J. Parallel Distrib. Comput. 72 (2012) 219–230

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Immediate detection of predicates in pervasive environments

Ajay D. Kshemkalyani

University of Illinois at Chicago, Chicago, IL 60607, USA

ARTICLE INFO

Article history:
Received 31 March 2011
Received in revised form
19 September 2011
Accepted 20 September 2011
Available online 29 September 2011

Keywords:
Sensor networks
Predicate detection
Pervasive computing
On-line algorithms

ABSTRACT

An important problem in pervasive environments is detecting predicates on sensed variables in an asynchronous distributed setting to determine context and to respond. We do not assume the availability of synchronized physical clocks because they may not be available or may be too expensive for predicate detection in such environments with a (relatively) low event occurrence rate. We address the problem of detecting *each* occurrence of a global predicate, at the *earliest* possible instant, by proposing a suite of three on-line middleware protocols having varying degrees of accuracy. We analyze the degree of accuracy for the proposed protocols. The extent of false negatives and false positives is determined by the run-time message processing latencies.

1. Introduction

A pervasive environment can be modeled as a networked autonomous embedded system that interacts with the physical world through sensors and actuators. Such systems aim to sense–monitor–actuate the physical world. A pervasive application is context-aware in that it can adapt its behavior based on the characteristics of the environment [2–4,10,13–15,27,29,31–33]. A central issue is that of monitoring predicates (or properties) defined on variables of the environment. In the general case, the predicate is on a pattern of events and has two components – a spatial component, and a temporal component on the monitored variables. The temporal component specifies various timing relations, such as those in [5,6,18,28], on the observed values of the variables. In this paper, we consider the "instantaneous" snapshot of the variables, to capture their values at the same instant in physical time in the asynchronous message-passing distributed setting.

The existing literature on predicate detection for pervasive environments, e.g., [2–4,13,14,18,29,31–33], except for [15], assumes that it can take instantaneous snapshots in the system. This is possible with physically synchronized clocks. There is a wide body of literature on providing tight clock synchronization for wireless sensor networks [30]. The use of synchronized clocks has the following limitations. (1) Clock synchronization does not come for free as its costs are incurred by a lower layer. (2) The cost overhead may be unnecessary when dealing with lifeform and object movements, which are typically slow, compared to the order of precision provided by synchronized clocks. (3) Synchronized clocks also impose an inevitable skew ϵ, which leads to imprecision in detecting predicates in physical time. Predicate detection is prone to false negatives and false positives when there are "races". When two or more events occur at different sites and it is not possible for a global observer to ascertain the physical time order in which they occurred, a race takes place. It has been shown that when the overlap period of the local intervals, during which the global predicate is true, is less than 2ϵ, false negatives occur [26]. (4) In very resource-constrained sensor systems or those in remote environments, clock synchronization service may be unavailable. (5) Furthermore, even if one of the many clock synchronization protocols is available, it may be too expensive in terms of energy usage in such systems and environments. (6) The use of physically synchronized clocks imposes cross-layer dependence and hampers portability. (7) Physically synchronized clocks also provide more exposure to privacy concerns and security issues by requiring all users to participate in the network layer synchronization protocol. A user may refuse to participate in clock synchronization. Therefore, a companion paper [20] explored the option of using lightweight middleware protocols, without accessing physically synchronized clock service, to detect global predicates. A drawback of those protocols is that a predicate gets detected after each sensor has sensed one more event, its next, locally. So there may be considerable delay. Immediate detection is desirable for applications that require real-time on-line actuation and raising alarms [10]. Early detection was considered in [15]. Their algorithm detects a conjunctive predicate only after all but one sensors have sensed one more event, their next, locally.

In this paper, we complement the study of [20] by proposing a suite of clock-free algorithms to detect global predicates immediately. We characterize and qualify the error as a function of the message transmission delay in reporting sensed values. We

E-mail address: ajayk@cs.uic.edu.

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.09.004

express the accuracy of our algorithms in terms of a parameter Δ. Define Δ as the upper bound on the asynchronous message transmission delay for a system-wide broadcast. Δ includes the delays for queueing in local outgoing and incoming buffers, retransmission if needed for reliability, propagation, process scheduling and context switching, until the received message is processed. None of the algorithms need to know or use Δ in the code! Actually, the accuracy is determined by the actual message transmission delay Δ_{actual} in any particular race condition, and $\Delta_{actual} \leq \Delta$. Thus, the accuracy of the algorithms is adaptable to the actual operating conditions of the sensor network and can be much better.

The skew that governs the imprecision using physical clocks is of the order of microsecs to millisecs if software protocols are available. (Hardware solutions achieve nanosec skews but are impractical in sensornets). Although Δ, that determines the accuracy of our algorithms, is of the order of hundreds of millisecs to secs in small-scale networks, such as smart offices and smart homes, it may be adequate when the number of processes is low and/or the rate of occurrence of sensed events is comparatively low. This is the case for several environments such as office, home, habitat, wildlife, nature, and structure monitoring. Lifeform and physical object movements are typically much slower than Δ. And in the wild, remote terrain, nature monitoring, events are often rare, compared to Δ. Thus, we may not need the precision (in urban settings or the wild) or be able to afford the associated cost (in the wild) of synchronized physical clocks. Further, clock synchronization may not be usable even in urban environments due to privacy concerns.

Predicate types: Our algorithms can detect conjunctive and relational predicates ϕ [9]. ϕ is *conjunctive* if $\phi = \wedge\phi_i$, where ϕ_i is defined on variables local to a single sensor i. For example, ψ : `number of persons in room` $> 3 \wedge temp > 30C$. When ψ becomes true, it triggers setting the thermostat to $25C$. Here, ψ is a conjunctive predicate. ϕ is *relational* if it is an arbitrary logical expression on system-wide variables. For example, $\chi : x_i + y_j > 20$, where x_i and y_j are the number of people in adjoining rooms i and j, respectively. When χ becomes true, turn on the "Capacity exceeded" sign. Here, χ is a relational predicate. The characterization of the accuracy of detection is the same for conjunctive and relational predicates but the level of accuracy is lower for relational predicates. Relational predicates are harder to detect because it requires examining the state lattice to consider "combinations of states" which can together satisfy the predicate. This requires non-polynomial time, which we explicitly avoid. See also [20].

Note that the relational predicate ϕ is an arbitrary Boolean expression. Thus, a predicate like "for all i in site a, there exists a j in site b, such that i and j satisfy a predefined function f" can also be evaluated by evaluating f for all i in a and for all j in b.

Summary of contributions:

1. We present a suite of three algorithms: a simple clock-free algorithm, an interval-vector based algorithm, and a consensus based algorithm, to *immediately* detect *each new* occurrence of a global predicate on variables tracking sensed physical world properties. We do not use physical clocks because they are not affordable or available or needed in certain pervasive environments.
2. The algorithms examine approximations to the actual states that did occur in the physical world execution, without incurring the costs of building a state lattice. We characterize the extent of the errors in detecting a predicate. This is the first work that provides guarantees for the novel problem of *immediate* and *repeated* predicate detection.
3. In the consensus based algorithm, in addition to the traditional positive and negative outcomes, we introduce a *borderline* bin to catch most of the race conditions. The application can deal with borderline outcomes as it deems appropriate, based on its semantics.

In Section 2, we give the system and execution model. In Section 3, we give three algorithms for approximate snapshots to detect the predicate of interest immediately. The first is a simple clock-free algorithm, the second is based on "interval vectors" which are a form of logical clock, and the third is a consensus based algorithm. Although the first two algorithms are simple, they are able to detect predicates that held at any instant in time, immediately and in a repeated manner. This is unlike the algorithms of [11,12] which detect predicates one time only (see [22]) and after a potentially long delay, under the *Possibly* and *Definitely* modalities. Further, those algorithms can detect predicates on in-network variables only, and not on sensed physical world variables. They can detect only conjunctive predicates and not relational predicates. The consensus based algorithm builds on our first two algorithms. In Section 4, we analyze the performance of the algorithms. In Section 5, we present an application scenario and give a concluding discussion.

2. System and execution model

Sensor–actuator networks and pervasive environments are distributed systems that interact with the physical world in a sense-and-respond manner [16]. The *world plane* consists of the physical world entities and the interactions among them. The *network plane* consists of sensors and actuators and the communication network connecting them. For the network plane, we adapt the standard model of an asynchronous message-passing distributed execution (see [19,20,23]). Each sensor–actuator is modeled as a process $P_i (i \in [1 \cdots n])$; the local execution is a sequence of alternating states and "relevant" events that trigger state transitions. The local execution at P_i is $s_i^0 e_i^1 s_i^1 e_i^2 s_i^2 \cdots$. The global execution is the set of local executions, one per process. Let $\mathcal{P}$ be the set of all processes. An event is a *sensing (observation) event* or an *actuation event* of the world plane by the network plane, or a *message send event* or a *message receive event* in the network plane. Assume a maximum of p sensing events at any process. The communication between any pair of processes is FIFO. Messages (in the network plane) assemble global properties from locally sensed values, and actuate the controlled devices. The messages among the network plane processes are *control* messages.

