Appears in the International Workshop on Networked Sensing Systems (INSS) 2004

Approximate Causal Observer[1]

Sandeep S. Kulkarni Mahesh (Umamaheswaran) Arumugam
Software Engineering and Network Systems Lab
Department of Computer Science and Engineering
Michigan State University, East Lansing MI 48824

Abstract

In this paper, we focus on the problem of approximate causal delivery. This problem identifies the tradeoff between causal delivery and timely delivery of messages. Causal delivery requires that delivery of a message, say m, be delayed until all messages on whom m is causally dependent are delivered. By contrast, timely delivery requires that messages be delivered as soon as possible. However, the requirements of causal delivery and timely delivery are conflicting. We show how a simple logical timestamp program can be used to obtain a solution for approximate causal observer. This solution is intended for sensor networks that provide simple guarantees about the clock drift among sensors and about maximum delay of messages that are not lost. Our solution lets the sensors to choose the level of causality violations it can tolerate (0% or more) and the time for which it will have to buffer the received messages. We also show that our solution provides a continuum where the application can choose the size of the timestamps it maintains by identifying the level of causality violations it can tolerate.

Keywords: Causal Delivery, Timeliness, Logical Timestamps, Sensor Networks

1 Introduction

The ability to *observe* distributed computations is one of the important problems in many systems, especially, in sensor networks (e.g., MICA motes [1, 2]). For example, consider an application in sensor networks where a group of sensors need to help pursuers track moving evaders [3]. In such a system, the sensors communicate their observations about the evader(s) they are tracking with pursuer(s). The pursuers in turn communicate with the base station or an observer. The base station is more powerful (e.g, PC) and is responsible for providing visualization. Thus, the base station *observes* the communication among sensors and uses its high computing power/memory to display/interpret the communication among sensors.

The order in which the observer receives messages may be different from the order in which the communication occurred in the network, due to message collisions/corruptions, routing, etc. Hence, the observer needs to reorder messages consistently. One way to obtain such consistent view is to ensure causal order delivery at the observer. Another requirement for such observers is that they should be *timely*, i.e., the observer needs to reconstruct the underlying computation quickly. For example, in the sensor network application discussed above, the visualization of the underlying computation should be in real-time and, hence, the visualization should not significantly lag behind the original computation.

Since causal delivery and timeliness are conflicting goals, it is necessary to develop protocols where the observer can choose the level of causality violations it can accept to ensure timely delivery of messages. We call this the problem of *approximate causal delivery*. And, an observer that provides approximate causal delivery is called *approximate causal observer*.

Contributions of the paper. We focus on adapting the algorithm in [4] for obtaining approximate causal delivery. The main contributions of this paper are as follows:

- We present two algorithms for approximate delivery. Based on the ability to tolerate causality violations, the first algorithm allows the observer to reduce the delivery time of messages. In the second algorithm, the system checks the message queue before delivery to determine if it might violate the requirements of

[1]Email {sandeep,arumugam}@cse.msu.edu. Tel: +1-517-355-2387. Fax: +1-517-432-1061. This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF Equipment Grant EIA-0130724, and a grant from Michigan State University

causal delivery, and hence, we reduce the causality violations further.

- We study the effect of changing various parameters (maximum clock drift among sensors, maximum message delay, and message rate) on approximate causal delivery through simulations.
- We show that the timestamp provides a continuum, i.e., the application developer can choose the size of timestamp, as small as 4 bytes, considering the number of causality violations the application can handle.

Organization of the paper. The rest of the paper is organized as follows. In Section 2, we present the algorithm for causal delivery from [4]. In Section 3, we present our approaches for approximate causal delivery. In Sections 4 and 5, we present our simulation model and the results. Finally, in Section 6, we make concluding remarks.

2 Logical Timestamps and Causal Delivery

Based on the result from [5], the system must provide some guarantees that will enable the observer to obtain a tradeoff between causal delivery and timely delivery. We focus on the following two guarantees about the bound on maximum clock drift (for example, using [6]) among different processes (sensors) and the bound on message delay.

Guarantees of the distributed system (sensor network).

$G1$. Physical clock at process j, $rt.j$, is non-decreasing, and at any time, the difference between the clock values of any two processes is bounded by ϵ. In other words,
$$\forall j, k : |rt.j - rt.k| \leq \epsilon$$

$G2$. Let m_j be a message sent by process j to k. Also, let st_m denote the clock value of j when j sent m_j, and let rd_m denote the clock value of j when k received m_j. We require that k should receive m_j within time δ unless m_j is lost. In other words,
$$((rd_m \leq (st_m + \delta)) \quad \vee \quad rd_m = \infty)$$

Before presenting the programs, we define the notion of *happened-before*, $\longrightarrow$, among events.

