TEMPORAL MESSAGE ORDERING IN WIRELESS SENSOR NETWORKS[1]

Kay Römer

Institute for Pervasive Computing
ETH Zurich, Switzerland
email: roemer@inf.ethz.ch

ABSTRACT

Wireless sensor networks (WSN) are envisioned to fulfill complex monitoring tasks in the near future. A typical WSN application like object tracking fuses sensor readings produced by nodes throughout the network to obtain a high-level sensing result such as the current speed of a tracked vehicle. In order to produce correct results, the application typically has to process sensor events in the order of their occurrence at the sensor nodes. Temporal message ordering ensures that sensor events arrive at the application in this order. Due to the requirements and characteristics of sensor networks, temporal message ordering is a non-trivial problem in this environment. We motivate the need for temporal message ordering in WSN and present an energy efficient temporal message ordering scheme for sensor networks.

1 INTRODUCTION

Wireless sensor networks (WSN) [1] are currently an active field of research. A WSN consists of large numbers of cooperating small-scale nodes capable of limited computation, wireless communication, and sensing. In a wide variety of application areas including geophysical monitoring, precision agriculture, habitat monitoring, transportation, military systems, and business processes, WSNs are envisioned to fulfill complex monitoring tasks.

While individual sensor nodes have only limited functionality, the global behavior of the WSN can be quite complex. This is achieved, in part, through *data fusion*, the process of correlating individual sensor readings originating from various nodes into a high-level sensing result. Data fusion often depends on the time of occurrence of fused sensor readings. Consider for example a tracking sensor network, where size, shape, direction, location, or velocity of a tracked phenomenon (e.g., object, creature, gas cloud) is determined by fusing time-stamped proximity detections from sensors at different locations.

[1]The work presented in this paper was supported in part by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

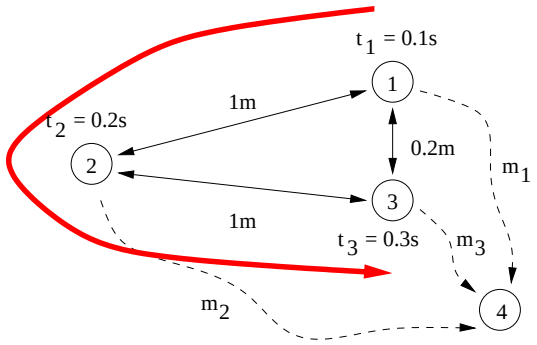

Figure 1: A sensor network for estimating the velocity of moving objects. The thick line indicates the path of the moving object.

Note that such applications require a common physical time scale which is shared by the nodes of the sensor network. Recently, first principles and algorithms for physical time synchronization in sensor networks have been published [3, 4, 23]. Additionally, data fusion algorithms often must process sensor readings in the order of their occurrence time at the sensing nodes, which we call *temporal order*. Consider the following two examples.

Example 1: Object Tracking

Consider a sensor network which is used to estimate the velocity of a moving object. Figure 1 depicts the path of a moving object (thick line) and three nodes of such a network that detect the moving object at times $t_1 < t_2 < t_3$ in their proximity and send off notification messages m_1, m_2, m_3 to node four. The messages contain time ($m_i.t$) and location ($m_i.l$) of the object detection (the object's location is approximated by

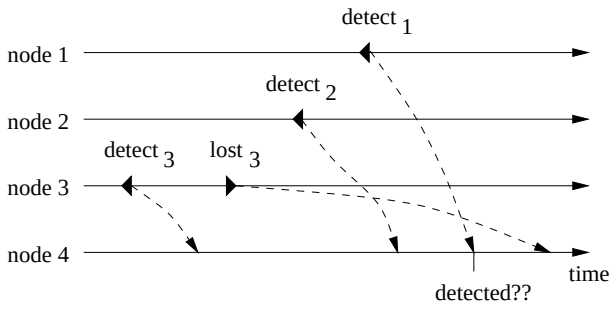

Figure 2: Node-time diagram of a sensor network for reliable object detection.

the location of the detecting node). We will call such tagged messages *sensor events*. Node four fuses these sensor events into a series of velocity estimates of the tracked object, e.g. $\frac{|m_2.l-m_1.l|}{m_2.t-m_1.t} = \frac{1m}{0.1s} = 10\frac{m}{s}$ using sensor events m_1 and m_2.