The happens before relation $\longrightarrow$ on the set of events E is defined as follows [24]. For events $e_1, e_2 \in E$, $e_1 \longrightarrow e_2$ if (i) e_1 and e_2 occur at the same process and e_1 occurs before e_2; or (ii) e_1 is a send event and e_2 is the corresponding receive event; or (iii) there is an event e_3 such that $e_1 \longrightarrow e_3$ and $e_3 \longrightarrow e_2$. $(E, \longrightarrow)$ is a partial order. A global state is a collection of local states, one per process. A consistent global state (CGS) is a global state in which, for every receive event that occurs before a local state, the corresponding send event also occurs before the local state at the sending process [7].

Our algorithms work even if communication is unreliable by way of message loss. Thus, a message may be lost but all delivered messages are correct. A lost message may lead to a wrong inference around the time that the message is lost, but it has no ripple effect on future detection. Our characterization of the accuracy uses a (bounded) Δ which can also include re-transmission attempt latencies.

A *sensing* event occurs whenever a sensed value, whether discrete or continuous, changes significantly. A sensed event is modeled as $e = (i, val, t_s)$ to denote the sensor process, value of the attribute sensed, and the physical time of occurrence. It is a straightforward exercise to deal with multiple attributes per process. For each process-attribute, an interval is represented by a value, start time, and finish time as $I = (val, t_s, t_f)$. The interval is implicitly defined by two consecutive events $(i, val1, t1)$ and $(i, val2, t2)$ for that process-attribute, as: $I = (val = val1, t_s = t1, t_f = t2)$. Our goal is to evaluate a predicate ϕ whenever the global state changes.

Problem. Detect *each* occurrence of a global (conjunctive or relational) predicate ϕ on sensed attribute values of the world plane, that held at some instant on a physical time axis, but without using physically synchronized clocks in the network plane having asynchronous message transmissions. Each detection must occur on-line at the *earliest* possible instant on that same physical time axis.

Predicate detection for distributed deterministic programs was first addressed in [9] for relational predicates and in [11,12] for conjunctive predicates. Drawbacks of [11,12] are manyfold. (1) They cannot do immediate detection but rather need to wait for an additional event at each process to occur before being able to perform detection. (2) They cannot do repeated detection because their algorithms hang after a predicate is once detected (see [22] for a detailed discussion). (3) They detect predicates only in the *Possibly* and *Definitely* modalities and not those that held at an instant in physical time. (4) They cannot detect predicates on sensed physical world values but only on in-network variables, because they rely on passively piggybacking timestamps on in-network messages to advance vector time, and there are no in-network messages in sensornets (refer [20,23] for a detailed discussion). (5) They can detect only conjunctive predicates and cannot detect relational predicates. Predicates detection for pervasive environments was addressed in [15,20]. The problem of repeated detection of a predicate, viz., detection of each new occurrence of the predicate, was formalized and solved for conjunctive predicates in [22].

Due to the inherent asynchrony of the control messages and lack of a global observer, there are many possible observations of the world plane execution. The distributed computing literature has defined a lattice of global states and its sub-lattice of consistent global states (CGSs) for executions of distributed programs with semantic deterministic sends and receives [9]. This lattice has been used to make assertions about ϕ under *all possible runs* of the same deterministic distributed program. Due to variations in local program scheduling and transmission and propagation times, the same program passes through different paths in the state lattice in different executions. The time cost of creating this lattice of CGSs and of detecting a relational predicate ϕ is exponential, $O(p^n)$ [9]. Our problem of sensing the physical world is different in a subtle way. We are also hampered by the lack of a global observer. However, we do not make assertions about "all possible executions of the same deterministic distributed program" of the world plane (see [20]), as the world plane does not admit reruns and the world plane execution has nondeterministic factors like human will and nature; there is only the actual nondeterministic execution to make assertions about. The message-passing of the world plane cannot be captured or mimicked by the network plane because the message-passing of the world plane occurs over what are termed as "covert" channels [16]. It is not known how to track the communication over covert channels. The network plane can only sense events and actuate the world plane interfaces. The control messages of the network plane induce an *artificial (non-semantic)* lattice of CGSs [20]. This lattice is artificial because the receipt of the control messages has no semantic significance, and is solely a function of transmission and propagation latencies and local scheduling. We make approximations to the actual path traced by the physical world execution, without constructing the lattice and incurring that overhead.

Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of intervals, one per process. Intervals in $\mathcal{I}$ overlap in physical time iff $\max_i(I_i.t_s) < \min_i(I_i.t_f)$. For this set of intervals, we define a number:

Definition 1. $overlap(\mathcal{I}) = \min_i(I_i.t_f) - \max_i(I_i.t_s)$

overlap gives the maximum overlap across all intervals in $\mathcal{I}$. *overlap* is useful to characterize the accuracy of our protocols, as the best approximation to physical time. As our model does not use physically synchronized clocks, an event is a pair (i, val); and neither do we know $I.t_s$ and $I.t_f$.

In our algorithms, an event notification is sent (broadcast) at each sensing event, which are the relevant events. These event notifications are control messages. The receive events for these control messages have no significance as far as the world plane is concerned because control messages are unrelated to the semantics of the world plane. Therefore, from the perspective of the world plane, the number of CGSs induced is p^n. From the perspective of the network plane, however, the number of CGSs (induced on the events of the world plane) is given by using a result of Charron-Bost [8]. The result states that for any poset, (i) the number of consistent cuts is given by the number of anti-chains of sizes 1 through n, and (ii) there is a bijective mapping from the set of anti-chains to the set of consistent cuts. Therefore, from the perspective of the network plane, the number of CGSs (induced on the sensing events of the world plane) is given by the sum of the number of anti-chains of sizes 1 through n, over the set of sensing events S of the world plane in the partial order $(E, \longrightarrow)$. Furthermore, for every anti-chain, there is a unique CGS and vice versa. Note that $S \subseteq E$; E contains all the sensing events (atomically with their event notification broadcasts) and the receive events for the event notifications. The event notifications help to build the partial order $\longrightarrow$ and cut down the number of CGSs from p^n, but the event notifications and their corresponding receive events have no semantic significance to the world plane. The receipt of event notifications in our algorithms, corresponding to events in $E \setminus S$, helps to dynamically compose and observe CGSs, without actually incurring the overhead of constructing the state lattice. We classify a CGS corresponding to an anti-chain of size 1 as *elementary*, and a CGS corresponding to an anti-chain of size greater than one as *composite*. This classification is useful in the proof structure.

3. Approximate snapshots

3.1. Simple clock-free algorithm

Algorithm 1 Simple Clock-Free Algorithm: Code at P_i to detect a predicate using event notifications.

int: array *Value_Vector*$[1 \ldots n]$

When event $e = (i, val)$ occurs at P_i:
(1) transmit to sink (or broadcast to $P_j \in \mathcal{P} \setminus \{P_i\}$) event notification (i, val)
(2) (if broadcasting is being used) $Evaluate\_State(i, val)$

On P_i receiving event notification $e = (z, val)$ from P_z:
(1) $Evaluate\_State(z, val)$

$Evaluate\_State(z, val)$ at P_i:
(1) $Value\_Vector[z] \longleftarrow val$
(2) **if** $\phi((\forall j) Value\_Vector[j]) = true$ **then**
(3) observed *Value_Vector* satisfies ϕ
(4) raise alarm/actuate

Algorithm 1 gives a simple clock-free algorithm for evaluating ϕ. Each time a new value is sensed by a sensor, it is transmitted to a sink which is one of the n nodes (or a system-wide broadcasting policy can be used). Whenever *Evaluate_State* is invoked, it is executed atomically with its invocation. A node tracks the latest sensed value reported by P_z in *Value_Vector*$[z]$. For

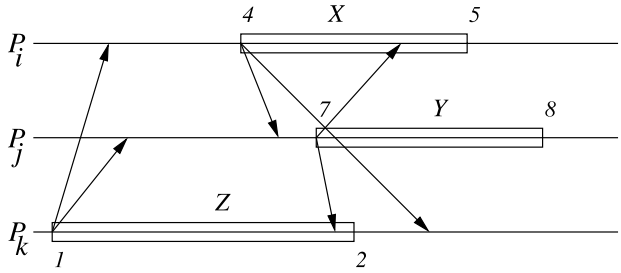

Fig. 1. Overlap, a potential false negative.

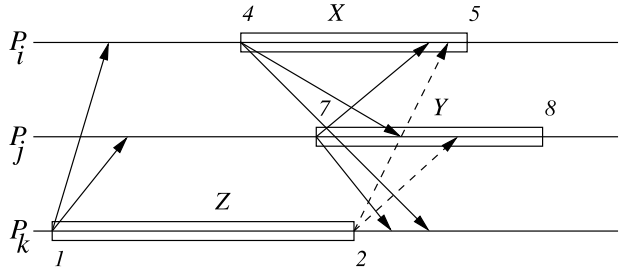

Fig. 2. Overlap, a potential false negative.