Happened-before. The happened-before relation [7] is the smallest transitive relation that satisfies, for any events e, f, $e \longrightarrow f$ if (1) e and f are events on the same process and e occurred before f, or (2) e is a send event in one process and f is the corresponding receive event in another process. □

Logical timestamp program. In the solution to the logical timestamps proposed in [4], the timestamp of an event e_j at process j is of the form $\langle rt.e_j, c.e_j, kn.e_j \rangle$, where $rt.e_j$ denotes the physical clock value of j when e_j was created. The variable $c.e_j$ denotes the difference between the knowledge j had about the maximum clock value in the system and the physical clock value of j. The variable $kn.e_j$ is an array of size 2ϵ. The variable $kn.e_j[t]$, $-\epsilon \leq t < \epsilon$, captures the knowledge about the number of events f such that $r.f = r.e_j + t$ and $f \longrightarrow e$. The program for logical timestamps is given in Figure 1.

Initially:
$rt.j, r.j, c.j = 0$,
$\forall t : t \neq 0 : kn.j[t] = 0, kn.j[0] = 1$
Local event e_j or send event e_j (message being sent is m_j)
$c.j := max(0, r.j + c.j - rt.j)$
$\forall t : -\epsilon \leq t < \epsilon : kn.j[t] := kn.j[t + rt.j - r.j]$
$kn.j[0] := kn.j[0] + 1$
$r.j := rt.j$
$r.e_j, c.e_j, kn.e_j := r.j, c.j, kn.j$
if e_j is a send event then
$r.m_j, c.m_j, kn.m_j := r.j, c.j, kn.j$
Receive event e_j (message m received with timestamp $\langle r.m, c.m, kn.m \rangle$)
$c.j := max(0, r.j + c.j - rt.j, r.m + c.m - rt.j)$
$\forall\, t : -\epsilon \leq t < \epsilon : kn.j[t] :=$
$max(0, kn.j[t + rt.j - r.j], kn.m[t + rt.j - r.m])$
$kn.j[0] := kn.j[0] + 1$
$r.j := rt.j$
$r.e_j, c.e_j, kn.e_j := r.j, c.j, kn.j$

Figure 1. Logical timestamp program at process j

To compare two logical timestamps, $\langle r.e_j, c.e_j, kn.e_j \rangle$ and $\langle r.f_k, c.f_k, kn.f_k \rangle$, we use the following $less$ relation.

$$less(\langle r.e_j, c.e_j, kn.e_j \rangle, \langle r.f_k, c.f_k, kn.f_k \rangle) \quad \text{iff}$$

$$(r.e_j + c.e_j, kn.e_j[c.e_j], kn.e_j[c.e_j - 1], \ldots, kn.e_j[c.e_j - \epsilon + 1], j)$$
$$< \text{// lexicographic comparison}$$
$$(r.f_k + c.f_k, kn.f_k[c.f_k], kn.f_k[c.f_k - 1], \ldots, kn.f_k[c.f_k - \epsilon + 1], k)$$

Causal delivery program. The causal delivery program proposed in [4] is as follows: Whenever a process, say j, receives a message m, j buffers the message until $delcond(m, j) = (rt.j = r.m + c.m + \delta + \epsilon)$ is satisfied. As soon as the $delcond(m, j)$ is satisfied, the message is delivered. If two or more messages satisfy the delivery condition simultaneously then process j uses the $less$ relation to determine the causal relation among the messages and delivers them accordingly. We refer the reader to [4] for the correctness of these programs.

3 Approximate Causal Delivery

The algorithm presented in Section 2 uses the delivery condition $delcond(m, j) = (rt.j = r.m + c.m + \delta + \epsilon)$ to deliver a message m to process (sensor) j. This condition is necessary for correctness, i.e., to ensure *all* messages are delivered in causal order. Thus, $c.m + \delta + \epsilon$ is the approximate delay in obtaining causal delivery. We consider the case where messages are delivered before this delivery condition is satisfied. Based on this approach for reducing the delay, we present two algorithms for approximate causal delivery: (1) deliver after partial wait and (2) check before delivery.