Note that good estimates are only obtained if the sensor events are processed in temporal order. If, for example, node four processes sensor events in the order of their arrival and m_2 arrives after m_1 and m_3 due to network delays, the bad velocity estimate $\frac{|m_3.l-m_1.l|}{m_3.t-m_1.t} = \frac{0.2m}{0.2s} = 1\frac{m}{s}$ is obtained (using sensor events m_1 and m_3). □

Example 2: Composite Predicate Detection

Consider a sensor network for detecting the presence of certain objects. Assume that nodes emit *detect* events when first detecting the object, and *lost* events when they no longer detect the object. Similar to example 1 and Figure 1, three sensor nodes send such sensor events (containing time t and their location l at the time of occurrence of the event) to node four. In order to reduce the frequency of false detections due to noisy sensor readings, an object detection is only reported by node four if at least three sensor nodes "see" the object concurrently. Figure 2 depicts a sample node-time diagram. Note that there is no point in time at which all nodes see the object concurrently, since lost_3 happens before both detect_1 and detect_2. However, after arrival of detect_1, detect_2, and detect_3 (messages exchanged depicted by dashed arrows), node four is tempted to falsely report an object detection, because lost_3 is delayed in the network. □

In the above examples, *temporal message ordering* can be used to ensure that sensor events arrive at node four in temporal order, such that the data processing algorithms can process sensor events in the order of arrival without obtaining bad results. As the above examples indicate, temporal message ordering is an important component of many sensor network applications performing time-dependent data fusion.

In the remainder of the paper we show that temporal message ordering is a non-trivial problem in sensor networks. Despite this, to our knowledge the problem has not been studied in the context of sensor networks. We therefore present and evaluate a temporal message ordering scheme for sensor networks.

2 TEMPORAL MESSAGE ORDERING IN WIRELESS SENSOR NETWORKS

In this section we want to examine why temporal message ordering in sensor networks is an issue in practice. This discussion will be followed by a list of requirements on a temporal message ordering scheme for wireless sensor networks.

2.1 The Need for Temporal Message Ordering

Note that messages can arrive out of temporal order at a receiver only if a sensor event is passed by a later sensor event in the network due to variable network delays. Consider for example Figure 1, where sensor event m_3 can only arrive before m_2 at node four, if

$$t_3 - t_2 < d_2 - d_3 \quad (1)$$

holds, where d_i is the delay for sending sensor event m_i from node i to node four. If the tracked object has a velocity of $10\frac{m}{s}$, $t_3 - t_2$ is as small as 100ms for the situation depicted in Figure 1. We will now show that $d_2 - d_3$ can be significantly larger than 100ms.

Message delays can be rather high in sensor networks due to their typically limited communication capacity which is shared by nodes within communication range of each other. The RENE motes developed at UC Berkeley, for example, operate at 19.2kbit/s [8]. In a network of such sensor nodes, the single-hop delay for a 50 byte message is about 20ms in an otherwise idle network. However, in dense sensor networks both the sensing ranges and the communication ranges of multiple nodes overlap. That is, many nodes will see tracked objects at the same time from different points of view. This somewhat redundant information can be used to obtain very accurate traces. However, when the nodes want to send off the respective sensor events almost concurrently, they have to compete for the shared communication medium. Assuming that ten nodes see the object, the first of the ten sensor events will have an effective single-hop delay of 20ms while the last sensor event will have an effective single-hop delay of 200ms in case of perfect network allocation. Typical CSMA-based MAC protocols introduce additional variable backoff delays for avoiding/handling collisions [31].

If sensor events have to be sent along multiple hops in a network, the end-to-end delay increases with the number of hops. In a busy network tracking many objects, the multi-hop end-to-end delay can be estimated by multiplying the single-hop delay with the number of hops. Using the above numbers, the effective end-to-end delay d can be estimated as $200ms \leq$

$d \leq 2s$ for a ten hop path. In energy-efficient MAC protocols the radio is switched off periodically, which causes additional variable delays ranging from zero to hundreds of milliseconds or even seconds [31].

Additionally, other kinds of network dynamics (e.g., nodes dying due to exhausted batteries, nodes being destroyed due to environmental influences, temporary environmental obstructions, nodes being added to the network) lead to frequent network reconfigurations. Such reconfigurations often require lengthy procedures like neighbor and route discovery, which additionally increase the maximum message delay in wireless sensor networks. Discovery of new neighbors, for example, can take as long as five seconds [14] in radio systems using frequency hopping such as Bluetooth. Note that that multiple such dynamic reconfigurations might be necessary for the delivery of a single message along multiple hops.