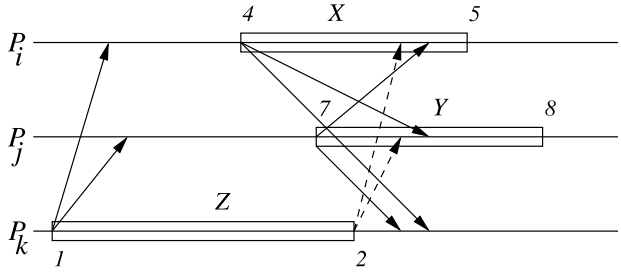

Fig. 3. Overlap, an "inevitable" false negative.

now, assume that a distinguished node (sink) runs *Evaluate_State* to evaluate ϕ. (Note, ϕ can be a relational predicate, like $\sum_{k=1}^{n} Value\_Vector[k] < 20$.) However, if broadcasting is used, all nodes can run *Evaluate_State*; we discuss the implications in Section 3.2. If $overlap \geq \Delta$, then the algorithm can detect that the intervals overlap. Similarly, if $overlap \leq -\Delta$, then the algorithm can detect that the intervals do not overlap.

It is critical to analyze behavior in the face of race conditions, i.e., when $-\Delta < overlap < \Delta$. We explain our results using some examples for this range.

Examples. The examples use timing diagrams and show three intervals X, Y, and Z, at processes P_i, P_j, and P_k, respectively. These intervals are such that ϕ is true over the sensed values in these intervals, and false over other combinations of these and preceding or succeeding intervals. The integer at the start of an interval is the local sequence number of that interval. Messages in regular lines are the notifications sent at the start of X, Y, and Z. Messages in dotted lines are those sent at the start of the next intervals following these – such messages are shown only when they are relevant to the explanation of the example. If event notifications are sent to a sink (instead of being broadcast), the sink could be P_i, P_j, or P_k.

In Fig. 1, P_i and P_j will be able to detect ϕ. However, P_k cannot detect ϕ because by the time it receives the value sensed by P_i, P_k's locally sensed value has changed. We will revisit this example in Section 3.2 to show that P_k will also be able to detect ϕ using "interval vectors" in the next algorithm. In Section 3.3, we revisit

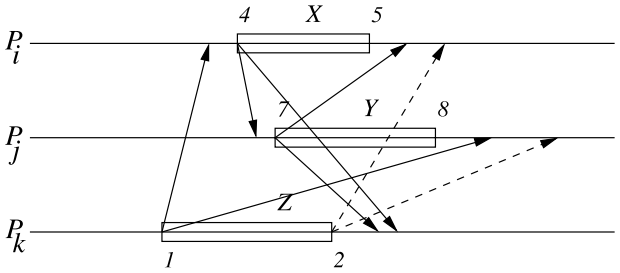

Fig. 4. Overlap, a potential false negative.

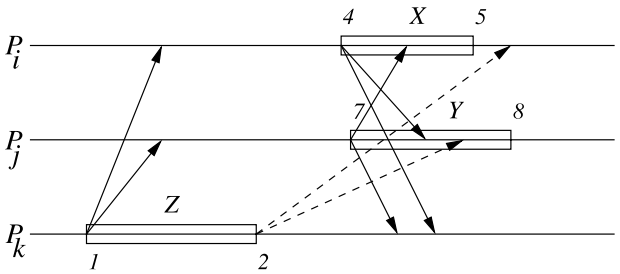

Fig. 5. No overlap, a potential false positive.

this example to show that this detection can be identified as a true positive by the consensus-based algorithm.

In Fig. 2, P_i and P_j, but not P_k, will be able to detect ϕ. However, even with the use of "interval vectors", P_k remains unable to detect ϕ.

In Fig. 3, none of the processes will be able to detect ϕ, even using our next algorithm using "interval vectors". This is an "inevitable" false negative; ϕ cannot be definitively detected in our model, even if it is conjunctive. (Further, even after using lattice evaluation, we can only suspect that this overlap might have occurred [20].)

In Fig. 4, none of the processes will be able to detect ϕ. However, with the use of "interval vectors", we will observe in Section 3.2 that P_j will be able to detect ϕ.

In Fig. 5, P_i and P_j will detect ϕ, resulting in a false positive. This appears inevitable due to the message pattern. However, we will show in Section 3.3 that using consensus, this case can be identified as a *potential* false positive and thus be eliminated.

In Fig. 6, none of the processes detect ϕ, resulting in a true negative. A false positive is not possible.

In Fig. 7, P_j detects a false positive. However, we will see in Section 3.2 that using "interval vectors", this false positive can be eliminated.

Theorem 1. *For a single observer in a system without any synchronized clocks, for the detection algorithm in Algorithm 1, we have for any $\mathcal{I}$ for which ϕ is true:*

1. $overlap \geq \Delta \Longrightarrow \phi$ *is correctly detected*
2. $0 \leq overlap < \Delta \Longrightarrow$ *any outcome is possible*
3. $0 \geq overlap > -\Delta \Longrightarrow$ *any outcome is possible*
4. $overlap \leq -\Delta \Longrightarrow \phi$ *is correctly detected as not holding*

Proof. 1. Consider the duration $[\max_i(I_i.t_s), \max_i(I_i.t_s) + \Delta]$. During this duration, the values $Value\_Vector[z]$, for all z, are received at the sink or observer. Furthermore, these values are not overwritten until after $\max_i(I_i.t_s) + \Delta$. Therefore, in this duration, there is at least one instant when $Value\_Vector[z](\forall z)$ is available, and ϕ can be correctly evaluated over this vector of values.

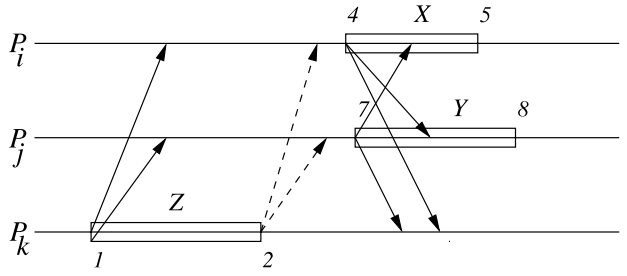

Fig. 6. No overlap, false positive not possible.

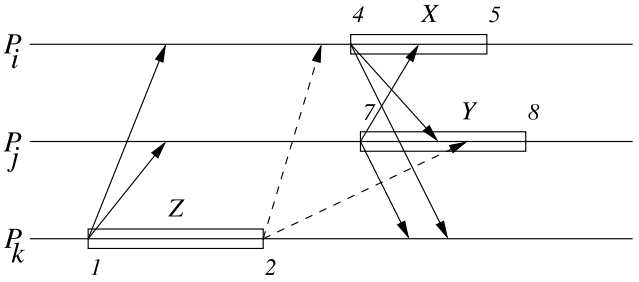

Fig. 7. No overlap, a potential false positive.

2. Let t_{last} ($\in [\max_i(I_i.t_s), \max_i(I_i.t_s) + \Delta]$) be the instant at which the last of the values in *Value_Vector*$[z]$, for all z, are received at the sink or observer. Two cases are possible at the sink or observer.
   (a) The values *Value_Vector*$[z]$, for all z, are not overwritten until after t_{last}. Therefore, there is at least one instant when *Value_Vector*$[z](\forall z)$ is available, and ϕ can be correctly evaluated over this vector of values. If ϕ holds, we have a true positive.
   (b) At least one of these values *Value_Vector*$[z]$, for all z, is overwritten before they are all received. Therefore, there is no instant when *Value_Vector*$[z](\forall z)$ is available. Even if *Value_Vector*$[z](\forall z)$ held in this duration, it is not observable, and ϕ cannot be correctly evaluated over this vector of values. If ϕ holds, we have a false negative.
3. Consider the duration $[\max_i(I_i.t_s) - |overlap|, \max_i(I_i.t_s)]$. Let P_j be any process whose interval ends at $\max_i(I_i.t_s) - |overlap|$. Let $k = \text{argmax}_i(I_i.t_s)$. There are two cases. P_j's new value, which negates ϕ, may or may not have reached the sink before P_k's value reaches the sink.
   (a) In the former case, there is no instant when *Value_Vector*$[z]$ $(\forall z)$ is available at the sink. Hence ϕ is not evaluated over the value vector, resulting in a true negative.
   (b) In the latter case, there is an instant when *Value_Vector*$[z]$ $(\forall z)$ is available at the sink. Hence ϕ when evaluated over the value vector, results in a false positive.
4. Consider the duration $[\max_i(I_i.t_s) - |overlap|, \max_i(I_i.t_s)]$. Let P_j be any process whose interval ends at $\max_i(I_i.t_s) - |overlap|$. Let $k = \text{argmax}_i(I_i.t_s)$. The new value of P_j, which negates ϕ, reaches the sink/observer before $\max_i(I_i.t_s)$ at which P_k's interval begins. When the event notification from P_k arrives at the sink, P_j's value has already changed at the sink. Therefore, there is no instant when *Value_Vector*$[z](\forall z)$ is available at the sink, and ϕ is correctly detected as not holding for this value vector. □

From the application's perspective, we can classify the outcome of detection or non-detection as follows.

Corollary 1. *For a single observer in a system without any synchronized clocks, for the detection algorithm in Algorithm 1, we have for any $\mathcal{I}$ for which ϕ is true:*

1. *Positive detection* $\Longrightarrow$ *overlap* $> -\Delta$
2. *Negative detection* $\Longrightarrow$ *overlap* $< \Delta$.

When *overlap* in $[-\Delta, \Delta]$, we cannot predict the outcome and there will be *potential false positives* when *overlap* in $[-\Delta, 0]$ and *potential false negatives* when *overlap* in $[0, \Delta]$. Although we expect that in pervasive environments, these cases are infrequent, we still do not know the *overlap* and cannot identify these *potential false outcomes* from the definitive outcomes.