Deliver After Partial Wait (DAPW). In this algorithm, we use the following delivery condition: $delcond(m, j) = (rt.j = r.m + ¢(c.m + \delta + \epsilon))$, where $0\% < ¢ < 100\%$. Thus, $¢ = 0\%$ means that the messages are delivered to the observer when the clock of observer is at least $r.m$, or as soon as the message arrives at the observer, whichever is later. And, $¢ = 100\%$ means that the messages are delivered in perfect causal order.

Check Before Delivery (CBD). In DAPW, whenever a message, say m_1 is about to be delivered to process j, if there is a casually related message m_2 such that $send(m_2) \longrightarrow send(m_1)$ is true and m_2 is scheduled for delivery at a later time than m_1 then a causality violation is inevitable. Hence, in the second algorithm, whenever message m_1 is about to be delivered at process j, j checks the message queue to determine if there is any message m_2 such that $less(\langle r.m_2, c.m_2, kn.m_2\rangle, \langle r.m_1, c.m_1, kn.m_1\rangle)$ is $true$. If there exists such a message m_2 then j sets the delivery time of m_1 as $delcond(m_1, j) = delcond(m_2, j)$. If there are no such messages then m_1 is delivered based on the DAPW algorithm.

4 Simulation Model

Our simulation model consists of n sensors and a base station (observer). The sensors communicate with each other. Every message sent by a sensor is also sent to the base station (for visualization). Messages are buffered at the base station and delivered such that the number of causality violations is acceptable. We do not assume that the base station can precisely determine causal relations between two messages. Now, we show how our simulation model ensures system properties stated in Section 2.

At each step of simulation, some random sensor chooses to advance its clock in such a way that $G1$ is not violated. Whenever a sensor advances its clock, it sends a message to randomly selected subset of sensors, with a probability of message rate. And, a message, say m, can remain in transit for a random duration as long as $G2$ is not violated.

5 Simulation Results

We developed an event simulation program in Java. In this program, we do not associate units for maximum clock drift (ϵ) and maximum message delay (δ). Furthermore, we find that the ratio $\frac{\epsilon}{\delta}$ is important than the individual parameters. The results presented here are the mean of three simulations. For a given value of the input parameters, the percentage of causality violations in different simulations are similar. The raw data of these simulations is available at [8].

The number of causality violations at the observer is computed as follows: For each m_1, we compute the number of messages delivered before m_1 (say, m_2) such that $send(m_1) \longrightarrow send(m_2)$. We say that these messages violate backward causality. Likewise, for each m_1, we compute the number of messages delivered after m_1 (say, m_2) such that $send(m_2) \longrightarrow send(m_1)$. We say that these messages violate forward causality. The number of causality violations is the average of messages that violate backward/forward causality.

To compute these causality violations, we also maintain vector timestamps [9, 10] in addition to the logical timestamps from Section 2. These vector timestamps identify the actual causal relation among events in the system and are not used in any way to determine when messages are delivered.

Now, we present the effect of ϵ, δ and message rate on causal delivery of messages. Unless otherwise specified, these simulations include 10 sensors and one base station (observer), and message rate is 0.1. For different parameters, we refer the reader to [8].

5.1 Effect of Maximum Clock Drift (ϵ)

The effect of ϵ on causal delivery of messages using DAPW and CBD is shown in Figure 2. The graphs show the number of causality violations as a function of the percentage of delay, ¢, used in $delcond$. In these experiments, we use $\delta = 10$.

DAPW. When the ratio $\frac{\epsilon}{\delta}$ is larger, the number of causally dependent messages for a given message m is large. Hence, there is a higher probability for causality violations. Therefore, the number of causality violations increase (cf. Figures 2 (a) and 2 (c)).

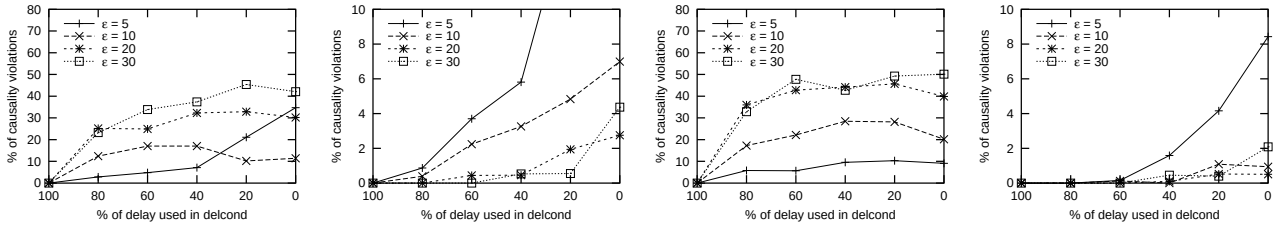

Figure 2. Effect of ϵ on (a) DAPW with delay $N(\frac{\delta}{2}, \frac{\delta}{4})$, (b) CBD with delay $N(\frac{\delta}{2}, \frac{\delta}{4})$, (c) DAPW with delay $N(\frac{\delta}{4}, \frac{\delta}{8})$, and (d) CBD with delay $N(\frac{\delta}{4}, \frac{\delta}{8})$. (Scale of DAPW and CBD graphs are different.)

