Temporal Predicate Detection Using Synchronized Clocks

Ajay D. Kshemkalyani, *Senior Member*, *IEEE*

Abstract—Advances in clock synchronization techniques allow an approximated global time in ubiquitous environments. This paper presents an event stream-based online algorithm that fuses the data reported from the processors in such a network to detect time-based predicates. The algorithm has low space, time, and message complexities. The paper also considers the detection of simultaneous events as a special case.

Index Terms—Event streams, sensor networks, ad hoc networks, data fusion, time, synchronized clocks, intervals.

1 Introduction

Advances in wireless communication and sensor and actuator technologies have given rise to ubiquitous systems, e.g., ad hoc networks and sensor networks [1], [25]. Here, numerous small devices operate collectively and form a dynamic ambient network that connects each device to more powerful networks and processing resources. Monitoring events in such resource-constrained environments is a challenge. Event-based data streams represent relevant state changes at the processes that are monitored [10], [15], [20]. This paper gives an online algorithm to detect *predicates defined on the relative occurrence of events* from event streams that are reported by the various components.

Examples. Consider a study of wildlife habitat that monitors the activity of different animal and bird species at a watering hole or at a river. The study is interested in inferring relationships such as: (tigers drink *before* deer) or (tigers drinking *overlaps* with lions drinking) or (elephants and storks drink *concurrently*). The period for which a sensor senses a particular species visiting the watering hole is an interval of interest at that sensor. Such intervals from different sensors, reporting data about various species, are reported in data streams to a central processor. At this processor, the data from the interval streams is fused to detect the predicates of interest.

As another example, consider a secure banking application, where entry into the bank vault is possible only by a set S of two or more *simultaneous* biometric passwords for authentication. Here, the authorized people concerned must simultaneously and remotely present their passwords to gain entry to the vault. Here, the predicate for success is (S_1 and S_2 and $\ldots S_n$) simultaneously. When a biometric password is applied at sensor i, the corresponding time interval gets reported to a central monitoring process, which fuses similar interval information it receives from the other biometric sensors. The combined information from all of the sensors is fused to detect the predicate for successful access to the bank vault.

A formal model specification and problem definition are given in Section 2. Section 3 describes the related work. Section 4 gives the processing at the sensors/processes. Sections 5 and 6 give two algorithms to detect predicates defined on the relative occurrence of events. Section 7 gives the algorithm to detect simultaneous events by deriving it from the predicate detection algorithms. Section 8 presents a discussion.

• *The author is with the Department of Computer Science (MC 152), 851 South Morgan Street, University of Illinois at Chicago, Chicago, IL 60607-7053. E-mail: ajayk@cs.uic.edu.*

Manuscript received 20 July 2006; revised 23 Jan. 2007; accepted 15 May 2007; published online 11 June 2007.
Recommended for acceptance by L. Schwiebert.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0281-0706.
Digital Object Identifier no. 10.1109/TC.2007.70749.

2 System Model and Problem Definition

The process execution model is as follows: E_i is the linearly ordered set of discrete events executed by process P_i in an execution. Variable x local to process P_i is denoted as x_i. The control program at P_i monitors a "local predicate."

Definition 1. *A* local predicate ϕ_i *at* P_i *is any predicate defined on variables local to process* P_i *and that can be evaluated by* P_i.

In general, the local predicate can include references to physical time as well as temporal logic operators, as long as the predicate can be locally evaluated. The monitoring program at each process tracks the local time *intervals* of interest, which are the durations during which the *local predicate* is true. Such an interval at process P_i is identified by the (totally ordered) corresponding adjacent events within E_i for which the local predicate is true. Intervals are denoted by capitals X, Y, and Z. Fig. 1a shows processes $P_1 \ldots P_n$. Fig. 1b is a timing diagram that shows the intervals during which the local predicates are true.

Event streams from the processes report intervals in which the local predicates are true to a central data fusion server P_0 (see Fig. 1a). Information about the reported intervals is "fused" and examined to *detect* global states of the execution that satisfy a given global predicate. In this paper, we consider global predicates that are defined on the relative timing occurrences of intervals across different processes. Furthermore, the global predicate must be expressible in conjunctive form, i.e., $\phi = \bigwedge_i^t \phi_i$, which is thus a conjunct over the local predicates ϕ_i, and timing relationships are included in the conjunction operation $\bigwedge^t$. The following examples are different formulations of a predicate to detect an explosion:

- ($temp_i > 80°$C and $audio_i > 45$ decibels) AFTER ($lumens_j > 1000L$). This predicate denotes an explosion at P_j of a certain intensity that is also detected by P_i some distance away. The light is seen at explosion location j, while the corresponding noise and temperature increases are experienced some time later at a location i.
- There is an instant at which ($temp_i > 120°$C) SIMULTANEOUSLY ($audio_j > 50$ decibels). This is a simplified form of the above predicate, specified using only the temperature and noise parameters at two different locations.

Clocks in sensor networks, ad hoc networks, and wireless networks [24]—when synchronized via GPS [16], NTP [19], or any of the many efficient synchronization protocols for wired as well as wireless media, such as those surveyed in [5], [7], [8], [21], [23]—allow the assumption about an approximate single global time axis. With synchronized clocks, a *distributed execution* can be modeled as the interleaving of all the local executions E_i on a common time axis. This simplifies the detection of a global state [4] in the ubiquitous environment, which is essentially a distributed asynchronous message-passing system. A *global state* is defined to contain one local state of each process. Using a common time axis, a global state can be specified 1) as occurring at the same time instant at each process or 2) in terms of specific relationships among the local states (one local state from each process).

For a single time axis, it has been shown [2], [9] that there are 13 ways in which two time intervals can be related to one another on that time axis. For intervals X and Y, the 13 relations are illustrated in Fig. 2. The set of these 13 relations is denoted $\Re$. There are six pairs of inverses as shown and *equals* is its own inverse.

