Immediate Detection of Predicates in Pervasive Environments

Ajay D. Kshemkalyani
University of Illinois at Chicago
ajay@uic.edu

ABSTRACT

An important problem in pervasive environments is detecting predicates on sensed variables in an asynchronous distributed setting to determine context. 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 instance, by proposing a suite of 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.

Categories and Subject Descriptors

C.2.4 [**Distributed systems**]: Distributed applications

General Terms

Theory, Design, Performance

Keywords

sensor networks, pervasive computing, predicate detection

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, 6, 7, 15, 17]. 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 [3, 4, 10], 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, 6, 10, 17], except for [7], 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 [16]. Its costs are incurred by a lower layer, and it also imposes 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". 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 [14]. In very resource-constrained sensor systems or those in remote environments, clock synchronization service may be unavailable or be too expensive in terms of energy usage. Therefore, a companion paper [12] 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. Early detection was considered in [7]. 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 [12] 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 express the accuracy of our algorithms in terms of a parameter Δ. Define Δ as the bound on the asynchronous message transmission delay for a system-wide broadcast. Δ includes the delays for queuing in local outgoing and incoming buffers, retransmission if needed for reliability, 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 than predicted using the upper bound Δ.

The skew that governs the imprecision using physical clocks

ARM'2010, November 30, 2010, Bangalore, India.

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.

Our algorithms can detect conjunctive and relational predicates ϕ [5]. ϕ is conjunctive if $\phi = \wedge\phi_i$, where ϕ_i is defined on variables local to a single sensor. ϕ is relational if it is an arbitrary logical expression on system-wide variables. The characterization of the accuracy of detection is the same 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 [12].

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 [8]. 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 [11, 12]). Each sensor-actuator is modeled as a process $P_i (i \in [1 \ldots n])$; the local execution is a sequence of alternating states and state transitions caused by "relevant" events. An event is a sensing (observation) or actuation of the world plane by the network plane. Assume a maximum of p such 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.

Our algorithms work even if communication is unreliable. 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.

An event occurs whenever a sensed value, whether discrete or continuous, changes significantly. A sensed event is modeled as $e = (P_i, val, t_s)$ to denote the sensor process, value of the attribute sensed, and the physical time of occurrence. 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 $(P_i, val1, t1)$ and $(P_i, val2, t2)$ for that process-attribute, as: $I = (val1, t_s = t1, t_f = t2)$. 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.

Due to the inherent asynchrony of the control messages and lack of a global observer, there are many possible observations of the execution. The distributed computing literature has defined a lattice of global states and its sub-lattice of consistent global states for executions of distributed programs with semantic deterministic sends and receives [5]. This lattice has been used to make assertions about ϕ under *all possible runs* of the same distributed program. Due to variations in local program scheduling and transmission times, the same program passes through different paths in the state lattice in different executions. The time cost of detecting a relational predicate ϕ is exponential, $O(p^n)$ [5]. The 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 distributed program" of the world plane (see [12]); there is only the actual execution to make assertions about. And in the world plane execution, there is no "message passing" that can be captured by the network plane. The control messages of the network plane induce an *artificial (non-semantic)* lattice of consistent global states [12]. 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 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 (P_i, val); and neither do we know $I.t_s$ and $I.t_f$.

3. APPROXIMATE SNAPSHOTS

3.1 Simple Clock-Free Algorithm

int: array $Value\_Vector[1 \ldots n]$
When event $e = (P_i, val)$ occurs at P_i:
(1) transmit to sink (or broadcast) event notification (P_i, val)
(2) (if broadcasting is being used) $Evaluate\_State(P_i, val)$
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

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

Figure 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). *Evaluate_State* is executed atomically with its invocation. A node tracks the latest sensed value reported by P_z in $Value\_Vector[z]$. For now, assume that a distinguished node (sink) runs *Evaluate_State*

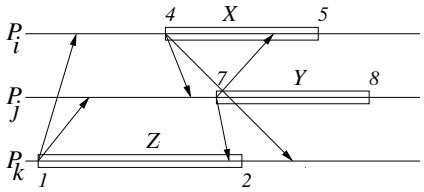

Figure 2: Overlap, a potential false negative.