We enhance this algorithm in two ways; see Table 1.

Table 1
Comparison of proposed algorithms.

Algorithm	Features
Simple Clock-Free Algorithm (Section 3.1)	*Value_Vector* at sink (or at all) node(s); Send to sink (or broadcast) event notification; Evaluate ϕ whenever *Value_Vector* changes
Interval Vector Algorithm (Section 3.2)	*Value_Vector*, *Interval_Vector* at all nodes + Broadcast *Value_Vector*, *Interval_Vector*+ Evaluate ϕ by all nodes whenever *Interval_Vector* changes
Consensus Algorithm (Section 3.3)	Interval Vector algorithm + Transmit (or broadcast) *Consensus_Message*+ Consensus evaluated at sink (or by all) node(s)

3.2. Improved Accuracy using Interval Vectors

We propose using an *Interval_Vector*, in conjunction with the *Value_Vector*, to track more up-to-date sensed values in the *Value_Vector*. This helps to reduce the number of *potential false outcomes*. The vectors track the interval numbers and the corresponding sensed value readings of the latest intervals being considered. $Interval\_Vector[j] = k$ (at any process) is used to identify the kth interval at process P_j, that began when the kth event was sensed by the sensor at P_j and that would end at the $k+1$th event sensed by P_j. *Value_Vector*$[j] = x$ (at any process) is used to identify that the value x held during the $Interval_Vector[j]$th interval at process P_j.

Although the proposed interval vectors are similar to logical vector clocks [25], there are several differences – for example, (i) there is no underlying computation message exchange and all event notifications using *Interval_Vector*s are control messages, whereas logical vector clocks operate by piggybacking timestamps on the underlying computation messages to advance vector time; (ii) *Interval_Vector*s track the progress of the local interval counter at each process by catching up or synchronizing on the latest known intervals of other processes, and do not track the causality induced by message communication; whereas the logical vector clocks track the causality induced by message sends and receives; (iii) on receiving an *Interval_Vector*, the local component of the *Interval_Vector* does *not* advance; whereas on receiving a message with a piggybacked timestamp, the logical vector clock advances the local component. A variant of the interval vector was used in [6]. A further discussion of interval clocks can be found in [20].

The algorithm is given in Algorithm 2. There is no improvement in the characterization of the outcomes, over that given in Theorem 1 and Corollary 1. However, we do get a quantitative improvement by way of reducing false outcomes. Specifically, in Theorem 1.2, the number of false negatives is decreased, and in Theorem 1.3, the number of false positives is decreased, as explained by the following examples.

Examples. In Fig. 1, P_k can now detect ϕ using the *Interval_Vector* to update the ith component of its *Value_Vector*. Thus, a false negative gets eliminated. However, in Fig. 2, the use of *Interval_Vector* cannot help P_k in eliminating its false negative conclusion. Similarly, in Fig. 3, the use of *Interval_Vector* cannot

Algorithm 2 Interval Vector Algorithm: Code at P_i to detect a predicate using event *vector* notifications.

int: **array** $Interval\_Vector[1 \ldots n]$
int: **array** $Value\_Vector[1 \ldots n]$
boolean: new

When event $e = (i, val)$ occurs at P_i:
(1) $Interval\_Vector[i] + +$
(2) $Value\_Vector[i] \longleftarrow val$
(3) broadcast to $P_j \in \mathcal{P} \setminus \{P_i\}$, event notification $(i, Interval\_Vector, Value\_Vector)$
(4) $Evaluate\_State(i, Interval\_Vector, Value\_Vector)$

On P_i receiving event notification $e = (z, IV, VV)$ from P_z:
(1) $Evaluate\_State(z, IV, VV)$

$Evaluate\_State(z, IV, VV)$ at P_i:
(1) $new \longleftarrow 0$
(2) **for** $x = 1$ to n
(3) **if** $IV[x] > Interval\_Vector[x]$ **then**
(4) $new \longleftarrow 1$
(5) $Interval\_Vector[x] \longleftarrow IV[x]$
(6) $Value\_Vector[x] \longleftarrow VV[x]$
(7) **if** $new = 1$ or $z = i$ **then**
(8) **if** $\phi((\forall j) Value\_Vector[j]) = true$ **then**
(9) observed $Value\_Vector$ satisfies ϕ
(10) raise alarm/actuate

help any process in eliminating its false negative conclusion. But in Fig. 4, the use of *Interval_Vector* allows at least one process, viz., P_j, to eliminate its false negative conclusion.

In Fig. 5, the use of *Interval_Vector* cannot help P_i or P_j in eliminating their false positive conclusion. However, in Fig. 7, the use of *Interval_Vector* allows P_j reading a false positive to eliminate it.

Theorem 2. *For a single observer in a system without any synchronized clocks, for the detection algorithm in Algorithm 2, we have for any $\mathcal{I}$ for which ϕ is true:*

1. $overlap \geq \Delta \Longrightarrow \phi$ *is correctly detected*
2. $0 \leq overlap < \Delta \Longrightarrow$ *any outcome is possible*
3. $0 \geq overlap > -\Delta \Longrightarrow$ *any outcome is possible*
4. $overlap \leq -\Delta \Longrightarrow \phi$ *is correctly detected as not holding.*

Proof. 1. Same as for Theorem 1.1.
2. Same as for Theorem 1.2 except that some false negatives are avoided in Case 2(b) because interval vectors allow transitive information to aid in the early receipt of $Value\_Vector[z]$ $(\forall z)$. In these cases, the existing values of $Value\_Vector[z]$ may not be overwritten early enough, and a true positive results instead of a false negative.
3. Same as for Theorem 1.3 except that some false positives are avoided in Case 3(b) because interval vectors allow transitive information to aid in the early receipt of P_j's new value, which negates ϕ, at the sink, before P_k's values reaches the sink. Hence the false positive becomes a true negative.
4. Same as for Theorem 1.4. □

Corollary 2. *For a single observer in a system without any synchronized clocks, for the detection algorithm in Algorithm 2, we have for any $\mathcal{I}$ for which ϕ is true:*

1. *Positive detection* $\Longrightarrow overlap > -\Delta$
2. *Negative detection* $\Longrightarrow overlap < \Delta$.

3.3. Improved Accuracy through Consensus

Algorithms 1 and 2 have these drawbacks:

1. a positive detection may be false (with $overlap \in [-\Delta, 0]$)
2. a negative detection may be false (with $overlap \in [0, \Delta]$).

Our next algorithm eliminates the first drawback, and reduces the number of instances that suffer from the second drawback. Rather than a positive and a negative bin, it creates *three* bins: positive, negative, and *borderline*. Positives are all true with $overlap \in (0, \infty)$; borderline cases satisfy $overlap \in (-\Delta, \Delta)$; negatives are true with $overlap \in (-\infty, 0)$ or a few "inevitable" false cases with $overlap \in (0, \Delta)$. The application is free to classify the borderline cases in either direction.

Observe in Algorithms 1 and 2, that the execution for each sensed event is very simple, namely statements *Evaluate_State* .(2) and .(1)–(8), respectively. The information to evaluate the statement(s) is broadcast (assume so for Algorithm 1 also), hence it is available to all the sensors without added message cost, for "almost free". As all the sensors execute the procedure instead of some sink, we have multiple (n) observers. The execution of the statement(s) is affordable, and all sensors get to know the outcome.

However, due to Theorem 1.(2,3) and Corollary 1, and due to Theorem 2.(2,3) and Corollary 2, each observer may arrive at different outcomes. To see this, consider a worst-case scenario where all the n sensors detect (almost) simultaneous state changes, and execute the event broadcast in response. This is a n-way race condition. Due to non-determinism of message transmission times, each of the n processes will observe one of $O(n!)$ possible orderings of the n broadcasts. Assuming that $overlap \in (-\Delta, \Delta)$, there may be *potential false negatives* and *potential false positives*, and these will be different for the n observers.

Examples. In Fig. 2, $Interval\_Vector = [4, 7, 1]$ will be detected by P_i and P_j but not P_k; P_k has a false negative.

In Fig. 4, $Interval\_Vector = [4, 7, 1]$ will be detected by P_j but not P_i and P_k; P_i and P_k have a false negative.

In Fig. 5, $Interval\_Vector = [4, 7, 1]$ will be detected by P_i and P_j but not P_k; P_i and P_j have a false positive.

So it appears we have entropy or chaos in the observations among the n processes.

More generally, we have the following. The application is observing a *single* instance of the real-world execution. There are a maximum of np state transitions in the global execution for a global observer, corresponding to the np sensing events. In the Simple Clock-Free algorithm, there are also np global states observed by any observer. However, there are two levels of approximations in the observations.

1. Each of the n observer processes will observe its best approximation of the actual global states at each of the np events; each of the n observers may see different approximations of the same actual global state.
2. Further, the np approximations of the actual global states seen by a observer process will be observed in an ordering that is the best approximation to the actual ordering of the actual global states at the np events. (That is, each of the n observers may observe a different permutation of (its approximations of the global states at) the np events, than each other or in the actual execution. Thus, of the $O(\frac{(np)!}{(p!)^n})$ valid permutations, there is one permutation for the actual execution, and some n will be observed.)