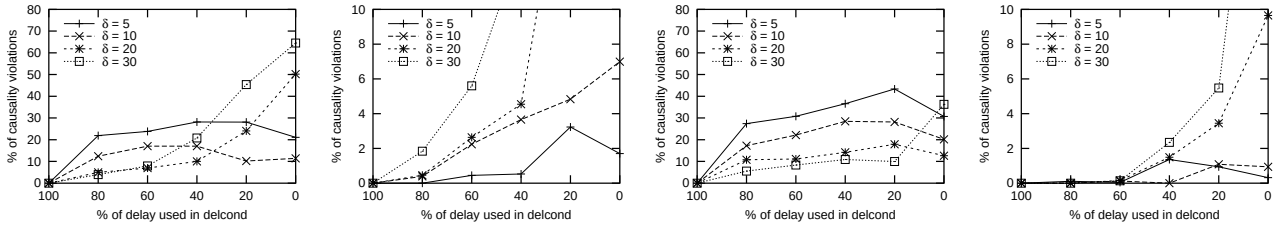

Figure 3. Effect of δ on (a) DAPW with delay $N(\frac{\delta}{2}, \frac{\delta}{4})$, (b) CBD with delay $N(\frac{\delta}{2}, \frac{\delta}{4})$, (c) DAPW delay $N(\frac{\delta}{4}, \frac{\delta}{8})$, and (d) CBD message delay $N(\frac{\delta}{4}, \frac{\delta}{8})$. (Scale of DAPW and CBD graphs are different.)

When message delay is determined from the distribution $N(\frac{\delta}{4}, \frac{\delta}{8})$, 95% of the messages are received within $\frac{\delta}{2}$. By contrast, with $N(\frac{\delta}{2}, \frac{\delta}{4})$, 95% of the messages are received in δ. Thus for $N(\frac{\delta}{4}, \frac{\delta}{8})$, the probability of causality violations is less since number of messages that causally depend on a given message is more than that for $N(\frac{\delta}{2}, \frac{\delta}{4})$ (cf. Figures 2 (a) and 2 (c)).

For small values of $\frac{\epsilon}{\delta}$, there exists a threshold T such that, the causality violations increase suddenly when ¢ $< T$. (For example, in Figure 2(a), for $\epsilon = 5$, causality violations increase when ¢ $< 40\%$.) When $T \leq$ ¢ $< 100\%$, the delay in delivery captures most of the causal relation among messages. When this delay is reduced (i.e., ¢ $> T$), the messages are delivered faster and, hence, the causal relation among messages is not captured. For larger values of $\frac{\epsilon}{\delta}$, the observer captures most causally related messages even when the delay in delivery for less.

CBD. As $\frac{\epsilon}{\delta}$ ratio increases, causality violations decrease (cf. Figures 2(b) and 2(d)), which is exactly opposite to DAPW. CBD postpones the delivery of m_1 if the message queue contains causally preceding messages of m_1. Thus, as $\frac{\epsilon}{\delta}$ ratio increases, CBD can detect/prevent most of the causality violations since the probability that the message queue contains one or more causally related messages is high.

Further, we note that the number of causality violations is less when CBD is used with message delay of $N(\frac{\delta}{4}, \frac{\delta}{8})$. This is due to the fact that there is a high probability that at least one causally related message for a given message m will be present in the message queue of the observer when m is about to be delivered.

Comparison. From Figure 2, we conclude that the number of causality violations in CBD are an order of magnitude less than that in DAPW. For small values of $\frac{\epsilon}{\delta}$, CBD performs almost similar to DAPW, since the number of causally related messages is less. However, for large values of $\frac{\epsilon}{\delta}$, CBD outperforms DAPW.

5.2 Effect of Maximum Message Delay (δ)

The effect of δ on causal delivery of messages using DAPW and CBD is shown in Figure 3. In these experiments, we use $\epsilon = 10$. The results are similar to that in Section 5.1.