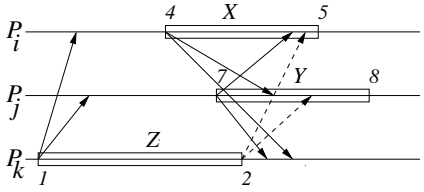

Figure 3: Overlap, a potential false negative.

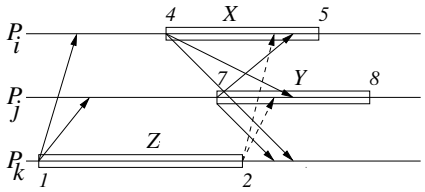

Figure 4: Overlap, an "inevitable" false negative.

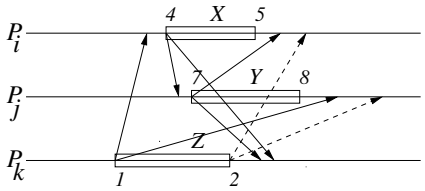

Figure 5: Overlap, a potential false negative.

to evaluate ϕ. However, if broadcasting is used, all nodes can run *Evaluate_State*; we discuss the implications later. 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 Figure 2, 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 this example to show that this detection can be identified as a true positive by the consensus-based algorithm.

In Figure 3, 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 Figure 4, 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 [12].)

In Figure 5, 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 Figure 6, 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 Figure 7, none of the processes detect ϕ, resulting in a true negative. A false positive is not possible.

In Figure 8, P_j detects false a 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 Figure 1, we have:*

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*

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 Figure 1, we have:*

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 don't 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.

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)

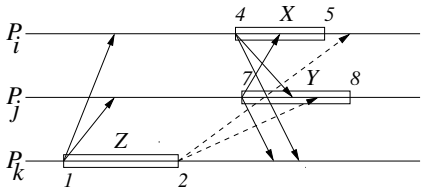

Figure 6: No overlap, a potential false positive.

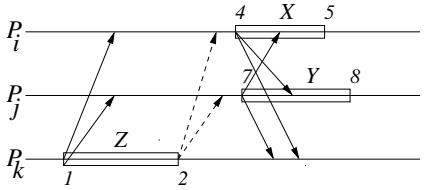

Figure 7: No overlap, false positive not possible.

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. $IV[j] = k$ (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 $IV[j]$th interval at process P_j.

Although the proposed interval vectors are similar to logical vector clocks [13], there are some differences – for example, (i) there is no underlying computation message exchange and all event notifications are control messages, and (ii) on receiving an interval vector, the local component of the interval vector does *not* advance. Variant of the interval vector was used in [4].

The algorithm is given in Figure 9. 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 Figure 2, 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 Figure 3, the use of *Interval_Vector* cannot help P_k in eliminating its false negative conclusion. Similarly, in Figure 4, the use of *Interval_Vector* cannot help any process in eliminating its false negative conclusion. But in Figure 5, the use of *Interval_Vector* allows at least one process, viz., P_j, to eliminate its false negative conclusion.

In Figure 6, the use of *Interval_Vector* cannot help P_i or P_j in eliminating their false positive conclusion. However, in Figure 8, the use of *Interval_Vector* allows P_j reading a false positive to eliminate it.