Both levels of approximations are the best approximations that can be made by the algorithm, and inevitable due to the message transmission latencies that arise at run-time. The Interval Vector algorithm has the same properties, but gives better approximations as discussed in Section 3.2.

As there are only np true global states, that occur in one sequence, and the n observers are trying to all observe their approximations of the np states in an approximate serial order, we propose to run a *consensus* algorithm among the n processes' inferences about their approximate observations. We note here that the algorithm considers primitive CGSs as well as composite CGSs.

Consider any of the np sensing events e. The event notification (EN) broadcast communicates it to all processes. *Evaluate_State* gets executed at each process, based on the latest state information at that process. So there are n evaluations system-wide of ϕ over the most recent estimate (approximation) of the state immediately following e. Globally for all np sensing events, there are n^2p evaluations.

However, due to asynchrony in message transmissions and race conditions, the n evaluations may not see the same state. In order for the witness observers to corroborate their observations of positives determined from *Evaluate_State(IV)*, we use a *Consensus_Message(IV)* that may be broadcast (or sent to a sink). To determine the number of witnesses of the *same* state IV, each process collates the received *Consensus_Messages*. Specifically, when a process receives a *Consensus_Message(IV)*, it maintains a *count* of "confirmations" for that IV. Thus, it counts the number of witnesses of any state by counting the number of *Consensus_Message*s it receives for that state's IV. The algorithm is given in Algorithm 3.

- If $\phi(e)$ is true, *flag* on the *EN* broadcast is set. When the *EN* is received, and locally where e occurs, each P_i tests if it can confirm the IV of e using its local knowledge as an approximation to e (*Evaluate_State*.(1–4)). P_i can "locally confirm" P_z's VV, even if $IV \neq Interval\_Vector$ – this is useful [17] because if every P_i "locally confirms" P_z's VV, it must have occurred in physical time. If P_i can confirm, it sends a *Consensus_Message*(IV).

  The n^2p *Evaluate_State* executions for the np sensing events, can trigger up to n^2p *Consensus_Messages* globally.
- The receipt of an *EN* also creates a potentially *new* observation point at a *composite state* formed by merging the received *EN*'s IV and VV into the local *Interval_Vector* and *Value_Vector*, representing the IVs and VVs received cumulatively so far (*Evaluate_State*.(5–12)). A new (neither seen nor evaluated locally so far) composite state is formed if and only if $z\_new = 1 \wedge i\_new = 1$ across all n iterations of the loop of line (6). $z\_new = 1$ is needed to ensure that the message from P_z brings in new information. $i\_new = 1$ is needed to ensure that a new state, and not the same one as included in IV, has formed locally at P_i, i.e., there has been progress at P_i due to a local event or a message receive from another process. If a new composite state *Interval_Vector* forms in *Evaluate_State*.13 (this happens at most $n(n-1)p$ times globally), and ϕ holds, a *Consensus_Message* is sent.

Worst-case number of *Consensus_Messages* is $2n^2p - np$. By combining sends (lines (4) and (16)), this is at most n^2p.

Note, the *Consensus_Message* need not be broadcast. If it is broadcast (including to the sender), all processes learn the results for "almost free" and will see the same result. As explained before, on receiving a *Consensus_Message(IV)*, a process counts the number of received *Consensus_Messages* for that specific IV in the *count* field of type *Interval_Vector_Record*. The use of a timer set to 2Δ in maintaining the list of *Interval_Vector_Record*s serves as a performance optimization mechanism and is described later in Section 4.3. The idea is that from the time the first *Consensus_Message* for a new IV is seen, all *Consensus_Messages* for that IV that are to arrive must arrive within 2Δ.

Algorithm 3 Consensus Algorithm: Code at P_i to detect a predicate using consensus.

```
int: array Interval_Vector[1...n]
int: array Value_Vector[1...n]
type Interval_Vector_Record
     array [1...n] of int: vector
     int: count
Interval_Vector_Record: list Interval_Vector_History
boolean: flag, z_new, i_new

When event e = (i, val) occurs at P_i:
(1)  Interval_Vector[i]++
(2)  Value_Vector[i] ← val
(3)  if φ((∀j)Value_Vector[j]) = true then
(4)       flag ⟵ 1
(5)  else flag ⟵ 0
(6)  broadcast to P_j ∈ P \ {P_i}, event notification
     (i, Interval_Vector, Value_Vector, flag)
(7)  Evaluate_State(i, Interval_Vector, Value_Vector, flag)

On P_i receiving event notification e = (z, IV, VV, b) from P_z:
(1)  Evaluate_State(z, IV, VV, b)

Evaluate_State(z, IV, VV, b) at P_i:
(1)  if b = 1 then
(2)       if IV[i] = Interval_Vector[i] then
(3)            VV of z is "locally confirmed" by i to satisfy φ
(4)            broadcast to P_j ∈ P, Consensus_Message(z, IV, i)
(5)  z_new, i_new ⟵ 0
(6)  for x = 1 to n
(7)       if IV[x] > Interval_Vector[x] then
(8)            z_new ⟵ 1
(9)            Interval_Vector[x] ⟵ IV[x]
(10)           Value_Vector[x] ⟵ VV[x]
(11)      else if IV[x] < Interval_Vector[x] then
(12)           i_new ⟵ 1
(13) if z_new = 1 ⋀ i_new = 1 then
(14)      if φ((∀j)Value_Vector[j]) = true then
(15)           observed Value_Vector satisfies φ
(16)           broadcast to P_j ∈ P,
               Consensus_Message(z, Interval_Vector, i)

On P_i receiving Consensus_Message(trigger, IV, s) from P_s:
(1)  if IV is a new interval vector then
(2)       create record x of type Interval_Vector_Record
(3)       x.vector ⟵ IV
(4)       x.count ⟵ 1
(5)       insert x in Interval_Vector_History
(6)       start timer for 2Δ for x
(7)  else
(8)       let x be record of IV in Interval_Vector_History
(9)       x.count++
(10) if x.count = n then
(11)      Corollary 3.1; raise alarm/actuate(true positive,IV)
(12) else
(13)      Corollary 3.2; await more confirmations or timer pop

On P_i getting a timer pop for Interval_Vector_History.x:
(1)  Corollary 3.2; raise alarm/actuate(borderline,IV)
(2)  delete record x from Interval_Vector_History
```

To characterize the extent to which the witness observers corroborate their observations of positives determined from *Evaluate_State*, we have introduced the "confirmation" *count*. If (all // at least one but not all //none) processes see the same state, identified using its *Interval_Vector*, and ϕ is true in it, then we say that that state is confirmed by (all // only some // none).

Definition 2. $\phi(VV, IV)$ is:

1. confirmed by all iff $count(IV) = n$
2. confirmed by only some iff $n > count(IV) > 0$
3. confirmed by none iff $count(IV) = 0$.

In the ideal case (far-spaced global interval transitions, spaced further than Δ apart, across all sensors), each of the *np* global states following the sensed events will be observable by all processes. For each such global state in which ϕ was true, there will be n confirmations (one by each observer). If *Consensus_Message* is broadcast, all processes learn about it. For the non-ideal case, the number of confirmations is less than n. We have this result:

Theorem 3. *For n observers in a system without any synchronized clocks, for the detection algorithm in Algorithm 3, we have for any IV for which ϕ is true:*

1. $overlap \geq \Delta \Longrightarrow \phi$ *is confirmed by all*
2. $0 \leq overlap < \Delta \Longrightarrow \phi$ *is confirmed by all, only some, or none*
3. $0 \geq overlap > -\Delta \Longrightarrow \phi$ *is confirmed by only some, or none*
4. $overlap \leq -\Delta \Longrightarrow \phi$ *is confirmed by none.*

Proof. 1. Let $k = \text{argmax}_i(I_i.t_s)$. We have two cases.
   (a) The CGS corresponding to IV is elementary. At P_k, ϕ (*Value_Vector*) evaluates to true at the event occurring at $I_k.t_s$. The event notification reaches all processes within $I_k.t_s + \Delta$ and all processes will locally confirm the VV of P_k (*Evaluate_State*.(1)–(4)) because $overlap \geq \Delta$ and the processes have not seen another event. At each of the n local confirmations, a *Consensus_Message* for the IV of P_k is broadcast. Hence $count(IV) = n$ at each observer.
   (b) The CGS corresponding to IV is composite. In or before the duration $[I_k.t_s, I_k.t_s + \Delta]$, all the processes will have received the event notifications from P_k and all other processes. They will all detect a new composite state when the last of these event notifications arrives. Note that no process changes its state until after this duration finishes. Hence, all processes detect the *same* new composite state. When each process detects a new composite state and evaluates ϕ over it (*Evaluate_State*.(13)–(16)), it broadcasts a *Consensus_Message* for the IV of the composite state. Hence $count(IV) = n$ at each observer.