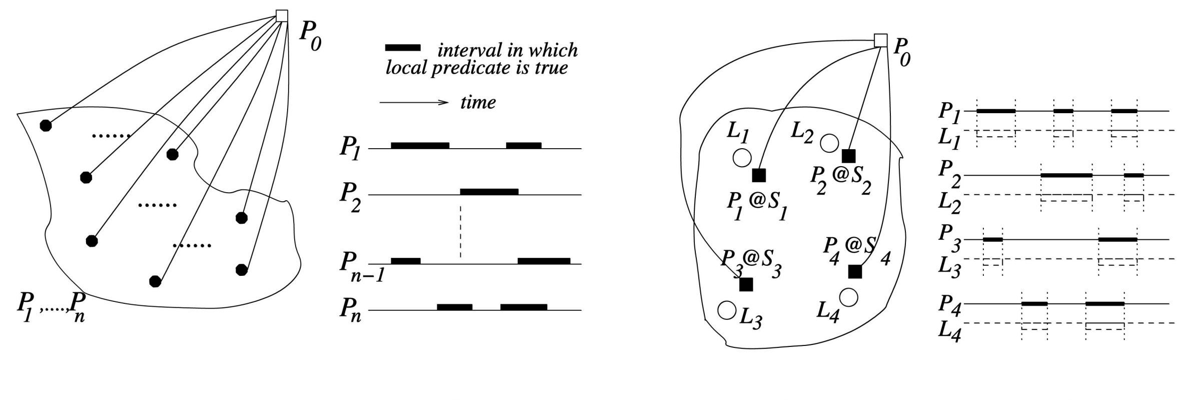

Fig. 1. Organization of the network. (a) $P_1, \ldots P_n$ are processes and P_0 is the data fusion server. (b) Timing diagram for a *computer network* (no sensing). (c) A *sensor network* showing processes P_1-P_4 at sensors S_1-S_4, respectively, sensing locations L_1-L_4. (d) Timing diagram for intervals at locations L_1-L_4, as modeled by processes P_1-P_4 in the sensor network.

Problem Definitions. In this paper, the event streams generated by processes $P_1, \ldots P_n$ need to be fused at the central server P_0 to solve the following global predicate detection problem [3], [13].

Problem *Predicate_Rel***.** Given a relation $r_{i,j}$ from $\Re$ for each pair of processes P_i and P_j, identify in an online manner the earliest intervals (if they exist), one from each process, such that each relation $r_{i,j}$ is satisfied for the (P_i, P_j) pair.

Example specification. We assume that intervals X_i, Y_j, and Z_k occur at different locations P_i, P_j, and P_k, respectively, but global time is available in the system at all sites. Two example specifications of predicates are:

1. (X_i precedes Y_j) AND (X_i overlaps Z_k) AND (Y_j finishes Z_k).
2. (X_i overlaps Y_j) AND (Y_j contains Z_k) AND (Z_k met by X_i).

The problem in each case is to identify the global state in a distributed execution when the predicate is true. Example solutions are illustrated in Fig. 3.

Problem *Predicate_Rel* restricts the input to a single relationship $r_{i,j}$ for process pair (P_i, P_j). Problem *Predicate_Rel** allows $r^*_{i,j}$ to be a subset of $\Re$ and a solution can satisfy any of the relations in $r^*_{i,j}$ for the (P_i, P_j) pair.

Problem *Predicate_Rel****.** Given a set of relations $r^*_{i,j} \subseteq \Re$ for each pair of processes P_i and P_j, determine online the earliest intervals, if they exist, one from each process, such that any one of the relations in $r^*_{i,j}$ is satisfied (by the intervals) for each (P_i, P_j) pair. If a solution exists, identify the relationship from $\Re$ for each pair of intervals in the solution.

Example specification. Assume that intervals X_i, Y_j, and Z_k occur at different locations P_i, P_j, and P_k, respectively. Let $r^*_{i,j} = \{p, m, q\}, r^*_{i,k} = \{o, c\}, r^*_{j,k} = \{f, ob\}$. The set of intervals in Fig. 3a satisfies this specification, but there may be other mutual placements of the intervals that can also satisfy this specification.

To illustrate an application of *Predicate_Rel**, the paper derives a solution to the basic task of detecting the *simultaneous* occurrence of each local predicate ϕ_i (at all i).

Problem *Simultaneous***.** Identify the earliest set of intervals $\mathcal{I} = \{I_1, I_2, \ldots I_n\}$ (if such a set exists), where I_i is from process P_i and ϕ_i is true in I_i such that there is some instant of time that belongs within all of these intervals.

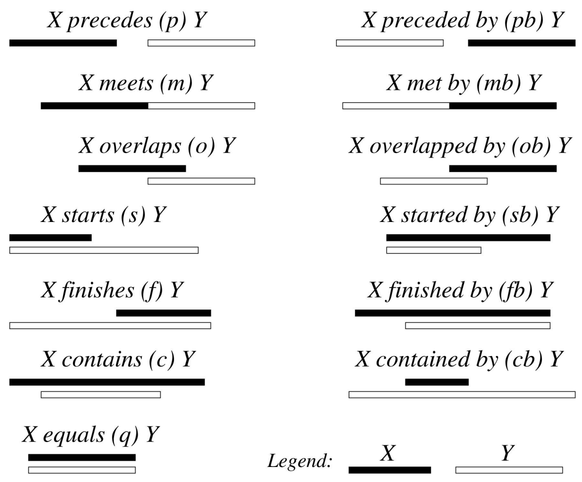

Fig. 2. The 13 relations $\Re$ between intervals [2], [9].

3 Related Work

To our knowledge, the existing approaches on event aggregation (e.g., [10], [15], [17], [20], [22]), cannot detect distributed predicates that are defined on the relative timing occurrences of intervals of interest across different processes. The problems *Predicate_Rel* and *Predicate_Rel** [3] were defined earlier to detect predicates in distributed executions, using *logical time-based* [14] causality relationships defined in [12], [13]. As distributed executions encounter uncertain message propagation times and uncertain CPU scheduling delays, different executions of the same distributed program lead to different interleavings of the events. To analyze the causality relationships among the events across different processes, vector clock-based[1] logical time [6] was necessary [3] to solve *Predicate_Rel* and *Predicate_Rel**. In contrast, the objective of this paper is to capture relative time dependencies among intervals at different processes. Hence, vector clocks are redundant. To appreciate the differences, consider these architectures:

- *Computer network.* Each location L_i $(1 \leq i \leq n)$ has a device/processor and a control program that monitors variables that are defined and used within the code. The activity of the device/processor is modeled as the activity

1. Each process stores a vector VT of size n. The jth entry of the vector at process i tracks the local time of j as known to i. Vector clocks have the property that, for events e and f, $VT(e) < VT(f)$ if and only if $e \longrightarrow f$, where $\longrightarrow$ denotes Lamport's causality relation [14].