This leads to a situation, where the minimum message delay is quite small (hundreds of milliseconds) and the maximum message delay is rather large (seconds or even tens of seconds). That is, the right hand side of inequality 1 can also be rather large (seconds or tens of seconds). Even with $d_2 = 2$s and $d_3 = 200$ms, the right hand side of inequality 1 evaluates to 1.8s, which is indeed much greater than $t_3 - t_2 = 100$ms. Following the initial argumentation, this indicates that there is a real need for temporal message ordering especially in large, dense sensor networks.

2.2 Requirements

Now that we have pointed out the basic need for temporal message ordering, we want to examine requirements on temporal message ordering schemes for sensor networks.

Sensor nodes are small-scale devices; in the near future, low-cost platforms with volumes approaching several cubic millimeters are expected to be available [13, 29]. Such small devices are very limited in the amount of energy they can store or harvest from the environment. Thus, *energy efficiency* is a major concern in a WSN. In addition, many thousands of sensors may have to be deployed for a given task—an individual sensor's small effective range relative to a large area of interest makes this a requirement, and its small form factor and low cost makes this possible. Therefore, *scalability* is another critical factor in the design of the system.

As mentioned above, sensor networks are subject to frequent partial failures such as exhausted batteries, nodes destroyed due to environmental influences, or communication failures due to obstacles in the environment. The overall operation of the sensor network should be *robust* despite such partial failures.

In many applications, the sensor network will actually be part of a control loop, where obtained sensing results are used to trigger an immediate action. A surveillance sensor network, for example, might contain nodes equipped with simple motion detection sensors and CCD video cameras. Only when the motion detection sensors detect a potential intruder, video cameras along the path of the intruder are turned on, since running all the video cameras all of the time would be very costly (in terms of energy consumption, data processing, and data communication overheads). Another example are emergency detection sensor networks in facilities dealing with toxic substances, where machinery has to be shut down and sluices have to be locked immediately upon detection of a dangerous situation.

Such sensor network applications typically require an action (e.g., turning on video cameras, shutting down machinery) to be triggered as soon as possible after the occurrence of the triggering phenomenon (e.g., entering intruder, leaking gas), a requirement we call *immediacy*.

3 RELATED WORK

In this section we review related work, mostly in the context of traditional computer networks.

3.1 Physical Time and Temporal Message Ordering

Temporal message ordering requires that all participating sensor nodes share a common physical time base. Due to the requirements mentioned in Section 2.2, physical time synchronization is a non-trivial problem in sensor networks. In general, approaches developed for traditional networks are not suitable for sensor networks [3]. First approaches for physical time synchronization in wireless sensor networks are described in [4, 23].

Temporal message ordering has also been an issue in traditional computer networks, for example in the context of system monitoring and distributed event systems in local area networks.

Delaying techniques such as [17, 19, 26] assume that their is a small upper bound D on the message delay in the network, i.e., it takes at most time D to send a message from any node in the network to any other node. The receiver which wants to receive messages in temporal order, maintains a list of messages m sorted by their time stamps $m.t$ in ascending order. When a new message m arrives at the receiver, it is inserted into this list at the right position. The first element m_0 of the list is removed and passed to the application as soon as the receivers clock shows a value greater than $m_0.t + D$. By this time, all messages with time stamps smaller than $m_0.t$ have been received and inserted into the list due to the upper bound D on the message delay.

While this approach looks attractive at first glance because it does not require any extra message exchanges for achieving temporal message ordering, it does not fulfill the requirements pointed out in Section 2.2. As we have shown in Section 2.1, there is no small upper bound on the network delay

in sensor networks. Using delaying techniques with a value of D smaller than the maximum network delay will cause messages to arrive out of temporal order. Using a large value of D precludes immediacy, since evaluation of all messages is artificially delayed by D at the receiver.

Heartbeat protocols such as [6] assume FIFO channels between all nodes in the network. As with the delaying techniques, the receiver maintains an ordered list of messages. The first message m_0 is removed from the list and delivered to the application, as soon as the receiver has received a messages m_i with $m_i.t > m_0.t$ from each node i in the network. The FIFO property ensures, that the receiver has received all messages with time stamps earlier than $m_0.t$ at this point in time. In order to prevent starvation, all nodes send time-stamped heartbeat messages to the receiver at regular intervals Δ.