2. Let $k = \text{argmax}_i(I_i.t_s)$. We have two cases.
   (a) The CGS corresponding to IV is elementary. At P_k, ϕ (*Value_Vector*) evaluates to true at the event occurring at $I_k.t_s$. The event notification reaches all processes within $I_k.t_s + \Delta$. However, as $\Delta > overlap$, some x processes, $x \in [0, n-1]$, may sense a new event before this event notification arrives. These x processes will not be able to locally confirm the VV of P_k, while the remaining $(n-1) - x$ processes will locally confirm the VV of P_k (*Evaluate_State*.(1)–(4)) because they have not seen another event. At each of the $(n-1) - x$ local confirmations, a *Consensus_Message* for the IV of P_k is broadcast. In addition, P_k locally confirms its own VV and broadcasts a *Consensus_Message* for the corresponding IV. Hence $count(IV) = (n - x) \in [1, n]$ at each observer.
   (b) The CGS corresponding to IV is composite. In or before the duration $[I_k.t_s, I_k.t_s + \Delta]$, all the processes will have received the event notifications from P_k and all other processes. They will all detect a new composite state when the last of these event notifications arrives. As $overlap < \Delta$, some x processes, where $x \in [0, n]$, may change their state before the last of these notifications arrives, and send their own *EN*s negating the state corresponding to the IV. Hence, at most $n - x$ processes detect the *same* new composite state, while at least x processes will detect some other state. When each of the at most $n - x$ processes detects the same new composite state and evaluates ϕ over it (*Evaluate_State*.(13)–(16)), it broadcasts a *Consensus_Message* for the IV of the composite state. Hence $count(IV) \leq (n - x) \in [0, n]$ at each observer.

3. Let $k = \text{argmax}_i(I_i.t_s)$. Let $l = \text{argmin}_i(I_i.t_f)$. We have two cases.
   (a) The CGS corresponding to IV is elementary. This state may form at P_k if P_l's new changed value does not reach P_k at $I_k.t_s$. By the time the event notification from P_k reaches P_l, P_l has sensed a new value. Hence, P_l will not be able to locally confirm the VV corresponding to the IV. However, the other $n - 2$ processes may be able to locally confirm the VV corresponding to the IV if they do not sense a new local event before receiving the event notification from P_k. Hence, these other $n-2$ processes besides P_l will broadcast the *Consensus_Message* if they can locally confirm the IV (*Evaluate_State*.(1)–(4)). In addition, P_k locally confirms its own VV and broadcasts a *Consensus_Message* for the corresponding IV. Hence, $count(IV) \in [0, n-1]$ at each observer.
   (b) The CGS corresponding to IV is composite. By the time the event notification from P_k reaches P_l, it has sensed a new value. Hence, at P_l, a composite CGS corresponding to the IV will not be created. However, a composite CGS corresponding to the IV may be created at processes besides P_l if (i) event notifications about the newly sensed values at P_l or other processes (that negate ϕ) and denote a different IV may not have reached the other processes; and (ii) they themselves have not sensed a new value denoting a different IV. Hence, these other processes besides P_l may broadcast the *Consensus_Message* if ϕ evaluates to true (*Evaluate_State*.(13)–(16)). Hence, $count(IV) \in [0, n-1]$ at each observer.

4. Let $k = \text{argmax}_i(I_i.t_s)$. Let $l = \text{argmin}_i(I_i.t_f)$. Before time $I_k.t_s - \Delta$, P_l has sensed a new event. This event notification has reached P_k before $I_k.t_s$. Hence, the IV cannot be created by P_k and an elementary CGS for the IV cannot form. Similarly, by the time P_k's event notification reaches any other process P_m, the new event notification from P_l has reached that process P_m and overwritten the value corresponding to the IV under consideration. Hence, the IV with P_k's value and P_l's value corresponding to the IV under consideration can never be formed at any process P_m or P_l. Hence, no composite CGS can form for the IV. Hence, no *Consensus_Message* is broadcast for the IV, and $count(IV) = 0$ at each observer. □

In the proof of Theorem 3, note that there is no relationship between the size of an anti-chain forming a CGS, and the number of processes that can confirm the predicate in the CGS.

From the application's perspective, we can classify the outcome of detection or non-detection into three bins: *positive*, *borderline*, and *negative*, as follows. We also classify the examples in Figs. 1–7 in these bins. The figures do not show the *Consensus_Message*s to avoid overcrowding, but visualize that the *Consensus_Message* is broadcast at each execution of *Evaluate_State* in which ϕ is true.

Corollary 3. *For n observers in a system without any synchronized clocks, for the detection algorithm in Algorithm 3, we have for any IV for which ϕ is true:*

1. *Confirmed by all $\Longleftrightarrow$ positive bin $\Longrightarrow$ true positive $\Longrightarrow$ overlap ≥ 0.*
   Examples: Fig. 1.
2. *Confirmed by only some $\Longleftrightarrow$ borderline bin $\Longrightarrow$ $\Delta >$ overlap $> -\Delta$.*
   The application can choose to classify this case as either a positive or negative, depending on application semantics. It is better to err on the side of safety.
   Examples: Figs. 2, 4 and 5.
3. *Confirmed by none $\Longleftrightarrow$ negative bin $\Longrightarrow$ (true negative ($\Longrightarrow$ overlap < 0) $\oplus$ false negative having $0 <$ overlap $< \Delta$).*
   Examples: Figs. 6 and 7; and Fig. 3, resp.

Thus, those states that are confirmed by all the observers (and in which ϕ is true) correspond to Corollary 3.1. Those that are confirmed by only some observers (and in which ϕ is true) correspond to Corollary 3.2. Those that are seen by none of the observers correspond to Corollary 3.3.

The algorithm gives the following advantages:

1. There are no false positives, and all interval overlaps with *overlap* $\geq \Delta$ and some with *overlap* $\in [0, \Delta)$ are explicitly identified and declared.

   Examples. In Fig. 1, $IV = [4, 7, 1]$ will be confirmed by all, i.e., by P_i, P_j, and P_k.

2. Some of the cases having *overlap* $\in (-\Delta, \Delta)$ are explicitly identified.

   Examples. In Fig. 2, $IV = [4, 7, 1]$ will be confirmed by P_i and P_j but not P_k; in Fig. 4, $IV = [4, 7, 1]$ will be confirmed by P_j but not P_i and P_k; in Fig. 5, $IV = [4, 7, 1]$ will be confirmed by P_i and P_j but not P_k and would be a false positive if declared.

   Such cases are placed in the bin "borderline" and the application has the choice of raising an alarm or not.

Examples. Fig. 3 is an unfortunate false negative, but Theorem 3.(2,3,4) shows that *overlap* $\in (-\infty, \Delta)$ must hold. Figs. 6 and 7 are identified as true negative.

For the borderline bin, the application can treat the cases with *overlap* > 0 as negatives (because the overlap period was only a small positive), or cases with *overlap* < 0 as positives (because the interval "almost" overlapped). Essentially, all cases in this bin can be treated alike.

It is important to understand that the algorithm implicitly builds on-the-fly the lattice of *those* (up to n^2p) states that the nodes do actually observe collectively, based on the *np* events. It also performs the corroborations among the n^2p observations on-the-fly.

Example. In Fig. 8, the global state passes through:

$\cdots [3, 6, 0], [4, 6, 0], [4, 7, 0], [4, 7, 1], [4, 7, 2], [4, 8, 2] \cdots$.

Each sensed event triggers broadcast of the event notification, indicated by the solid arrows. The ensuing execution of *Evaluate_State* at each process (indicated by the circles) updates their *Interval_Vectors*. Assume that ϕ is true in $[4, 7, 1]$. In this example, each process is able to construct this state $[4, 7, 1]$ as soon as it receives the event notifications from the other two processes. On constructing $[4, 7, 1]$, *Evaluate_State* locally detects ϕ and broadcasts the *Consensus_Message*, indicated by dashed arrows. Once $IV([4, 7, 1]).count = 3$ for the *Interval_Vector_Record*

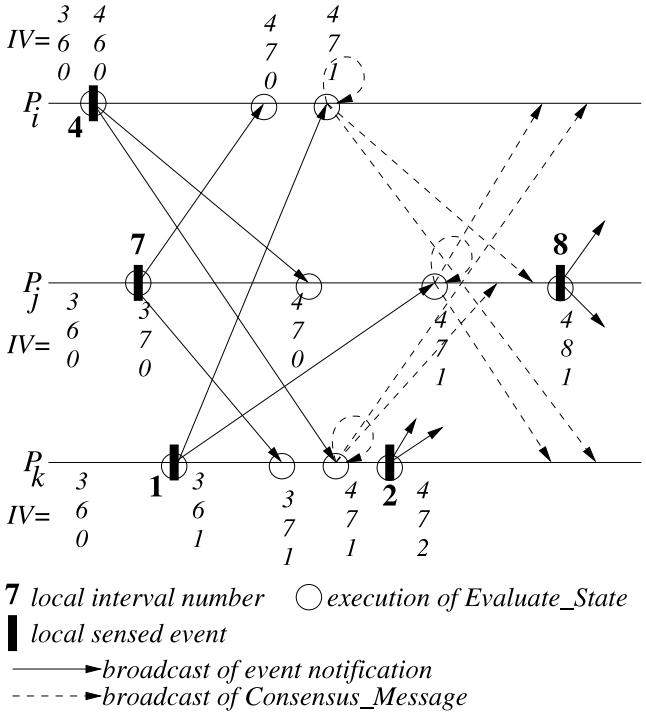

Fig. 8. Example of on-the-fly state construction and evaluation.

of *Consensus_Message*($[4, 7, 1]$) at a process, the vector $[4, 7, 1]$ is "confirmed by all" at that process and is hence a true positive. This will happen at all three processes. If at one process $IV([4, 7, 1]).count = 3$, then it is guaranteed that all processes will eventually see the count 3 locally.