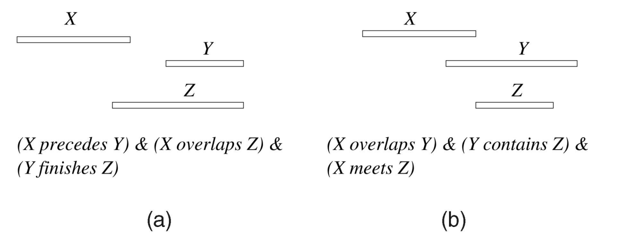
X
Y
Z
(X precedes Y) & (X overlaps Z) & (Y finishes Z)
(a)
X
Y
Z
(X overlaps Y) & (Y contains Z) & (X meets Z)
(b)

Fig. 3. Example problem specifications. The intervals X_i, Y_j, and Z_k are at different processes.

of a process P_i. A timing diagram showing the intervals at each process P_i, when ϕ_i is true, is given in Fig. 1b.

Causal relations among the events can be captured because the monitored variables used in the relations are internal to the process/device. Relations based on relative or absolute timing occurrences can also be detected if (physical) clocks are synchronized.

- *Sensor network.* Sensors $S_1, S_2, \ldots S_n$ at locations $L_1, L_2, \ldots L_n$, respectively, sense physical parameters of interest (e.g., temperature, pressure, humidity, and chemical concentrations) at those locations. The sensor S_i models the activity of location L_i as a process P_i, appropriately timestamping relevant events within the process. A timing diagram for four sensors S_1-S_4, shown in Fig. 1c, is given in Fig. 1d.

  Causality relations among the events in the environment at $L_1 \ldots L_n$ cannot be captured because there is no mechanism for the processes $P_1 \ldots P_n$ in the sensors S_1 to S_n to model or capture causal relations in the environment external to themselves. Thus, an algorithm based on logical vector time cannot *sense* causality-based predicates in the environment, external to the computer network, because the communication within the environment has no mechanism to represent causal relations [3].

For both of the architectures, we assume a loosely coupled ad hoc asynchronous message-passing system in which any two processes belonging to the process set $N = \{P_1, P_2, \ldots, P_n\}$ can communicate over logical channels. Each process sends its gathered data eventually and asynchronously (via any routes) in a FIFO stream to a data fusion server P_0 [10], [15], [20] (see Fig. 1a). Note that the network may be wired or wireless. The following gives the differences from [3]:

1. In [3], *causality-based* predicates were specified. The solution to $Predicate\_Rel$ and $Predicate\_Rel^*$ used vector time (i.e., no single time axis) for the *computer network* architecture. Scalar (logical or physical) time, even with synchronized clocks, is inadequate because it cannot capture causal dependencies in the distributed program.

   In this paper, we adapt the approach of [3] to solve $Predicate\_Rel$ and $Predicate\_Rel^*$ for predicates specified using *relative timing constraints* (i.e., not causality constraints) on a *single time scale* for the *computer network architecture*.
2. Our solution also solves $Predicate\_Rel$ and $Predicate\_Rel^*$ in the *sensor network* architecture when predicates are specified using *relative timing constraints* (i.e., not causality constraints) on a *single time scale*. Note that the use of logical vector clocks does not help to solve $Predicate\_Rel$ and $Predicate\_Rel^*$ in the *sensor network* architecture when predicates are specified using causality relationships.
3. In [3], the set $\Re$ had 40 orthogonal causality-based relations. For the predicates specified using relative timing constraints on a single time axis, this paper considers $\Re$ containing 13 orthogonal relations. The structure of the algorithm is similar to [3]. However, there is no known mapping between the 40 orthogonal relations of [3] and the 13 orthogonal relations used here. Hence, the details have to be worked out from scratch.

The relationship of our results to the results of [3] is summarized in Table 1. In addition, this paper formulates and solves problem $Simultaneous$.

4 Processing at the Sensors/Processes

An interval at P_i begins when the local predicate ϕ_i becomes true and ends when ϕ_i becomes false. We assume the physical clock has infinitely fine granularity so each (event-triggered) state transition at a process occurs at a distinct tick (*local discreteness*). There are two consequences of *local discreteness* and the model for intervals. 1) An interval has a nonzero duration, implying that *point intervals* are not allowed. 2) An interval can begin at P_i only after the previous interval at P_i ends (see Fig. 1b)—termed the *local interval separation* property.

Processes $P_1, P_2, \ldots, P_n$ track the start and end timestamps of their local intervals, using the synchronized clocks. t_i^- and t_i^+ denote the timestamps at process P_i at the start and at the end of an interval, respectively. Each process P_i $(1 \leq i \leq n)$ maintains the data structure Log_i, shown in Fig. 4. Log_i is constructed and sent to P_0 asynchronously over a logical FIFO channel when an interval completes. P_0 then uses the *Log*s reported to determine the relationship between interval pairs (see Section 5). The maximum number of intervals at any process is assumed to be p.

Complexity. Each Log at a process P_i, $1 \leq i \leq n$, takes two integers space. As one log message is sent per interval, the number of messages is p for each P_i $(i \neq 0)$. This gives a total number of messages as np. The total message space overhead for any process is $2p$. Hence, the total message space complexity is $2np$.

5 Algorithm $Predicate\_Rel$

The central data fusion server P_0 maintains n queues, $Q_1, Q_2, \ldots, Q_n$ for *Log*s from each of the processes. The server runs algorithm $Predicate\_Rel$ [3] to process the interval information it receives in the queues. The algorithm detects a set of intervals, one on each process, such that each pair of intervals satisfies the relationship specified for that pair of processes. If no such set of intervals exists, the algorithm does not return any interval set. If there exists an interval record at the head of each queue and these interval records cannot be pruned,[2] then these intervals satisfy $r_{i,j}\ \forall\, i, j$, where $i \neq j$ and $1 \leq i, j \leq n$. Hence, these intervals form a solution set.