3.3 Improved Accuracy through Consensus

The algorithms of Figures 1 and 9 have these drawbacks:

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

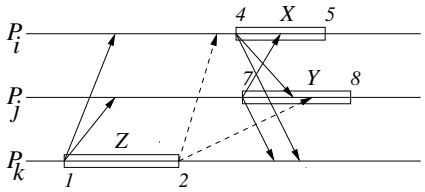

Figure 8: No overlap, a potential false positive.

int: **array** $Interval\_Vector[1 \ldots n]$
int: **array** $Value\_Vector[1 \ldots n]$
boolean: *new*
When event $e = (P_i, val)$ occurs at P_i:
(1) $Interval\_Vector[i]++$
(2) $Value\_Vector[i] \longleftarrow val$
(3) broadcast $(P_i, Interval\_Vector, Value\_Vector)$
(4) $Evaluate\_State(P_i, Interval\_Vector, Value\_Vector)$
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

Figure 9: Interval Vector Algorithm: Code at P_i to detect a predicate using event *vector* notifications.

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 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 the algorithms of Figures 1 and 9, 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 Figure 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, 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 nondeterminism 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 Figure 3, $IV = [4, 7, 1]$ will be detected by P_i and P_j but not P_k; P_k has a false negative.

In Figure 5, $IV = [4, 7, 1]$ will be detected by P_j but not P_i and P_k; P_i and P_k have a false negative.

In Figure 6, $IV = [4, 7, 1]$ 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 increased the entropy or the 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.

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)* to a sink. To determine the number of witnesses of the *same* state IV, the sink collates the received $Consensus_Message$s. 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 Figure 10.

- If $\phi(e)$ is true, $flag$ on the EN broadcast is set. When 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 [9] 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)). If a new (neither seen nor evaluated locally so far) 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

```
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 = (P_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 (P_i, Interval_Vector, Value_Vector, flag)
(7)  Evaluate_State(P_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 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 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 2.1; raise alarm/actuate(true positive,IV)
(12) else
(13)      Corollary 2.2; await more confirmations or timer pop
On P_i getting a timer pop for Interval_Vector_History.x:
(1)  Corollary 2.2; raise alarm/actuate(borderline,IV)
(2)  delete record x from Interval_Vector_History
```

Figure 10: Consensus Algorithm: Code at P_i to detect a predicate using consensus.

the results for "almost free" and will see the same result.

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 2. *For n observers in a system without any synchronized clocks, for the detection algorithm in Figure 10, we have for any IV:*

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*

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 Figures 2-8 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 2. *For n observers in a system without any synchronized clocks, for the detection algorithm in Figure 10, we have for any IV:*

1. *Confirmed by all $\Longleftrightarrow$ positive bin $\Longrightarrow$ true positive $\Longrightarrow$ $overlap \geq 0$*
   Examples: *Figure 2.*
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. For safety, a negative decision can be made.
   Examples: *Figures 3, 5, 6.*
3. *Confirmed by none $\Longleftrightarrow$ negative bin $\Longrightarrow$ (true negative ($\Longrightarrow$ $overlap < 0$) $\oplus$ false negative having $0 < overlap < \Delta$)*
   Examples: *Figures 7, 8; and Figure 4, resp.*

Thus, those states that are confirmed by all the observers (and in which ϕ is true) correspond to Corollary 2.1. Those that are confirmed by only some observers (and in which ϕ is true) correspond to Corollary 2.2. Those that are seen by none of the observers correspond to Corollary 2.3.

The algorithm gives the following advantages:

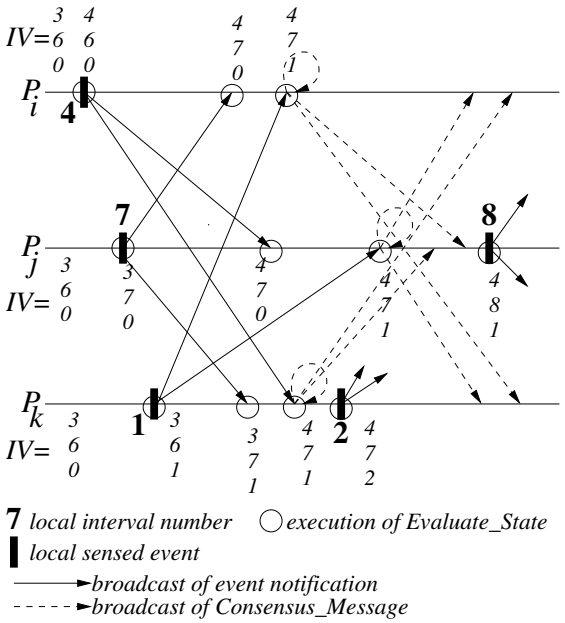

Figure 11: Example of on-the-fly state construction and evaluation.

