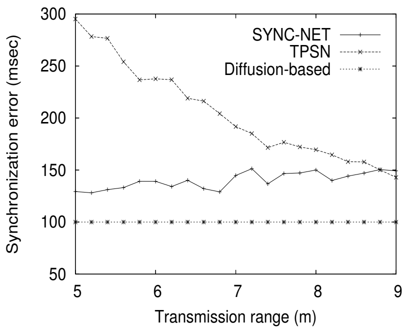
1) the average number of neighbors as the transmission range grows;
2) the convergence speed as the node density increases; and
3) how probabilistic synchronization initiation reduces the number of messages exchanged in the network.

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

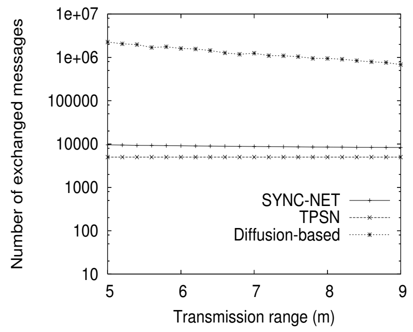
To verify that the nodes in C_{sync} can synchronize the C_{comm} overlay, we conduct an experiment where the inter-cluster range R_t is varied from double to four times the cluster range R_c (R_c = 10 m). We use average node densities of 2.5 nodes/cell, 5 nodes/cell, and 10 nodes/cell, for 500, 1000, and 2000 nodes, respectively. Results (given in our technical report [20]) illustrate that the average number of neighbors in



(a) Convergence speed



(b) Perceived accuracy



(c) Message overhead

Fig. 9. SYNC-NET performance compared to TPSN [4] and the Diffusion-based approach [10]

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, since the average number of neighbors is dominated by the ratio R_t/R_c.

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 three values of n (the number of nodes): 1000, 2000, and 3000. This results in node densities that range from about 2 nodes/cell to 6 nodes/cell. Fig. 10 shows that, as expected, 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. 11 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, since the percentage of non-participating nodes in C_{sync} reflects the percentage of reduced message overhead.

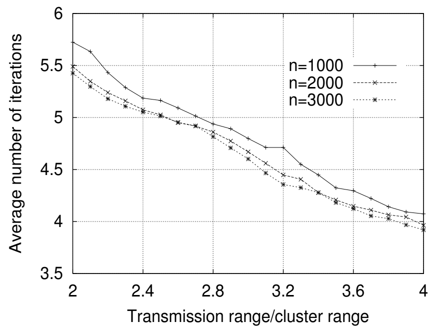


Fig. 10. Convergence speed for different Rt/Rc ratios

Finally, we study the effect of the synchronization probability P_s on the convergence speed and message overhead of

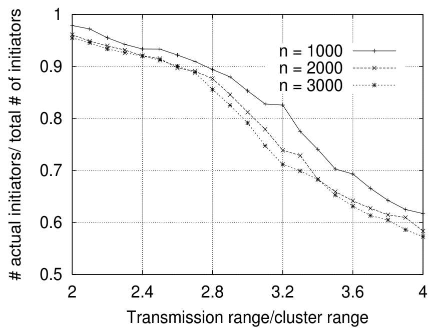


Fig. 11. Percentage of nodes from C_{sync} that participated as synchronization initiators

SYNC-NET. The probability P_s ranges from 0.01 to 1 in our experiments. The number of nodes n is set to 2000. The cluster range R_c is 6 m, while the transmission range R_t varies from $2R_c$ to $4R_c$. Fig. 12 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. This is demonstrated in Fig. 13, where smaller values of P_s generally result in a lower average number of initiators, and hence lower message overhead. The curves show more than one local minimum, which means that each transmission range has a unique behavior with different synchronization probabilities. We have not considered the effect of interference in our simulations, which will indeed be magnified by sending simultaneous long-range synchronization pulses by neighboring nodes in C_{sync}. Therefore, we surmise that a small value of P_s (e.g., 5%) will help achieve both goals: fast termination and lower message exchange. Even for a small probability, the convergence speed is within practical bounds.

VI. Deployment Considerations

In this section, we discuss several design and deployment considerations for our framework:

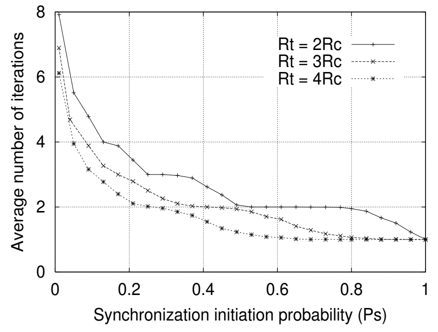


Fig. 12. Convergence speed for different P_s values

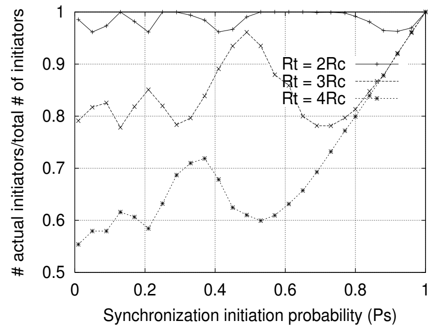


Fig. 13. Percentage of nodes from C_{sync} that participated as synchronization initiators