Assume that interval X occurs at P_i and interval Y occurs at P_j. For any two intervals X and X' that occur at the same process, if $precedes(X, X')$, then X is a *predecessor* of X' and X' is a *successor* of X. We use variants of the *prohibition* function $\mathcal{H}(r_{i,j})$, the *allows* relation $\mapsto$, and the lemmas given in [3] to show the design of the algorithm. Although the algorithm $Predicate\_Rel$ is the same [3], there is no known mapping of the 40 relations in [3] to the 13 relations used here. Hence, the results for the different set $\Re$ need to be developed independently from scratch—the domain and codomain of $\mathcal{H}$, as well as the set $\Re$, using which *allows* is defined, are different. Thus, the new prohibition function and allows relation are devised from scratch.

2. Henceforth, a reference to an interval will be to the record of the interval, depending on context.

TABLE 1
Objectives of This Paper Compared with Related Problem Specifications

Network architecture	Detect predicates on the relative occurrence of intervals using synchronized (scalar) clocks	Detect predicates on the relative occurrence of intervals, based on causality, captured by logical vector time
Computer network	this paper	solved in [3]
Sensor network	this paper	not solved

Definition 2. *Prohibition function $\mathcal{H} : \Re \rightarrow 2^{\Re}$ is defined to be*

$$\mathcal{H}(r_{i,j}) = \{R \in \Re \mid if\ R(X,Y)\ is\ true\ then\ r_{i,j}(X,Y')\ is\ false\ for\ all\ Y'\ that\ succeed\ Y\}.$$

Definition 3. *The "allows" relation $\mapsto$ on $\Re \times \Re$ is such that $R' \mapsto R''$ if the following holds: If $R'(X,Y)$ is true, then $R''(X,Y')$ can be true for some Y' that succeeds Y.*

Examples (refer to Fig. 2). Assume Y' succeeds Y.

1. $c \mapsto o$ because, if $c(X,Y)$ is true, then $o(X,Y')$ may also be true.
2. $m^{-1} \mapsto f^{-1}$ because, if $m^{-1}(X,Y)$ is true, then $fb(X,Y')$ may also be true.

Lemma 1. *If $R \in \mathcal{H}(r_{i,j})$, then $R \not\mapsto r_{i,j}$ else if $R \notin \mathcal{H}(r_{i,j})$, then $R \mapsto r_{i,j}$.*

Table 2 gives $\mathcal{H}(r_{i,j})$ for the 13 relations in $\Re$. It is constructed by analyzing each relation pair.

Example. The third row of Table 2 gives $\mathcal{H}(r_{i,j})$ for the relations o and ob.

- In column two, $\mathcal{H}(o_{i,j}(X_i,Y_j)) = \{p,m,o,s,f,fb,cb,q\}$. Hence, $p(X_i,Y_j)$ or $m(X_i,Y_j)$ or $o(X_i,Y_j)$ or $s(X_i,Y_j)$ or $f(X_i,Y_j)$ or $fb(X_i,Y_j)$ or $cb(X_i,Y_j)$ or $q(X_i,Y_j)$ implies that $o(X_i,Y'_j)$ can never hold for any successor Y'_j of Y_j.
- In column three,

  $$\mathcal{H}(ob_{j,i}(Y_j,X_i)) = \{p,m,mb,o,ob,s,sb,f,fb,c,cb,q\}.$$

  Hence, $p(Y_j,X_i)$ or $m(Y_j,X_i)$ or $mb(Y_j,X_i)$ or $o(Y_j,X_i)$ or $ob(Y_j,X_i)$ or $s(Y_j,X_i)$ or $sb(Y_j,X_i)$ or $f(Y_j,X_i)$ or $fb(Y_j,X_i)$ or $c(Y_j,X_i)$ or $cb(Y_j,X_i)$ or $q(Y_j,X_i)$ implies that $ob(Y_j,X'_i)$ can never hold for any successor X'_i of X_i.

As per Theorem 1, if R' allows R'', R'^{-1} does not allow R''^{-1}.

Theorem 1. *For $R', R'' \in \Re$ and $R' \neq R''$, if $R' \mapsto R''$, then $R'^{-1} \not\mapsto R''^{-1}$.*

The theorem can be observed to be true from Lemma 1 and Table 2 by using a case-by-case analysis. Table 3 shows the grid of the $\mapsto$ relation for this analysis. A "1" indicates that the relation in the row header allows the relation in the column header. The relation is transitive and the transitive dependencies are shown in Fig. 5. Alternately, this analysis is easier by using the following form of Theorem 1: "For $R' \neq R''$, if $R' \notin \mathcal{H}(R'')$, then $R'^{-1} \in \mathcal{H}(R''^{-1})$."

```
type Log = record          Start of an interval:
    start : integer;           $Log_i.start = t_i^-$.
    end: integer;          End of an interval:
end                            $Log_i.end = t_i^+$;
                               Send $Log_i$ to central process $P_0$.
```

Fig. 4. Data structure and operations to construct Log at P_i $(1 \le i \le n)$.

Examples.

1. $c \mapsto o \Rightarrow c^{-1} \not\mapsto o^{-1}$, which can be seen to be true.
2. $m^{-1} \mapsto f^{-1} \Rightarrow m \not\mapsto f$, which can be seen to be true.

Note $R' \neq R''$ in Theorem 1; otherwise, $R' \mapsto R'$ holds as for p, pb, and c, leading to $R'^{-1} \not\mapsto R'^{-1}$, a contradiction.

Lemma 2. *If the relationship $R(X,Y)$ between intervals X at P_i and Y at P_j is contained in the set $\mathcal{H}(r_{i,j})$ and $R \neq r_{i,j}$, then X can be removed from the queue Q_i.*

Lemma 3. *If the relationship between a pair of intervals X at P_i and Y at P_j is not equal to $r_{i,j}$, then either X or Y is removed from the queue.*

Lemma 3 guarantees progress; when two intervals are checked, if the desired relationship is not satisfied, at least one of them can be discarded.