This scheme also does not fulfill the requirements pointed out in Section 2.2. A small value of Δ results in a large message overhead, since each node will send a heartbeat message after Δ, which is neither scalable nor energy efficient. A large value of Δ precludes immediacy, since evaluation of all messages is artificially delayed by Δ at the receiver in the worst case. Also note that this scheme sends heartbeat messages even if no (real) messages have to be delivered at all, resulting in a waste of energy.

3.2 Logical Time and Causal Message Ordering

Physical time defines a total order on all time stamps, i.e., a physical time stamp t_1 is either less than ($<$), greater than ($>$), or equal ($=$) to a second physical time stamp t_2. Moreover, physical time provides a measure for the elapsed amount of real-time, i.e., $|t_2 - t_1|$ is a measure for the amount of real-time elapsed between t_1 and t_2.

In contrast, *logical time* [16, 18] defines only a partial order, i.e., a logical time stamp t_1 either happened before ($\rightarrow$), happened after ($\leftarrow$), or is unrelated ($\|$) to a second logical time stamp t_2. Moreover, logical time does not provide a measure for elapsed real-time, i.e., the operation $t_2 - t_1$ is not defined for logical time stamps.

In sensor networks, where sensor events are triggered by real-world phenomena, we want to be able to relate any two sensor events (and the respective time stamps). Moreover, we want to measure the amount of real-time elapsed between two sensor events like in the example in Section 1. Therefore, physical time must be used to relate events in the physical world. Logical time is not sufficient in the WSN domain.

Causal message ordering is somewhat similar to temporal message ordering. It is based on logical time and ensures that message m_1 is delivered before m_2, if the logical time stamp $m_1.t$ happened before $m_2.t$, i.e., $m_1.t \rightarrow m_2.t$. However, m_1 and m_2 can be delivered in any order if $m_1.t \parallel m_2.t$.

Causal message ordering has been an active research topic in many areas such as distributed databases, realtime systems, and fault tolerant systems. Examples include solutions for distributed systems in general [15, 25], for mobile computing systems [22, 27], and as part of total ordering multicast protocols with support for causal order delivery of messages from multiple sources [2, 5, 10, 20]. However, causal message ordering is not sufficient in wireless sensor networks, since it is based on logical time.

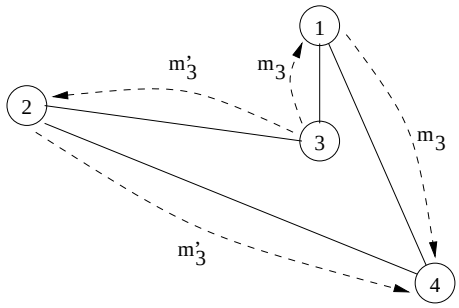

Figure 3: TMOS basic operation

4 THE TEMPORAL MESSAGE ORDERING SCHEME

This section describes the TMOS mechanism for temporal message ordering in detail. TMOS stands for **T**emporal **M**essage **O**rdering in **S**ensor networks.

We will explain the basic idea behind TMOS using the example from Section 1. Let us assume that a communication channel between any pair of nodes has the FIFO property, i.e., the messages will arrive at the receiver in the same order the sender sent them.

Figure 3 shows the four nodes from the example in Section 1. Nodes 1-4 are arranged on a logical ring as indicated by the solid lines in the figure. In order to deliver sensor event m_3, node three sends two copies (m_3 and m_3') along this ring toward node four. m_3 is sent clockwise, while m_3' is sent counter-clockwise along the ring. After receiving m_3, node one makes sure that all locally generated sensor events are sent, and forwards m_3 to node four. Likewise, after receiving m_3', node two makes sure that all locally generated sensor events are sent, and forwards m_3' to node four.

Similarly, if node one (two) wants to send a *locally generated* sensor event m_1 (m_2), it sends two copies of that event along the ring toward node four.

Due to the FIFO property, node four will receive at least one copy of all events with a time stamp earlier than $m_3.t$ before receiving the second copy of m_3. Consider for example message m_1 generated by node one. Since $m_1.t < m_3.t$, node one will send a copy of m_1 to node four before it forwards m_3 received from node three. The FIFO property ensures that m_1 arrives at node four before m_3.