Now visualize that P_j senses the next event (numbered "8") locally just before receiving the event notification from P_k, i.e., it transitions from $[4, 7, 0]$ to $[4, 8, 0]$ instead of to $[4, 7, 1]$. Then each process receives exactly 2 confirmations of $[4, 7, 1]$ from P_i and P_k, and this $IV = [4, 7, 1]$ can be classified in the "borderline" bin. Each process always receives the same number of confirmations for any particular IV.

In another scenario, imagine all processes locally sense a changed value in physical time immediately after P_k begins interval 1. This will result in an "inevitable" false negative for $IV = [4, 7, 1]$.

4. Performance

As a baseline for comparison, observe that *np* transmissions are essential to report the sensed events to a sink *even* for centralized on-line detection *using physically synchronized clocks*.

4.1. Simple Clock-Free Algorithm

The algorithm uses *np* event notifications.

- If sent to a sink, the messaging cost is the *same as* for centralized on-line detection with physically synchronized clocks.
- If broadcasting is done instead of sending to one sink, every process can know the impact of each sensed event (subject to our approximation results). In a single-hop or small wireless network, the broadcast is just a little more expensive. In a larger network, the extra messaging goes up by a factor of $\frac{n}{\log n}$ in a tree configuration and by a constant factor in a linear configuration.

  With broadcasting, the *np* transmissions result in n^2p executions of *Evaluate_State* across all the nodes, instead of *np* at a single sink.

4.2. Interval vector algorithm

The above analysis for the broadcast case applies except that the broadcast of event notifications is of 2 vectors instead of 2 integers.

4.3. Consensus algorithm

The first phase costs the same as the Interval Vector algorithm, which was analyzed in Section 4.2. We incur the following cost for the second phase to run consensus. Across the n^2p executions of *Evaluate_State*, *only* for those d times in which ϕ is evaluated to true (lines (1–3) and/or (13–14)), a *Consensus_Message* is transmitted. The worst-case, when $d = 2n^2p - np$ or simply n^2p (see Section 3.3), is very unlikely. If *Consensus_Message* is broadcast instead of transmitted to a sink, each node knows the precise outcome. There is not much difference between a broadcast and a "point-to-point" message in small networks (Section 4.1).

However, the *expected case* occurs when the predicate ϕ occasionally becomes true, and there are few race conditions because human and physical object movements in pervasive environments are typically much slower than the latencies that determine Δ. (In related work [15], simulations and analytical results for a smart office show that increasing message delays over a large range does not significantly increase probability of incorrect detection.) In our expected case, $d \in [0, n^2p]$ but $d \ll n^2p$. There will be d transmissions (or broadcasts) of *Consensus_Message*. As $d \ll n^2p$, and we have np *essential* transmissions of event notification messages for on-line detection even by a single sink (even with synchronized clocks), the consensus phase is not expected to increase the messaging cost noticeably! Yet, it offers the advantage of eliminating false positives and of classifying outcomes in the "borderline" bin.

In the worst-case, in which there are n^2p transmissions of the *Consensus_Message*, a node receives n^2p *Consensus_Messages*; at most n^2p will have unique IV vectors. A naive approach to correlate the IVs in these *Consensus_Messages* tracks n^2p entries in the *Interval_Vector_History*. This history can be a sorted list based on n keys, where the ith key is the component $IV[i]$. Then the access cost for inserting, deleting, and updating an index entry is $n(\log\ n^2p) = 2n \log\ n + n \log\ p$. Smart data structures can be used instead of the list and we can perform garbage collection to reduce this number significantly. In Fig. 3, we use a list and a simple observation to age and purge the record of an IV within 2Δ time of its first appearance in the list. The observation analyzes the slowest case. A locally sensed event causes *Evaluate_State* to detect ϕ, broadcast the event notification, and insert a record of the corresponding IV in the local *Interval_Vector_History*. Within Δ time, the event notification reaches all nodes, and their *Interval_Vector* is greater than or equal to the broadcast IV; if they also evaluate ϕ to be true for the IV broadcast, they will also send a *Consensus_Message* that must be received by others within another Δ period. Hence, if any confirmations of an entry in *Interval_Vector_History* arrive, they must within 2Δ of the insertion of the entry in the local *Interval_Vector_History*.

5. Discussion

Application Scenario: Consider a big exhibition hall within a convention center. The exhibition hall has k doors for entry-cum-exit, and has a room capacity of 100 people. A sensor at each door detects the movements of people in and out of the hall, by using RFID scanning of the convention badges (tickets). Each sensor is modeled as a process P_i, and tracks two variables: x_i, the number of people entered through that door, and y_i, the number of people departed through that door. An event at a sensor is the entry or exit of a person through the corresponding door. The global relational predicate to be detected is $\sum_{i=1}^{k}(x_i - y_i) > 100$. When the predicate becomes true, entry into the hall is not allowed, until the predicate becomes false. Although physically synchronized clocks could be implemented in this urban setting, their overhead is not necessary because the precision they provide is more than required for detecting human locomotion in this detection problem. Immediate detection of the predicate is required to prevent overcrowding and violating fire code norms. Other algorithms [15,20] cannot detect this predicate until an additional event is sensed at each of the sensors (doors). Due to concurrent traffic through the multiple doors (representing a race condition) and variations in the transmission delay, a false positive may occur when occupancy is below 101, while a false negative may occur when the occupancy is above 100. The consensus based algorithm will be able to place such false positives and false negatives in the borderline bin, and treat them as positives, to err on the safe side.

In small sensor networks that use a shared medium, there is a natural occurrence of total order and causal order among the broadcasts [1]. Even if the shared medium is not present or these message orders do not naturally occur, middleware could provide these orders. Analyzing the impact of these orders on the detection algorithm design and characterizing the errors is an open problem. We did not make any assumptions about these orders to make the results applicable to wired, wireless, and hybrid networks.

Our algorithms are distributed and symmetric, with low additional message overhead above that for centralized detection at a single sink. Distribution and symmetry are more conducive to tolerating failures and allowing sensor node mobility with few adaptations. This deserves further study.

Our model allows communication failures by way of message loss only. Except for potential false positives and false negatives in the temporal vicinity of a message loss, there are no long-term ripple effects on future detection.

To provide fault-tolerance, we can explore several directions, e.g., refine the "borderline" bin. If the number of witnesses is closer to 1, the outcome is likely a true negative; if closer to n, likely a true positive. This is intuitively supported by probability. Or, depending on the application, a borderline outcome can be classified as a negative or a positive. For example, the predicate in the application scenario earlier in this section may treat borderline outcomes as positives to err on the safe side. Predicate ψ considered in Section 1 requires the firing of the rule only if the condition is stable. Hence, a borderline detection of the condition should be treated as a negative.

Table 2 compares the algorithms, and those in [20]. BC is the acronym for a broadcast. Any and all nodes can act as sink. Note, in the Interval Vector algorithm, a variable number of processes may see the same positive. The Consensus algorithm declares a positive if all see the same positive. Also, typically ϕ evaluates in $O(n)$. The algorithms in [20] cannot do immediate detection and the detection latency is unbounded. This makes them unsuitable for real-time applications to sense the physical world.

This is the first work that attempts immediate and repeated detection of predicates (conjunctive and relational) that held at an instant in physical time, over the sensed values of the physical world. Our approach is to build approximations to the actual states that the world execution passed through without incurring the overheads of building the state lattice. Drawbacks of the algorithms in [11,12] are that (i) they cannot do immediate detection and rather, wait for the next event to occur at each process before detection; (ii) they cannot do repeated detection because their algorithms hang after a predicate is once detected (see [22]); (iii) they detect predicates only in the *Possibly* and *Definitely* modalities and not those that held at an instant in

Table 2
Algorithms for detecting global predicates over sensed physical world properties.

Algorithm → properties ↓	Strobe vector algorithm[a] [20]	Strobe scalar algorithm [20]	Simple clock-free algorithm	Interval vector algorithm	Consensus algorithm
Message complexity	1 BC of size $O(n)$ /event	1 BC of size $O(1)$ /event	1 msg of size $O(1)$ to sink /event (can BC instead)	1 BC of size $O(n)$ /event	1 BC of size $O(n)$/event $+d$ messages (or BCs), where $d \in [0, n^2p]$
Processing	$O(n^2p)$/node + [$O(n^3p)$+ ($O(np)$ eval of ϕ)] at sink	$O(np)$/node + [$O(n^2p)$+ ($O(np)$ eval of ϕ)] at sink	$O(p)$/node + $O(np)$ eval of ϕ at sink (if BC, at all)	$O(n^2p)$/node + $O(np)$ eval of ϕ at sink or at all	$O(n^2p)$/node + $O(np)$ eval of ϕ/node + $O(d)$ at sink (or at all)
Detection latency	After intervals complete	After intervals complete	$\leq \Delta$	$\leq \Delta$	$\leq 2\Delta$
Observer independence	Yes	Yes	No	No	Yes
Detection by all observers	No extra msg cost	No extra msg cost	Use BC instead of msg to sink	No extra msg cost	No extra msg cost
overlap $\geq \Delta$	True positive	True positive	True positive	True positive	True positive
overlap $\in (0, \Delta)$	Some true positive; Some false negative	Some true positive; Some false negative	Some true positive; Some false negative	Some true positive; Some false negative (better than SCF)[c]	Some true positive; Some false negative[b]; some in *borderline*
overlap $\in (-\Delta, 0)$	True negative	Some true negative; Some false positive	Some true negative; Some false positive	Some true negative; Some false positive (better than SCF)[d]	Some true negative; Some in *borderline*
overlap $\leq -\Delta$	True negative	True negative	True negative	True negative	True negative