Example. We want to detect X and Y, where $r_{i,j}(X,Y) = fb$. If $R(X,Y) = o$, we have that $o \not\mapsto fb$; hence, $o(X,Y)$ will not allow $fb(X,Y')$ to be true for any Y'. Hence, the record of X must be deleted. Further, $ob \not\mapsto f$ and, hence, $ob(Y,X)$ will not allow $f(Y,X')$ to be true for any X'. Hence, the record of Y must also be deleted.

Theorem 2. *Problem Predicate_Rel is solved by the algorithm in Fig. 6.*

Theorem 3. *The algorithm in Fig. 6 has the following complexities:*

1. *The total message space complexity is $2np$ (proven in Section 4).*
2. *The total space complexity at process P_0 is $2np$ (follows from (1)).*
3. *The time complexity at P_0 is $O(pn^2)$.*

Proof. The time complexity is the product of the number of steps needed to determine a relationship ($O(1)$ follows trivially from Fig. 2) and the number of relations determined. For each interval considered from one of the queues in $updatedQs$ (lines 6-12), the number of relations determined is $n-1$. Thus, the number of relations determined for each iteration of the **while** loop is $(n-1)|updatedQs|$. But, $\sum |updatedQs|$ over all iterations of the **while** loop is less than the total number of intervals over all of the queues. Thus, the total number of relations determined is less than $(n-1) \cdot x$, where $x = pn$ is the upper bound on the total number of intervals over all the queues. As the time required to determine a relationship is $O(1)$, the time complexity is $O(n^2p)$. □

The performance of the algorithm is given in Table 4.

6 A More General Class of Timing Predicates

In $Predicate\_Rel^*$, for each pair of processes (P_i, P_j), there is a set $r^*_{i,j} \subseteq \Re$ such that some relation in $r^*_{i,j}$ must hold. To solve $Predicate\_Rel^*$, given an *arbitrary* $r^*_{i,j}$, a solution based on algorithm $Predicate\_Rel$ will not work because, in the crucial tests in lines 13-14, neither interval may be removable and yet none of the relations from $r^*_{i,j}$ might hold between the two intervals.

TABLE 2
Prohibition Functions $\mathcal{H}(r_{i,j})$ for the 13 Relations $r_{i,j}$ in $\Re$

Relation r	$\mathcal{H}(r_{i,j}(X_i, Y_j))$	$\mathcal{H}(r_{j,i}(Y_j, X_i))$
$p = pb^{-1}$	$\emptyset$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$m = mb^{-1}$	$\{p, m, o, s, f, fb, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$o = ob^{-1}$	$\{p, m, o, s, f, fb, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$s = sb^{-1}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$f = fb^{-1}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$	$\{p, m, o, s, f, fb, cb, q\}$
$c = cb^{-1}$	$\{p, m, o, s, f, fb, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$
$q = q^{-1}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$	$\{p, m, mb, o, ob, s, sb, f, fb, c, cb, q\}$

p is "precedes," m is "meets," o is "overlaps," s is "starts," f is "finishes," c is "contains," and q is "equals."

Example. Let $r^*_{i,j} = \{pb, o\}$ and let the intervals under consideration at P_i and P_j be X and Y such that $c(X, Y)$ and, hence, $cb(Y, X)$ holds. This is illustrated in Fig. 7.

- $c \in \mathcal{H}(pb), cb \notin \mathcal{H}(p)$. Even though X will not form a part of a solution satisfying relation pb with any future Y', Y can form a solution satisfying relation p with any future X', i.e., $p(Y, X')$ may be true. So, the record of Y must be retained in Q_j.
- $c \notin \mathcal{H}(o), cb \in \mathcal{H}(ob)$. Even though Y will not form a part of a solution satisfying relations ob with any future X', X can form a solution satisfying relation o with any future Y', i.e., $o(X, Y')$ may be true. So, the record of X must be retained in Q_i.

The records of neither X nor Y can be deleted from Q_i or Q_j, respectively.

To avoid tracking multiple intervals in each queue and examining the exponential number of global states (up to p^n), we assume the CONVEXITY property [3] on $r^*_{i,j}$.

Definition 4.

CONVEXITY :

$$\forall R \notin r^*_{i,j} : \Big(\forall r_{i,j} \in r^*_{i,j}, R \in \mathcal{H}(r_{i,j}) \bigvee \forall r_{j,i} \in r^*_{j,i}, R^{-1} \in \mathcal{H}(r_{j,i})\Big).$$

By this property, there is no relation R outside $r^*_{i,j}$ such that, for any $r1, r2 \in r^*_{i,j}$, $R \mapsto r1$ and $R^{-1} \mapsto r2^{-1}$. It guarantees that either X or Y or both get deleted if $R(X, Y) \notin r^*_{i,j}$ and, hence, there is no explosion of global states. The following results [3] are used.

Lemma 4. *If the relation $R(X, Y)$ between intervals X and Y (at processes P_i and P_j, respectively) is contained in the set $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$, then interval X can be removed from the queue Q_i.*

Lemma 5. *If the relation $R(X, Y)$ between a pair of intervals X and Y (at processes P_i and P_j, respectively) does not belong to the set $r^*_{i,j}$, where $r^*_{i,j}$ satisfies* CONVEXITY, *then either interval X or interval Y is removed from the queue.*

TABLE 3
The "allows" Relation $\mapsto$ on $\Re \times \Re$, in Matrix Form, to Verify Theorem 1

$\mapsto$	p	pb	m	mb	o	ob	s	sb	f	fb	c	cb	q
p	1												
pb	1	1	1	1	1	1	1	1	1	1	1	1	1
m	1												
mb	1		1		1					1	1		
o	1												
ob	1		1		1					1	1		
s	1												
sb	1		1		1					1	1		
f	1												
fb	1												
c	1		1		1					1	1		
cb	1												
q	1												

Theorem 4. *If the set $r^*_{i,j}$ satisfies* CONVEXITY, *then Problem $Predicate\_Rel^*$ is solved by replacing lines 13 and 15 in algorithm $Predicate\_Rel$ in Fig. 6 by lines 13 and 15 in Fig. 8.*

The only changes to Algorithm $Predicate\_Rel$ are in lines 13 and 15. In Algorithm $Predicate\_Rel^*$, $R(X, Y)$ is checked for membership in $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ in line 13, instead of membership in $\mathcal{H}(r_{i,j})$. Both $\mathcal{H}(r_{i,j})$ and $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ are sets of size between 0 and $|\Re| = 13$. A similar observation holds for the change on line 15. This gives Corollary 1.