Node four inserts all received sensor events into a list which is sorted by ascending time stamps. The first sensor event in the list (i.e., the one with the smallest time stamp) is removed and delivered to the application as soon as its second copy arrives. When the second copy of a sensor event arrives at node four, we can be sure that at least one copy of all earlier sensor events from all nodes have been received and inserted into the list. That is, a sensor event is only delivered to the application, if all messages with earlier time stamps generated by all nodes have already been delivered. This equals delivery in temporal order.

TMOS consists of four major components:

1. **Delivery groups**: Clearly, the overhead of the above scheme depends on the number of nodes involved in temporal message ordering. On the other hand, often only sensor events generated by a certain sub-set of nodes have to be delivered in temporal order. Such a set of nodes is called a *delivery group* (DG). In order to achieve scalability and energy efficiency, it is important to identify small delivery groups. TMOS provides mechanisms for specifying delivery groups.

2. **Ring setup**: This component selects and maintains a logical ring for the nodes of a DG as depicted in Figure 3.

3. **Sensor event delivery**: This component deals with the delivery of sensor events to the receiver along the ring set up earlier.

4. **Basic routing and transport**: Sensor event delivery relies on underlying basic routing and transport mechanisms to forward a message to the next node on the ring.

In the following section, we will discuss these components in detail.

4.1 Delivery Groups

A delivery group is a set of sensor nodes, such that only sensor events originating from nodes in the same DG have to be delivered in temporal order. Sensor events generated by nodes in different DG can be delivered in any order. The messages generated by nodes in a DG are delivered to one or more receivers. However, for simplicity we will only consider a single receiver. We will call the nodes in the DG *producers* while the single receiver is called *consumer*.

For reasons of scalability and energy efficiency, delivery groups should be as small as possible, since smaller DGs allow shorter rings, which result in fewer message transfers for the delivery of a sensor event. In order to form small DGs, TMOS provides mechanisms for producers to specify which sensor events they are going to generate, and for consumers to specify which sensor events they want to receive. For this purpose, sensor events are assumed to contain a name (such as "detected" or "lost") identifying the type of event, a physical time stamp t, the physical location (x, y, z) of the sensor node, and optionally additional data such as sensor readings.

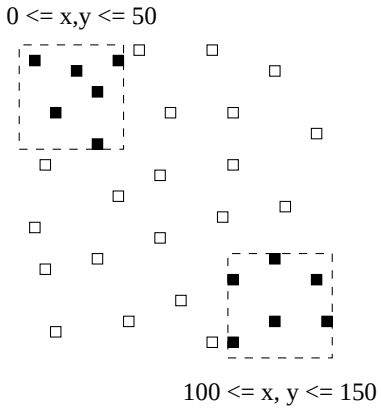

Figure 4: Two delivery groups resulting from two subscriptions with different area specifications.

Producers *advertise* sensor events by specifying the names and possible locations of sensor events they are going to generate. Consider the following two advertisements:

```
advertise detected at [10,15][0,10][20,25];
advertise lost at [10,15][0,10][20,25];
```

which announce that a producer will publish "detected" and "lost" events with $10 \leq x \leq 15$, $0 \leq y \leq 10$, and $20 \leq z \leq 25$, which represents the coverage area of its sensors. The two events might represent detection a tracked object ("detected") and no longer detecting a tracked object ("lost"). Note that advertisements of a producer change rarely over time, since the published sensor events typically depend on the sensors attached to the producer node. Moreover, nodes of a sensor network are typically immobile.

On the other hand, consumers specify a list of sensor events they want to receive by declaring their names and an area of interest. Such a subscription defines a delivery group, which includes all nodes in the network that produce sensor events with the given names in the given area. The following two subscriptions define two delivery groups:

```
subscribe {detected, lost}
  at [0,50][0,50][0,50];
subscribe {detected, lost}
  at [100,150][100,150][0,50];
```

The DG resulting from the first subscription will include producers that generate "detected" or "lost" events with $0 \leq x, y, z \leq 50$. The second subscription defines a DG containing producers also generating "detected" and "lost" events, but with $100 \leq x, y \leq 150$ and $0 \leq z \leq 50$. These two subscriptions might be issued by a tracking application, which is currently only interested in objects moving in the

two specified areas as depicted in Figure 4, which shows a simplified view of the $x - y$ plane.

Similar techniques for matching producers and consumers of information are used in content-based publish/subscribe systems in general, and Directed Diffusion[7, 9] in particular. However, in our context we use subscriptions and advertisements to group producers of related information.