As $\frac{\epsilon}{\delta}$ increases, the number of causality violations increase (cf. Figures 3(a) and 3(c)). Further, in Figure 3 (c), we ob-

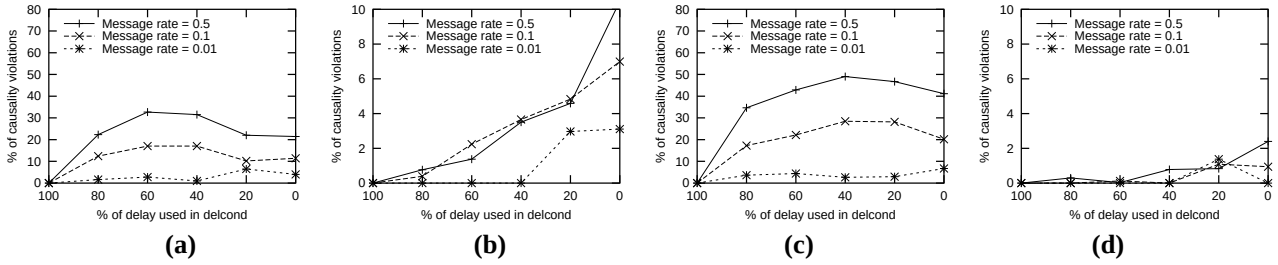

Figure 4. Effect of message rate on (a) DAPW with delay $N(\frac{\delta}{2}, \frac{\delta}{4})$, (b) CBD with delay $N(\frac{\delta}{2}, \frac{\delta}{4})$, (c) DAPW with delay $N(\frac{\delta}{4}, \frac{\delta}{8})$, and (d) CBD with delay $N(\frac{\delta}{4}, \frac{\delta}{8})$. (Scale of DAPW and CBD graphs are different.)

serve that the number of causality violations is more when DAPW is used with delay of $N(\frac{\delta}{4}, \frac{\delta}{8})$. And, for small values of $\frac{\epsilon}{\delta}$, there exists a threshold T such that causality violations increase suddenly when ¢ $< T$. (For example, in Figure 3 (a), for $\delta = 20$ when ¢ $< 60\%$.) Finally, we can observe that CBD results (cf. Figures 3(b) and 3(d)) are exactly opposite to DAPW (cf. Figure 3 (a) and 3 (c)).

Remark. We note that when the average message delay increases, the number of causality violations also increase. This is due to the fact that most messages arrive late and, hence, the message queue does not have sufficient information to prevent increased causality violations.

5.3 Effect of Message Rate

The effect of message rate on causal delivery of messages using DAPW and CBD is shown in Figure 4. In these experiments, we use $\epsilon = \delta = 10$.

As the message rate increases, more causally dependent messages for a message m are present in the system. Hence, the number of causality violations in CBD is significantly less than that in DAPW.

With message delay of $N(\frac{\delta}{4}, \frac{\delta}{8})$, the number of causality violations in CBD is in the order of $0\% - 2\%$ (cf. Figure 4 (d)). This is due to the fact most messages arrive within $\frac{\delta}{2}$, and, hence, CBD has more information in the queue to detect/prevent causality violations.

Further, for small values of message rate, causality violations in DAPW and CBD are nearly equal. This is due to the fact that at low message rates, CBD has very limited information to exploit among the messages in the queue.

5.4 Physical Clocks Vs. Partial Timestamps

In this section, we argue that the information maintained in CBD, although small, is important in reducing the number of causality violations. Towards this end, we compute the causality violations for the case where only physical clock is used to determine when a message should be delivered. To obtain an implementation that uses physical clock alone, we set the c value and all elements in kn to 0. We call this algorithm DPC1. We also consider the algorithm DPC2 where the c value is used but kn values are reset to 0. Other points on this continuum can be obtained by maintaining a subset of the kn values in the timestamp.

Notation. In this section, by *"2 $kn.e$ elements"* we mean that the simulation uses $kn.e_j[c.e_j]$ and $kn.e_j[c.e_j{-}1]$ elements instead of the $kn.e_j$ array for an event e_j. Similarly, by *"k $kn.e$ elements"* we mean that the simulation uses the first k $kn.e$ elements.

Figure 5 shows the simulation results for $\epsilon = \delta = 10$, message rate $= 0.1$ and 10 sensors. (We refer the reader to [8] for results with different parameters.)