Corollary 1. *The time, space, and message complexities of Algorithm $Predicate\_Rel^*$ are the same as those of Algorithm $Predicate\_Rel$, as stated in Theorem 3.*

7 Application: Detecting Simultaneous Events

Consider Problem $Simultaneous$, introduced in Section 2. Let $\mathcal{I} = \{I_1, I_2, \ldots I_n\}$ be a set containing one interval I_i from each process P_i such that the local predicate ϕ_i is true in I_i. As the algorithm for $Predicate\_Rel^*$ examines a pair of intervals at a time, we show the following:

Theorem 5. *There is a common instant in all of the intervals in $\mathcal{I} = \{I_1, I_2, \ldots I_n\}$ if and only if, for each pair of intervals I_i and I_j in $\mathcal{I}$, there is a common instant.*

Proof sketch. If there is a common instant in all the intervals in $\mathcal{I}$, trivially, for each pair of intervals I_i and I_j in $\mathcal{I}$, there is a common instant. Showing the theorem in the opposite direction is not as trivial. This result is a variant of the result in [13] and is hence not proved formally here. Define $I_{k_1,\ldots k_m}$ as the contiguous interval $\bigcap_{i=k_1,\ldots k_m} I_i$. Clearly, this is true for $m = 2$, which can serve as the base case for an induction-based proof. Now, the intersection $I_{k_1,\ldots k_m} \cap I_{k_{m+1}}$ must be a contiguous nonempty interval by the following logic:

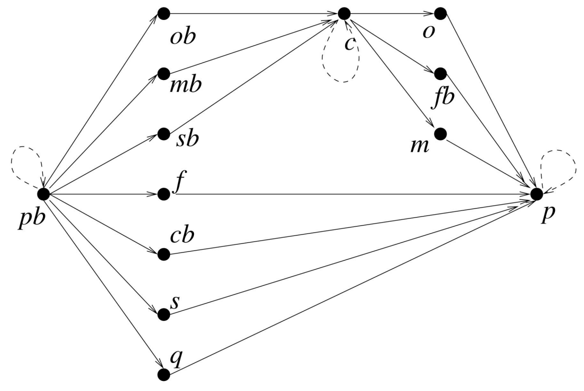

Fig. 5. The transitive relation "allows," $\mapsto$.

```
queue of Log: $Q_1, Q_2, \ldots Q_n = \perp$
set of int: $updatedQs$, $newUpdatedQs = \{\}$
On receiving interval from process $P_z$ at $P_0$
1:  Enqueue the interval onto queue $Q_z$
2:  if (number of intervals on $Q_z$ is 1) then
3:      $updatedQs = \{z\}$
4:      while ($updatedQs$ is not empty)
5:          $newUpdatedQs$=\{\}
6:          for each $i \in updatedQs$
7:              if ($Q_i$ is non-empty) then
8:                  $X$ = head of $Q_i$
9:                  for $j = 1$ to $n$
10:                     if ($Q_j$ is non-empty) then
11:                         $Y$ = head of $Q_j$
12:                         Test for $R(X,Y)$ using interval timestamps (Fig. 2)
13:                         if ($R(X,Y) \in \mathcal{H}(r_{i,j})$) and $R \neq r_{i,j}$ then
14:                             $newUpdatedQs = \{i\} \cup newUpdatedQs$
15:                         if ($R(Y,X) \in \mathcal{H}(r_{j,i})$) and $R \neq r_{j,i}$ then
16:                             $newUpdatedQs = \{j\} \cup newUpdatedQs$
17:         Delete heads of all $Q_k$ where $k \in newUpdatedQs$
18:         $updatedQs = newUpdatedQs$
19:     if (all queues are non-empty) then
20:         Heads of queues identify intervals that form the solution.
```

Fig. 6. Online algorithm at P_0 to solve $Predicate\_Rel$, based on [3].

TABLE 4
Complexities of $Predicate\_Rel$, $Predicate\_Rel^*$, and $Simultaneous$

Time complexity at server P_0	Total number of messages	Space at server P_0 (=total message space)	Space at P_i, $i \in [1,n]$ (in integers)
$O(n^2p)$	np	$2np$	2

- If $I_{k_1,\ldots k_m} \cap I_{k_{m+1}}$ were empty (see Fig. 9), then $I_{k_{m+1}}$ and $I_{k_1,\ldots k_m}$ are separated.
- Hence, $I_{k_{m+1}}$ cannot intersect with at least one interval among $I_{k_1}, I_{k_2}, \ldots I_{k_m}$.
- This leads to a contradiction because, for each pair of intervals I_i and I_j in $\mathcal{I}$, there is a common instant. □

Hence, from Theorem 5, to detect a common instant across all processes, we can examine a pair of intervals at a time for overlap. For events at P_i and P_j to occur simultaneously, intervals X and Y must have some common instants. To apply the algorithms presented in the previous section, we need to identify an appropriate $r^*_{i,j}$ and verify that it satisfies CONVEXITY. We proceed as follows:

1. Observe from Fig. 2 that, to satisfy the condition that two intervals overlap with each other,[3] $r^*_{i,j} = \{o, ob, s, sb, f, fb, c, cb, q\}$.
2. For each relation R in $\Re \setminus r^*_{i,j} = \{p, pb, m, mb\}$, observe from Table 2 that $R \in \mathcal{H}(\theta)$ or $R^{-1} \in \mathcal{H}(\theta^{-1})$, where $\theta \in r^*_{i,j}$.
   So, CONVEXITY is satisfied by $r^*_{i,j}$ and algorithm $Predicate\_Rel^*$ can be used.