Synchronization in non-clustered networks: We use node clustering to define synchronization regions and ensure that the synchronization process has predictable asymptotic behavior. Our proposed algorithms can, however, be adapted to synchronize flat (non-clustered) networks as follows. Assume that nodes use a transmission range R_c. Nodes can elect themselves as synchronization initiators using an iterative probabilistic approach (with a range R_c) as in the original algorithm. Elected initiators send synchronization pulses using a transmission range R_t ($R_t > R_c$). The network then goes through another round of synchronization, excluding the initiators of the previous round from acting as initiators. Neighboring initiators in the two rounds can then privately undergo one-to-one synchronization. In this scenario, every node will have to store its relative synchronization information with its neighbors to maintain synchronized paths.

Triggering synchronization: If clock skewness is variable due to conditions such as temperature, SYNC-NET must be periodically triggered for relative synchronization. SYNC-NET can be triggered at any node by timer expiration or by message exchange. For example, if the application can tolerate an error of up to 1 ms, two clocks lose synchronization at a maximum rate of 1 μsec per minute, and the network diameter is 40 hops, then SYNC-NET should be triggered every $1/(1 \times 10^{-3} \times 40/2)$ = 50 minutes. A node in C_{sync} that detects synchronization pulses from a neighbor should immediately start executing SYNC-NET. Since it is difficult to compute the clock skewness for the entire network during network operation, each node can set a timer to trigger synchronization taking into consideration an estimate of the expected skewness.

Re-clustering and synchronization: In clustered networks, clustering is periodically re-triggered to achieve certain goals, such as load-balancing. Synchronization should be triggered at least as frequently as clustering to maintain synchronized clustered network routing paths to the observer(s).

Effect of fading, path asymmetry, and packet loss: The wireless medium imposes challenges on SYNC-NET. Synchronization pulses and timestamp reports may not be received because of signal fading, path asymmetry, and packet loss. Using small values of P_s for gradual network synchronization reduces collisions. In addition, losing synchronization pulses is not critical since a node in C_{comm} does not exit the SYNC-NET protocol unless it is synchronized with all its neighbors. A node in C_{comm} can also rely on pulses from more than one node in C_{sync} to be synchronized with each neighbor. Therefore, pulse loss and path asymmetry will cause more nodes in C_{sync} to act as initiators, but SYNC-NET will still operate correctly.

Synchronization probability: Throughout the paper, we have considered a constant P_s value for all nodes. A variable P_s can be used to attain certain properties, such as favoring nodes with high connectivity for faster convergence, or favoring nodes with high remaining energy. We plan to explore these options in our future work.

Sleeping nodes: Putting cluster heads to sleep may be problematic for cluster head overlay connectivity and synchronized path availability. To avoid these problems, a node that belongs to C_{comm} should not switch to the sleeping mode unless it is not participating in the routing or synchronization processes.

Fault-tolerance: Networks deployed in hostile environments may experience unexpected node failures. This may hinder communication if the failed nodes are in the cluster head overlay. This problem is mitigated by the fact that applications that use clustered networks periodically re-cluster the network to maintain connectivity, and adjust to changing network conditions due to dispersion of new nodes or failure and energy depletion of others. Fault tolerance can also be achieved by maintaining backup cluster head overlays which can be synchronized using the pulses generated by the initiators of C_{sync}. If the clustering protocol can self-heal in case of failures, then the newly elected heads should trigger local synchronization by soliciting pulses from the C_{sync} nodes.

Applicability: Our protocols are applicable to any wireless network setting that requires scalable and rapid convergence. Examples include several data dissemination, peer-to-peer systems (e.g., BitTorrent [39]), and network monitoring applications, running on handheld or laptop devices in a wireless ad-hoc network.

VII. Conclusions

In this work, we proposed a distributed, high-level time synchronization framework for multi-hop sensor networks that integrates node synchronization with node clustering for scalability and fast convergence. Our framework serves the two major classes of network applications, namely, source-driven and data-driven network applications. 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 show (in Section VI) how to adapt SYNC-NET for flat networks.

For inter-cluster synchronization, results demonstrate that the proposed analytical density model is easily achieved in moderately dense networks, where the expected number of nodes per cell exceeds two. 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, while terminating more rapidly and significantly reducing message overhead. In our future work, we plan to study the effect of node distribution in the network, and the impact of variable probability P_s values, via both simulations and testbed experiments.