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 Figure 2, $IV = [4, 7, 1]$ 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 Figure 3, $IV = [4, 7, 1]$ will be confirmed by P_i and P_j but not P_k; in Figure 5, $IV = [4, 7, 1]$ will be confirmed by P_j but not P_i and P_k; in Figure 6, $IV = [4, 7, 1]$ 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: Figure 4 is an unfortunate false negative, but Theorem 2.(2,3,4) shows that $overlap \in (-\infty, \Delta)$ must hold. Figure 7 and 8 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 Figure 11, the global state passes through:

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

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_Vector*s. 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* 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

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 [7], 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*. Smart data structures can be used instead of the list and we can perform garbage collection to reduce this number significantly. In Figure 10, 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

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. 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 false positive; if closer to n, likely a true positive.

Table 2 compares the algorithms, and those in [12]. 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 $O(n)$.

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

Algorithm → Properties ↓	Strobe Vector algorithm [a] [12]	Strobe Scalar algorithm [12]	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 $\phi)]$ at sink	$O(np)$/node + $[O(n^2p)$ + $(O(np)$ eval of $\phi)]$ 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

6. ACKNOWLEDGEMENTS

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

7. REFERENCES

[1] K. Birman, T. Joseph, Reliable communication in the presence of failures. *ACM TOCS*, 5(1), 1987.

[2] 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*, 193-200, 2006.

[3] 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*, 283-289, 2006.

[4] P. Chandra, A. D. Kshemkalyani, Causality-based predicate detection across space and time. *IEEE Transactions on Computers*, 54(11): 1438-1453, 2005.

[5] R. Cooper, K. Marzullo, Consistent detection of global predicates. In *Proc. ACM/ONR Workshop on Parallel and Distributed Debugging*, 163-173, May 1991.

[6] P. Hu, J. Indulska, R. Robinson, An autonomic context management system for pervasive computing. In *IEEE International Conference on Pervasive Computing and Communications (Percom)*, 213-223, 2008.

[7] 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*, 2009.

[8] L. Kaveti, S. Pulluri, G. Singh, Event ordering in pervasive sensor networks. In *IEEE Int. Conf. on Pervasive Computing and Comm. Workshops*, 2009.

[9] A.D. Kshemkalyani, Temporal interactions of intervals in distributed systems. *Journal of Computer and System Sciences*, 52(2): 287-298, April 1996.

[10] A.D. Kshemkalyani, Temporal predicate detection using synchronized clocks. *IEEE Transactions on Computers*, 56(11): 1578-1584, November 2007.

[11] A.D. Kshemkalyani, M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*, Cambridge University Press, 2008.

[12] 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 Press, 2010.

[13] F. Mattern, Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, North-Holland, pp 215-226, 1989.

[14] J. Mayo, P. Kearns, Global predicates in rough real time. In *IEEE Symp. on Parallel and Distributed Processing*, 17-24, 1995.

[15] M. Roman, C.Hess, R. Cerqeira, A. Ranganathan, R.H. Campbell, K. Nahrstedt, A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4): 74-83, 2002.

[16] B. Sundararaman, U. Buy, A. D. Kshemkalyani, Clock synchronization for wireless sensor networks: a survey. *Ad-Hoc Networks*, 3(3): 281-323, May 2005.

[17] 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*, 336-345, 2005.