Then, from Table 2, we have

$$\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j}) = \{p, m, o, s, f, fb, cb, q\}, \tag{1}$$

$$\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j}) \setminus r^*_{i,j} = \{p, m, o, s, f, fb, cb, q\} \setminus \{o, ob, s, sb, f, fb, c, cb, q\} = \{p, m\}. \tag{2}$$

Substituting the values in lines 13 and 15 in Fig. 8 gives the code in Fig. 10, which is what we intuitively expect.

3. We rule out m and mb because of the state transitions of both intervals, one from true to false and the other from false to true, at the meeting instant.

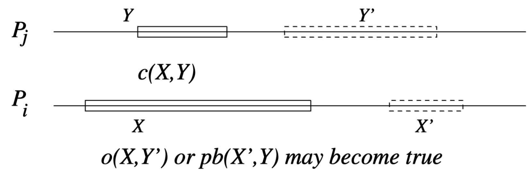

Fig. 7. Illustration of the example of "no queue pruning" when using the algorithm of Fig. 6.

```
13:     if ($R(X,Y) \in \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ and $R \notin r^*_{i,j}$) then
14:         $newUpdatedQs = \{i\} \cup newUpdatedQs$
15:     if ($R(Y,X) \in \bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i})$ and $R \notin r^*_{j,i}$) then
16:         $newUpdatedQs = \{j\} \cup newUpdatedQs$
```

Fig. 8. Algorithm $Predicate\_Rel^*$: Changes to $Predicate\_Rel$ when $r^*_{i,j}$ satisfies CONVEXITY.

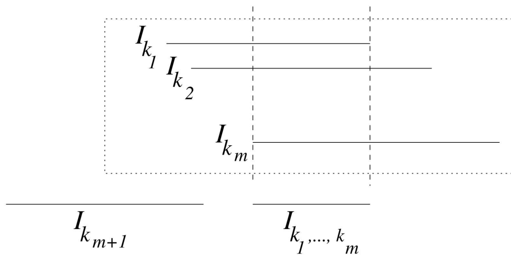

Fig. 9. The condition for intersection of intervals in a solution for simultaneous occurrence when examining intervals pairwise.

```
13:     if ($R(X,Y) \in \{p, m\}$) then
14:         $newUpdatedQs = \{i\} \cup newUpdatedQs$
15:     if ($R(Y,X) \in \{p, m\}$) then
16:         $newUpdatedQs = \{j\} \cup newUpdatedQs$
```

Fig. 10. Algorithm $Predicate\_Rel^*$ to detect simultaneous events: Changes to algorithm $Predicate\_Rel$ are listed.

8 Discussion

This paper presented an algorithm based on [3] to detect a global predicate specified using timing relationships across the various locations, using a single time axis. The proposed algorithm has low overhead (see Table 4) at the sensor nodes. The data fusion load is at the back-end server P_0, which can be a dedicated powerful processor. The paper also considered a special case of the algorithm wherein the simultaneous occurrence of events across the system can be detected. The space complexity of the algorithms for $Predicate\_Rel$, $Predicate\_Rel^*$, and $Simultaneous$ is conjectured to be optimal because each of the p intervals at each of the n processes needs to be transmitted and then stored in the queues at P_0. Similarly, the time complexity ($O(n^2p)$) is conjectured to be optimal because each of the p intervals at each of the n processes needs to be tested with $n-1$ other intervals.

We mention some limitations of the approach. 1) Global time is at best an approximation. Current synchronization techniques achieve precision of the order of microseconds (see survey [24]). However, this research assumes global time as an axiom. We do not consider the bounds on clock drift and skew as these are technology-dependent and will likely change as better clock synchronization mechanisms are discovered and implemented. 2) Even with synchronized clocks, the common time axis is not useful if predicates based on the causality relation [14] are specified. This is because causality-based relations cannot be determined based on physical time relationships, but require logical vector clocks. The algorithms in [3] can be used instead. 3) The availability of global time in some scenarios with limited resources and/or constrained network topologies may not be practical.

This research considered only conjunctive predicates, whereby each process can locally determine the start and end of the local intervals of interest. Hence, the global predicate was a conjunct of such local predicates. The key here was that any interval was involved in at most $n-1$ comparisons with the intervals from the heads of other queues, before some interval gets deleted—see the proof of Theorem 3. More general predicates, such as $x_i + y_j + z_k = 40$, termed *relational predicates*, incur exponential overhead in terms of time complexity at P_0 and, hence, are not considered here. The exponential cost arises because, if a pairwise comparison fails between a pair of intervals at queue heads, they may still form part of a solution and, hence, cannot be deleted from the queues.

Although relational predicates cannot be detected in polynomial time, the following types of queries under the "Simultaneous occurrence" relationship can be solved. Queries such as $x_i = y_j = z_k$ can be solved with the same overhead as for $Predicate\_Rel$. Each interval is involved in at most $n-1$ comparisons, after which at least one interval is deleted if the solution is not found. (Refer to the proof of Theorem 3.) This is due to the monotonicity property of time—the interval that finishes earliest can be deleted if a solution is not found. Detecting such predicates allows one to determine whether nearby sensors are simultaneously sensing the same/similar values, such as of temperatures.

For range predicates where the range is known to the sensors a priori, each reported interval is the duration in which the sensed values are within the range. If the range is not known a priori, then every single change in the sensed variable needs to be reported. This is not efficient.

Note that negation is easily supported. The global predicate should be in Conjunctive Normal Form, where each conjunct is local and may be a negation. Complex predicates such as $\phi =$ "tigers drink before deer and no birds drink in-between" can also be handled. Let T, B, and D denote the intervals in which tigers, birds, and deer drink. Then, ϕ is detected when $T\ m|o|s|cb\ \overline{B}$ AND $\overline{B}\ m|o|fb|c\ D$ AND $D\ pb\ T$. This assumes that there are several intervals on each process line and all of the intervals and their negated intervals eventually finish [18]. Further, variable latency which leads to messages from different nodes not arriving temporally ordered at P_0 is not a problem. There is a separate queue Q_i at P_0 for the event stream from each sensor S_i.

In sensor networks, a process P_i may batch the transmission of a sequence of messages for energy conservation, resulting in high latency. As each message sent to P_0 has the start and end timestamp of the corresponding interval (see the data structure of Fig. 4), the algorithm is immune to such latencies and jitter.