Advertisements and subscriptions are local operations, which announce information to the TMOS runtime system. For each subscription, the producer executes a *DG discovery algorithm* in order to find out the members of the respective DG. The consumer initiates discovery by flooding a DISCOVERY_REQUEST message through the network. This message contains the subscription (i.e., names of sensor events and area specification) and the address of the producer. Upon arrival of this messages a producer checks whether its advertisements match the subscription contained in the message. If so, he sends a DISCOVERY_REPLY message containing its address back to the consumer.

Note that the discovery algorithm is very similar to the route discovery phase of on-demand routing algorithms such as AODV [21] and DSR [11]. Therefore, DG discovery can often be piggybacked on messages of the underlying routing algorithm without introducing much additional overhead. We will discuss this issue further in Section 4.4.

As soon as the producer knows the members of a DG, he can set up a ring for delivery of sensor events, which will be described in the following section.

4.2 Ring Setup

Ring setup arranges the producers of a DG and the consumer on a logical ring with the goal of minimizing the amount of energy required for delivering sensor events along the ring.

In order to construct such an energy efficient ring, we assume that the underlying basic routing algorithm can discover energy efficient paths between any pair of nodes in the network and that it returns a non-negative number representing the energy overhead for sending a unit length message along this path. This number is also called the *distance* between this pair of nodes. We will discuss these issues in more detail in Section 4.4.

In order to ensure energy efficiency, the nodes are arranged such that the length of the ring is minimized, where the length of the ring is the sum of the distances of each pair of adjacent nodes on the ring. For this, we have to solve the Traveling Salesman Problem (TSP), which is known to be $\mathcal{NP}$-hard. However, there exist good heuristics for efficiently approximating our instances of the TSP problem.

Table 1 lists three of these heuristics called *Nearest Neighbor* (NN), *Nearest Insertion* (NI), and *Farthest Insertion* (FI) [24]. All heuristics construct a ring (also called *tour*) starting with an initial sub-tour consisting of one arbitrarily selected node and incrementally add nodes to this sub-tour until the sub-tour contains all nodes. The heuristics differ in how to select and insert the next node into the sub-tour.

heuristic	average case	worst case
NN	1.23	$(1 + \mathrm{ld}M)/2$
NI	1.27	$2 - 2/M$
FI	1.13	$\mathrm{O}(\log M)$

Table 1: TSP heuristics performance (M = number of nodes in the tour)

With NN, the node nearest to the last inserted node is selected and inserted (appended) after this node. With NI, the node nearest to the current sub-tour[1] is selected and inserted at the position that minimizes increase of the length of the current sub-tour. FI has the same insertion rule as NI, but selects the node farthest from the sub-tour.

Table 1 summarizes the quality of the tours constructed by the three heuristics, where M is the number of nodes in the tour. The numbers indicate by which factor the constructed tour is longer than the optimal tour in the worst case and in the average case according to extensive empirical studies [12].

In TMOS we use NN, since it allows a rather simple implementation of ring setup. This is due to the "greedy" property of NN that nodes are only appended to a sub-tour, i.e. a once constructed sub-tour is never changed. However, the energy efficiency of TMOS could be increased by about 10% on average by using FI instead of NN according to Table 1. The use of NI would guarantee a constant upper bound of two on the worst case performance.

The outline for a distributed algorithm implementing ring setup using NN is as follows: The algorithm exchanges a single type of message containing a set of unvisited nodes. The algorithm starts at the consumer by initializing the message to contain all producers of the DG as unvisited nodes. Then the consumer determines the node nearest to itself, records this node as its right ring neighbor, and sends the message off to this node. The receiving node records the sender of the message as its left ring neighbor and removes itself from the set of unvisited nodes. Then the nearest unvisited node is selected and recorded as the right ring neighbor. The message is then sent to this nearest node. If the set of unvisited nodes is empty, the message is sent back to the consumer. The consumer records the sender of this message as its left neighbor, which completes ring setup.

In order to make ring construction robust, the consumer sets up a timer when it starts executing ring construction. If the timer expires before arrival of the final message, the consumer repeats ring construction. In order to adopt to changes in the network, DG discovery and ring setup are periodically re-executed.

We will now describe the ring setup algorithm in more de-

[1]The node that has the smallest distance to any node in the current sub-tour.

tail. In order to distinguish delivery groups, the consumer assigns a globally unique identifier to each delivery group. Each node maintains two maps L and R which map delivery group IDs to the left and the right ring neighbor for this delivery group. Furthermore, each node maintains a map D of distances to all other members of the delivery group using the basic routing algorithm.