[a] If this algorithm uses a *borderline* bin also, some of the false negatives (*overlap* $\in (0, \Delta)$) & some of the true negatives (*overlap* $\in (-\Delta, 0)$) go in it. For the remaining false negatives, footnote (b) also applies.
[b] This algorithm cannot detect these as having occurred potentially, even for conjunctive ϕ. (Lattice evaluation can classify these in *borderline*. For relational ϕ, lattice evaluation can combine cases in *borderline* to a positive occurrence of some one state from the combination of the cases.)
[c] More accurate, i.e., fewer false negatives and more true positives, than Simple Clock-Free.
[d] More accurate, i.e., fewer false positives and more true negatives, than Simple Clock-Free.

physical time; (iv) they cannot detect predicates on sensed physical world values but only on in-network variables; (v) they detect only a conjunctive predicate and cannot detect a relational predicate. Predicate detection for pervasive environments was addressed in [15,20]. The algorithm in [15] detects a conjunctive predicate only after all but one sensors have sensed one more event, their next, locally. Further, it cannot do repeated detection because its algorithm hangs after a predicate is once detected. As [11,12,15] have different characteristics, we do not compare their performance metrics.

The three algorithms presented in this paper are the seminal results in this area of immediate and repeated predicate detection in pervasive environments. This is also the first work to provide guarantees on the performance. However, these algorithms offer rather weak guarantees. It is an open problem to provide stronger guarantees for the proposed algorithms, as also to devise newer algorithms with stronger guarantees.

Acknowledgment

A preliminary version of this result appeared in [21].

This work was supported by National Science Foundation grant CNS 0910988, "Context-Driven Management of Heterogeneous Sensor Networks."

References

[1] K. Birman, T. Joseph, Reliable communication in the presence of failures, ACM Trans. Computer Systems 5 (1) (1987).
[2] S. Bonetti, S. Mehrotra, N. Venkatasubramanian, Exploring quality in multi-sensor pervasive systems: a localization case study, in: IEEE Int. Conference on Pervasive Computing and Communications Percom Workshops, pp. 81–86, 2010.
[3] Y. Bu, T. Gu, X. Tao, J. Li, S. Chen, J. Lu, Managing quality of context in pervasive computing. in: International Conf. on Quality Software, pp. 193–200, 2006.
[4] Y. Bu, S. Chen, J. Li, X. Tao, J. Lu, Context consistency management using ontology based model, in: Proc. Current Trends in Database Technology, pp. 741–755, 2006.
[5] R. Cardell-Oliver, M. Renolds, M. Kranz, A space and time requirements logic for sensor networks. in: Second Int. Symp. on Leveraging Applications of Formal Methods, Verification, and Validation, pp. 283–289, 2006.
[6] P. Chandra, A.D. Kshemkalyani, Causality-based predicate detection across space and time, IEEE Trans. Comput. 54 (11) (2005) 1438–1453.
[7] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Trans. Computer Systems 3 (1) (1985) 63–75.
[8] B. Charron-Bost, Combinatorics and geometry of consistent cuts: application to concurrency theory, in: Workshop on Distributed Algorithms WDAG, Lecture Notes in Computer Science 392, Springer, pp. 45–56, 1989.
[9] R. Cooper, K. Marzullo, Consistent detection of global predicates. in: Proc. ACM/ONR Workshop on Parallel and Distributed Debugging, pp. 163–173, May 1991.
[10] C. Davison, D. Massaguer, L. Paradis, M. Rahimi, B. Xing, Q. Han, S. Mehrotra, N. Venkatasubramanian, Practical experiences in enabling and ensuring quality sensing in emergency response applications, in: IEEE Int. Conference on Pervasive Computing and Communications Percom Workshops, pp. 388–393, 2010.
[11] V.K. Garg, B. Waldecker, Detection of weak unstable predicates in distributed programs, IEEE Trans. Parallel & Distributed Systems 5 (3) (1994) 299–307.
[12] V.K. Garg, B. Waldecker, Detection of strong unstable predicates in distributed programs, IEEE Trans. Parallel & Distributed Systems 7 (12) (1996) 1323–1333.
[13] K. Henricksen, J. Indulska, A software engineering framework for context-aware pervasive computing, IEEE International Conference on Pervasive Computing and Communications Percom, pp. 77–86, 2004.
[14] P. Hu, J. Indulska, R. Robinson, An autonomic context management system for pervasive computing. in: IEEE International Conference on Pervasive Computing and Communications Percom, pp. 213–223, 2008.
[15] Y. Huang, X. Ma, J. Cao, X. Tao, J. Lu, Concurrent event detection for asynchronous consistency checking of pervasive context. in: IEEE Int. Conference on Pervasive Computing and Communications Percom, 2009.
[16] L. Kaveti, S. Pulluri, G. Singh, Event ordering in pervasive sensor networks. in: IEEE Int. Conference on Pervasive Computing and Communications Percom Workshops, 2009.
[17] A.D. Kshemkalyani, Temporal interactions of intervals in distributed systems, J. Comput. Syst. Sci. 52 (2) (1996) 287–298.
[18] A.D. Kshemkalyani, Temporal predicate detection using synchronized clocks, IEEE Trans. Comput. 56 (11) (2007) 1578–1584.
[19] A.D. Kshemkalyani, M. Singhal, Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, 2008.

[20] A.D. Kshemkalyani, Middleware clocks for sensing the physical world. in: Proc of the International Workshop on Middleware Tools, Services, and Run-Time Support for Sensor Networks MidSens'10, ACM DL, pp. 15–21, 2010.
[21] A.D. Kshemkalyani, Immediate detection of predicates in pervasive environments. in: 9th International Workshop on Adaptive and Reflective Middleware ARM'10, ACM DL, pp. 18–25, 2010.
[22] A.D. Kshemkalyani, Repeated detection of conjunctive predicates in distributed executions, Inform. Process. Lett. 111 (9) (2011) 447–452.
[23] A.D. Kshemkalyani, A. Khokhar, M. Shen, Execution and time models for pervasive sensor networks, in: IEEE International Symposium on Parallel and Distributed Processing IPDPS Workshops, pp. 639–647, 2011.
[24] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (7) (1978) 558–565.
[25] F. Mattern, Virtual time and global states of distributed systems, in: Parallel and Distributed Algorithms, North-Holland, 1989, pp. 215–226.
[26] J. Mayo, P. Kearns, Global predicates in rough real time. in: IEEE Symp. on Parallel and Distributed Processing, pp. 17–24, 1995.
[27] M. Roman, C. Hess, R. Cerqeira, A. Ranganathan, R.H. Campbell, K. Nahrstedt, A middleware infrastructure for active spaces, IEEE Pervasive Comput. 1 (4) (2002) 74–83.
[28] K. Romer, F. Mattern, Event-based systems for detecting real-world states with sensor networks: a critical analysis. in: DEST Workshop on Signal Processing in Wireless Sensor Networks at ISSNIP, pp. 389–395, 2004.
[29] M. Sama, D.S. Rosenblum, Z. Wang, S. Elbaum, Model-based fault detection in context-aware adaptive applications, in: Proc. 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering SIGSOFT'08/FSE-16, pp. 261–271, 2008.
[30] B. Sundararaman, U. Buy, A.D. Kshemkalyani, Clock synchronization for wireless sensor networks: a survey, Ad-Hoc Netw. 3 (3) (2005) 281–323.
[31] C. Xu, S.C. Cheung, Inconsistency detection and resolution for context-aware middleware support. in: Proc. ACM SIGSOFT Int. Symposium on Foundations of Software Engineering, pp. 336–345, 2005.
[32] C. Xu, S.C. Cheung, W. Chan, C. Ye, Heuristics-based strategies for resolving context inconsistencies in pervasive computing applications, in: 28th IEEE International Conference on Distributed Computing Systems, pp. 713–721, 2008.
[33] C. Xu, S. Cheung, W. Chan, C. Ye, Partial constraint checking for context consistency in pervasive computing, ACM Trans. Softw. Eng. Methodologies 19 (3) (2010) 1–61.

Ajay D. Kshemkalyani received the B.Tech degree in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 1987, and the MS and PhD degrees in Computer and Information Science from The Ohio State University in 1988 and 1991, respectively. He spent six years at the IBM Research Triangle Park working on various aspects of computer networks, before joining academia. He is currently a Professor in the Department of Computer Science at the University of Illinois at Chicago. His research interests are in distributed computing, distributed algorithms, computer networks, and concurrent systems. In 1999, he received the US National Science Foundation Career Award. He previously served on the editorial board of the Elsevier journal Computer Networks, and is currently an editor of the IEEE Transactions on Parallel and Distributed Systems. He has coauthored a book entitled Distributed Computing: Principles, Algorithms, and Systems (Cambridge University Press, 2008). He is a distinguished scientist of the ACM and a senior member of the IEEE.