If a sensor can sense multiple parameters (e.g., MICAz Crossbow can sense temperature and light), then each parameter is abstracted as a different process with its own timeline. So, in Fig. 1c and Fig. 1d, as many different processes as there are parameters that are sensed would run at the location of such a sensor. If sensor S_4 were a MICAz mote, there would be two process timelines, P_4' and P_4'', instead of P_4, each with their intervals that get projected separately on the time line L_4.

The paper assumed reliable FIFO channels from each P_i to P_0. If a message gets lost and if the interval corresponding to the lost message was part of a solution, then that solution would go undetected. That is the price to pay if unreliable delivery is assumed. There are well-understood mechanisms (such as using sequence numbers and ACKs) to impose reliable delivery over unreliable channels. Whether the cost of reliable delivery is acceptable depends on the application. For messages from P_i received out of order, the use of sequence numbers can be used to present a sequential order of the intervals in the queue Q_i at P_0.

The presented formalism assumed *local discreteness*, which implied that the *local interval separation* property holds and that no *point intervals* were allowed. Variations can be handled by adapting this formalism. For example, if *local interval separation* is relaxed, an interval can begin at the same instant at which the previous interval at the same process ends. X_i' would be a successor of X_i if $m(X_i, X_i')$ or $p(X_i, X_i')$. In Table 2, $\mathcal{H}(m)$ would exclude f, fb, q, and $\mathcal{H}(s)$, $\mathcal{H}(sb)$, $\mathcal{H}(q)$ would each exclude mb. In Table 3, for $\mapsto$, there would be "1" for (f,m), (fb,m), (q,m), (mb,s), (mb,sb), and (mb,q). Theorem 1 can be seen to still hold. Other variations, such as allowing *points*(one point per clock tick) and about clock properties and time density, can be similarly handled.

ACKNOWLEDGMENTS

Some portions of this paper appear in [11].

REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramanian, E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks,* vol. 38, no. 4, pp. 393-422, 2002.

[2] J. Allen, "Maintaining Knowledge about Temporal Intervals," *Comm. ACM,* vol. 26, no. 11, pp. 832-843, 1983.

[3] P. Chandra and A.D. Kshemkalyani, "Causality-Based Predicate Detection Across Space and Time," *IEEE Trans. Computers,* vol. 54, no. 11, pp. 1438-1453, Nov. 2005.

[4] K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. Computer Systems,* vol. 3, no. 1, pp. 63-75, 1985.

[5] J. Elson and K. Romer, "Wireless Sensor Networks: A New Regime for Time Synchronization," *Proc. First Workshop Hot Topics in Networks (HotNets-I),* Oct. 2002.

[6] C. Fidge, "Logical Time in Distributed Computing Systems," *Computer,* vol. 24, no. 8, pp. 28-33, Aug. 1991.

[7] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-Sync Protocol for Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems,* pp. 138-149, Nov. 2003.

[8] S. Ganeriwal, D. Ganesan, H. Sim, V. Tsiatsis, M. Hansen, and M. Srivastava, "Estimating Clock Uncertainty for Efficient Duty-Cycling in Sensor Networks," *Proc. Third ACM SenSys Conf.,* 2005.

[9] C.L. Hamblin, "Instants and Intervals," *The Study of Time,* pp. 324-332. Springer-Verlag, 1972.

[10] B. Krishnamachari, D. Estrin, and S. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," *Proc. Int'l Conf. Distributed Computing Systems Workshops,* pp. 575-578, 2002.

[11] A.D. Kshemkalyani, "Predicate Detection Using Event Streams in Ubiquitous Environments," *Proc. IFIP Conf. Network-Centric Ubiquitous Systems (NCUS),* pp. 807-816, Dec. 2005.

[12] A.D. Kshemkalyani, "Temporal Interactions of Intervals in Distributed Systems," *J. Computer and System Sciences,* vol. 52, no. 2, pp. 287-298, Apr. 1996.

[13] A.D. Kshemkalyani, "A Fine-Grained Modality Classification for Global Predicates," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 8, pp. 807-816, Aug. 2003.

[14] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM,* vol. 21, no. 7, pp. 558-565, July 1978.

[15] S. Li, Y. Lin, S. Son, J. Stankovic, and Y. Wei, "Event Detection Services Using Data Service Middleware in Distributed Sensor Networks," *Telecomm. Systems,* vol. 26, nos. 2-4, pp. 351-368, 2004.

[16] T. Logsdon, *The Navstar Global Positioning System.* Van Nostrand/Reinhold, 1992.

[17] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks," *Proc. Symp. Operating Systems Design and Implementation (OSDI),* 2002.

[18] F. Mattern and R. Schwarz, "Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail," *Distributed Computing,* vol. 7, no. 3, pp. 149-174, 1994.

[19] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Trans. Comm.,* vol. 39, no. 10, pp. 1482-1493, Oct. 1991.

[20] P. Pietzuch, B. Shand, and J. Bacon, "Composite Event Detection as a Generic Middleware Extension," *IEEE Network,* vol. 18, no. 1, pp. 44-55, Jan./Feb. 2004.

[21] K. Romer, "Time Synchronization in Ad-Hoc Networks," *Proc. ACM MobiHoc,* 2001.

[22] K. Romer and F. Mattern, "Event-Based Systems for Detecting Real-World States with Sensor Networks: A Critical Analysis," *Proc. DEST Workshop Signal Processing in Wireless Sensor Networks, Int'l Conf. Intelligent Sensors, Sensor Networks, and Information Processing,* pp. 389-395, Dec. 2004.

[23] W. Su and I. Akyildiz, "Time-Diffusion Synchronization Protocol for Sensor Networks," *IEEE/ACM Trans. Networking,* vol. 13, no. 2, pp. 384-397, 2005.

[24] B. Sundararaman, U. Buy, and A.D. Kshemkalyani, "Clock Synchronization for Wireless Sensor Networks: A Survey," *Ad-Hoc Networks,* vol. 3, no. 3, pp. 281-323, May 2005.

[25] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman, "A Taxonomy of Wireless Micro-Sensor Models," *ACM Mobile Computing & Comm. Rev.,* vol. 6, no. 2, Apr. 2002.