The algorithm uses a single message type which contains the following elements:

- ID: unique identifier of the DG
- C: consumer address
- U: set of unvisited nodes

In order to initiate ring setup, the consumer sets up a timer and performs the following steps:

/* Initialize message */
$m.ID :=$ ... /* a globally unique ID */
$m.C :=$ ... /* consumer address */
$m.U := \{...\}$ /* all producers in DG */
Choose n from $m.U$, such that $D(n)$ is minimized
/* Set right neighbor */
$R(m.ID) := n$
Send m to node n using basic routing
Receive m' from node n'
/* Set left neighbor */
$L(m'.ID) := n'$

If the timer expires before arrival of m', the whole procedure is repeated.

A producer node performs the following steps upon arrival of a ring construction message m:

Receive m from node n
/* Set left neighbor */
$L(m.ID) := n$
Remove self from $m.U$
/* Set right neighbor and forward message */
If $m.U$ is empty:
    $R(m.ID) := m.C$
    Send m to node $m.C$ using basic routing
Else:
    Choose n' from $m.U$, such that $D(n')$ is minimized
    $R(m.ID) := n'$
    Send m to node n' using basic routing

Distributed algorithms for NI and FI are somewhat more complicated, since they are not greedy algorithms. That is, the final neighbors cannot be determined until the whole ring has been constructed. Therefore, after ring construction, a second phase is needed to distribute the constructed ring to all producer nodes.

4.3 Sensor Event Delivery

After ring setup, each producer of the DG knows its left and right neighbor on the ring. The outline of the algorithm for delivering a sensor event m to the consumer along this ring is as follows. The producer sends two copies of the event to its two neighbors. Each producer forwards a received event copy to its other neighbor using the basic routing algorithm.

Upon arrival of the first copy of a sensor event, the consumer inserts the event into a list which is sorted by ascending sensor event time stamps. There is a copy counter associated with each list entry, which is set to one upon arrival of the first copy and incremented by one upon arrival of the second copy. Upon arrival of the second copy of the first (i.e., earliest) entry in the list, the respective sensor event is removed from the list and delivered to the application.

We will now explain the delivery algorithm in more detail. Let us assume that each sensor event contains an element ID holding the delivery group identifier. A producer P which wants to send a sensor event m to consumer C, performs the following steps:

/* Set up delivery group ID */
$m.ID :=$ ...
/* Send to left neighbor */
Send m to node $L(m.ID)$ using basic routing
/* Send to right neighbor */
Send m to node $R(m.ID)$ using basic routing

When a producer receives such a message m, it performs the following actions:

Receive m from node n
Send all pending local events
If $n = L(m.ID)$:
    Send m to node $R(m.ID)$ using basic routing
Else:
    Send m to node $L(m.ID)$ using basic routing

Since locally generated events might contain time stamps earlier than the time stamp of m, the producer has to send them off before forwarding m. This is important to ensure FIFO ordering over multiple producer hops on the ring.

The consumer maintains a list L of sensor events, which is ordered by ascending time stamps. Each entry of the list consists of a sensor event m and associated copy counter CC.

Upon arrival of a message m, the consumer performs the following steps:

```
Look up position k of message m in list L
If found:
    L[k].CC := 2
Else:
    Insert m into L with CC := 1
Repeat:
    /* Consider first element L[0] for delivery */
    If L[0].CC ≠ 2:
        /* Copies missing */
        STOP
    Deliver L[0]
    Remove L[0] from L
```

After updating the list L accordingly, the first elements of the list for which both copies arrived, are delivered to the consumer and removed from the list.

Due to failures in the network, one or both copies of a sensor event might not arrive at the consumer. We assume that the underlying routing algorithm can fix temporary problems by retransmitting packets or by using alternate routes through the network. See Section 4.4 for details.

At the TMOS layer, lost sensor events therefore indicate a permanent problem in the network. Fixing such a "broken ring" will typically require re-execution of DG discovery and ring construction as described in Sections 4.1 and 4.2.

A failure of a single node will typically affect only one of the two paths along which the two event copies are sent. In this case, only one event copy will arrive at the consumer. To detect such a situation, the consumer can set up a timer when the first event copy arrives. If the timer expires before the second copy arrives, the consumer assumes the ring is broken and re-executes DG discovery and ring setup.