References

[1] S. Madden, "The design and evaluation of a query processing architecture for sensor networks," Ph.D. dissertation, MIT, 2004.

[2] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," in *Proceedings of ACM MOBICOM*, 2001.

[3] R. Beehler, "Time/Frequency Services of the U.S. National Bureau of Standards and Some Alternatives for Future Improvement," *Electronics and Telecommunications Engineers*, vol. 27, 1981. [Online]. Available: citeseer.ist.psu.edu/johnson96dynamic.html

[4] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync Protocol for Sensor Networks," in *Proceedings of ACM SenSys*, November 2003.

[5] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time synchronization Using Reference Broadcasts," in *Proceedings of OSDI*, 2002.

[6] S. Ping, "Delay Measurement Time Synchronization for Wireless Sensor Networks," Intel Research, IBR-TR-03-013, Tech. Rep., June 2003.

[7] 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.

[8] K. Romer, "Time Synchronization in Ad-hoc Networks," in *Proceedings of ACM MobiHoc*, October 2001.

[9] J. V. Greunen and J. Rabaey, "Lightweight Time Synchronization for Sensor Networks," in *Proceedings of the Second Workshop on Sensor Networks and Application (WSNA'03)*, September 2003.

[10] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," in *Proceedings of IEEE INFOCOM*, March 2004.

[11] 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.

[12] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *Proceedings of ACM MOBICOM*, 2000.

[13] L. Meier, P. Blum, and L. Thiele, "Internal Synchronization of Drift-Constraint Clocks in Ad-hoc Sensor Networks," in *Proceeding of ACM MobiHoc*, May 2004.

[14] F. Cristian, "Probabilistic Clock Synchronization," *Distributed Computing*, pp. 146–158, 1989.

[15] J.-P. Sheu, C.-M. Chao, and C.-W. Sun, "A Clock Synchronization Algorithm for Multi-Hop Wireless Ad-Hoc Networks," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, March 2004.

[16] D. L. Mills, "Network Time Protocol (Version 3): Specification, Implementation and Analysis," *RFC 1305*, March 1992.

[17] M. L. Sichitiu and C. Veerarittiphan, "Simple, Accurate Time Synchronization for Wireless Sensor Networks," in *Proceeding of IEEE WCNC*, March 2003.

[18] 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.

[19] 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.

[20] O. Younis and S. Fahmy, "On Time Synchronization in Multi-hop Sensor Networks," Purdue University, Tech. Rep. CSD-TR-04-020, June 2004.

[21] C. R. Lin and M. Gerla, "Adaptive Clustering for Mobile Wireless Networks," in *IEEE Journal on Selected Areas in Communications*, September 1997.

[22] B. McDonald and T. Znati, "Design and Performance of a Distributed Dynamic Clustering Algorithm for Ad-Hoc Networks," in *Annual Simulation Symposium*, 2001.

[23] V. Kawadia and P. R. Kumar, "Power Control and Clustering in Ad Hoc Networks," in *Proceedings of IEEE INFOCOM*, April 2003.

[24] O. Younis and S. Fahmy, "Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach," in *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004, an extended version appeared in IEEE Transactions on Mobile Computing, 3(4), Oct-Dec 2004.

[25] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, October 2002.

[26] S. Basagni, "Distributed Clustering Algorithm for Ad-hoc Networks," in *International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, 1999.

[27] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "Initializing Newly Deployed Ad-hoc and Sensor Networks," in *Proceedings of ACM MOBICOM*, September 2004.

[28] H. Chan and A. Perrig, "ACE: An Emergent Algorithm for Highly Uniform Cluster Formation," in *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, January 2004.

[29] M. Chatterjee, S. K. Das, and D. Turgut, "WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks," *Cluster Computing*, pp. 193–204, 2002.

[30] F. Kuhn and R. Wattenhofer, "Constant-Time Distributed Dominating Set Approximation," in *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, July 2003.

[31] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, "Max-Min D-Cluster Formation in Wireless Ad Hoc Networks," in *Proceedings of IEEE INFOCOM*, March 2000.

[32] S. Banerjee and S. Khuller, "A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks," in *Proceedings of IEEE INFOCOM*, April 2001.

[33] H. S. Kim, T. Abdelzaher, and W. H. Kwon, "Minimum-Energy Asynchronous Dissemination to Mobile Sinks in Wireless Sensor Networks," in *Proceedings of ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2003.

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

[35] 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.

[36] J. Gomez and A. Campbell, "A Case for Variable-Range Transmission Power Control in Wireless Multihop Networks," in *Proceedings of IEEE INFOCOM*, March 2004.

[37] 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.

[38] "TinyOS," http://www.tinyos.net, 2005.

[39] BitTorrent, "http://www.bitconjurer.org/BitTorrent," 2004.