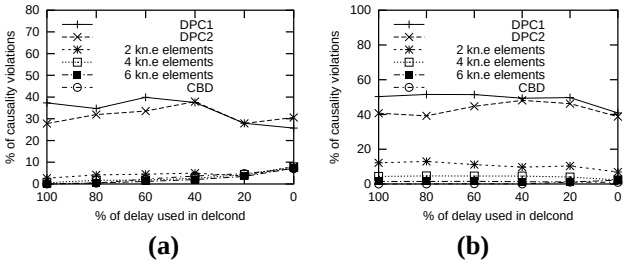

Figure 5. Effect of using partial timestamps on (a) CBD with delay $N(\frac{\delta}{2}, \frac{\delta}{4})$, (b) CBD with delay $N(\frac{\delta}{4}, \frac{\delta}{8})$.

From Figure 5, we observe that using physical clocks

alone for causal delivery of messages is not enough. Specifically, even when ¢ = 100%, DPC1 and DPC2 have around $30\% - 50\%$ of causality violations. And, maintaining just 2 $kn.e$ elements provides a significant reduction in number of causality violations ($10 - 15\%$). Moreover, if we increase the number of $kn.e$ elements in the timestamp, the causality violations can be further reduced. Maintaining just 6 $kn.e$ elements gives the same result as CBD. Thus, the timestamp provides a continuum in which the application developer can choose the size of the timestamps based on the requirements. This result is especially important in sensor networks. Specifically, in MICA motes [2], the payload size is just 29 bytes. Hence, the overhead in achieving approximate causal delivery should be small. Depending on the percentage of causality violations sensors can handle and the overhead involved, the developer can choose an appropriate size for the timestamp. For example, choosing 2 $kn.e$ elements (i.e., 4 bytes including $rt.j$ and $c.j$) will result in $10 - 15\%$ causality violations (as opposed to $30 - 50\%$ causality violations when using the physical clocks alone).

6 Conclusion

In this paper, we presented a solution for approximate causal delivery. We discussed the effect of the parameters such as maximum clock drift, maximum message delay, and message rate on causal delivery of messages. We showed that by using physical clocks alone, the number of causality violations increase significantly. By adding new variables to the timestamp, the number of causality violations can be reduced. In other words, we showed that our solution provides a continuum such that the application developer can choose the size of timestamps used in the system based on the number of causality violations the application can tolerate. This result is especially useful in sensor networks, since the sensors are resource constrained and the size of the payload in a message is very limited (e.g., 29 bytes in MICA). From Section 5.4, we note that maintaining just 2 $kn.e$ elements (i.e., 4 bytes) provides a significant reduction in causality violations ($10 - 15\%$) compared to using physical clocks alone ($30 - 50\%$). Hence, causal delivery of messages at the base station can be achieved easily in sensor networks with a small message overhead. To our knowledge, this result is the first of its kind for providing approximate causal delivery in sensor networks. Moreover, DAPW and CBD preserve the self-stabilization [11] property of the algorithm in [4], i.e., starting from arbitrary initial states, the system recovers to states from where causal delivery is achieved. Hence, if the sensors are corrupted, our algorithm ensures that eventually approximate causal delivery is restored.

References

[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *In Proceedings of the International Conference on Architectural Suppport for Programming Languages and Operating Systems (ASPLOS)*, November 2000.

[2] J. Hill and D. E. Culler. Mica: A wirleess platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, 2002.

[3] M. Demirbas, A. Arora, and M. G. Gouda. A pursuer-evader game for sensor networks. *In Proceedings of the Sixth Symposium on Self-Stabilizing Systems (SSS), Springer*, LNCS: 2704:1–16, June 2003.

[4] S. S. Kulkarni and Ravikant. Stabilizing causal deterministic merge. *In Proceedings of the Fifth International Workshop on Self-Stabilizing Systems, Springer*, LNCS:2194:183–199, October 2001.

[5] M. J. Fischer, N. A. Lynch, and M. S. Peterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, 1985.

[6] T. Herman. NestArch: Prototype time synchronization service. NEST Challenge Architecture. Available at: `http://www.ai.mit.edu/people/sombrero/nestwiki/index/ComponentTimeSync`, January 2003.

[7] L. Lamport. Time, clocks, and the ordering of events in a disributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[8] Simulation results and raw-data for approximate causal delivery. Available at: `http://www.cse.msu.edu/~arumugam/research/results/approxcd`.

[9] J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Proceedings of the 11th Australian Computer Science Conference*, 10(1):56–66, Feb 1988.

[10] F. Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[11] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11), 1974.