If two or more nodes fail concurrently, chances are that the consumer will not receive messages any longer. In this case the consumer cannot decide if the network is idle (i.e., does not generate sensor events) or if the ring is broken. However, if the producers send empty keep-alive events at regular (but long) intervals, the consumer can assume a broken ring if it did not receive any messages for a long period of time. Again, DG discovery and ring setup are re-executed in order to fix the problem.

4.4 Basic Routing and Transport

We assumed the existence of basic message routing and transport mechanisms in the sensor network with the following properties. For any pair of nodes A, B, the algorithm should:

- find an energy efficient path from A to B,
- return a non-negative number representing the energy overhead for sending a unit length message from A to B (the distance between A and B),
- ensure that messages sent by A arrive at B in the same order (FIFO property),
- deliver messages at most once, and
- deliver messages with high reliability.

Examples for such routing mechanisms are on-demand ad hoc routing protocols like AODV [21] or DSR [11] with power-aware distance metrics [30] and localized power-aware routing algorithms for sensor networks such as [28].

The route discovery phase of many of these demand-driven algorithms is very similar to DG discovery as described in Section 4.1. A route to a destination address A is typically discovered by flooding the network with a route discovery message containing A. If A receives the discovery message, it sends back a reply message along the reverse path. In this case, DG discovery can be easily piggybacked to route discovery by adding some additional elements to the route discovery messages.

A FIFO channel from node A to node B can be easily implemented by assigning monotonically increasing integer sequence numbers to messages sent by A. B remembers the sequence number of the last message LS_A received from A. Now if B receives a message m with sequence number S_m from A, it checks if $S_m = LS_A + 1$. If so, the message arrived in order and can be delivered; LS_A is set to S_m. Otherwise, m is buffered until all messages with sequence numbers $LS_A + 1, \ldots, S_m - 1$ arrived. Sequence numbers can also be used to implement at-most-once message delivery since duplicated messages will carry the same sequence number.

Reliable message delivery can be achieved by acknowledging single-hop message transfers and retransmitting lost or destroyed messages. In broadcast-based communication systems, acknowledgments due not require additional messages, since the sender of a message m can "hear" the receiver forwarding m to the next hop. Message loss due to node failures and network partitions has to be dealt with at the TMOS layer as described in the previous sections.

5 SIMULATION

In order to get a first impression of TMOS' performance, we simulated TMOS on top of a perfect routing layer. This routing layer uses the hop-length of the shortest path as the distance between nodes and delivers messages always along shortest paths. Figure 5 shows a sample ring constructed by TMOS under these conditions, where the black circle depicts the consumer and black squares depict producers in the DG.

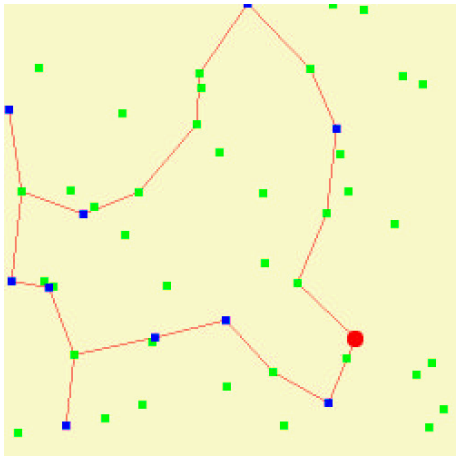

Figure 5: Ring constructed by TMOS.

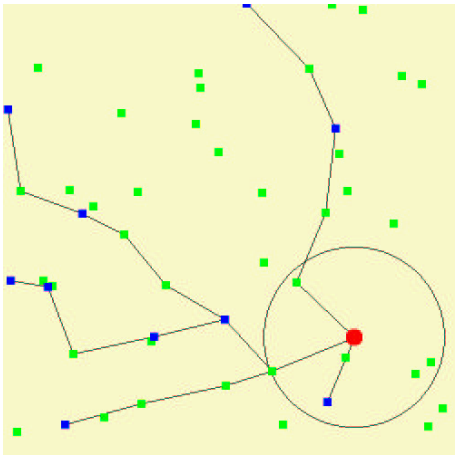

Figure 6: Shortest paths from producers to consumer used in the simulation of the heartbeat protocol. The large circle depicts the communication range of the nodes.

For comparison we also simulated the heartbeat algorithm described in Section 3. There, every producer node must send a heartbeat event to the consumer in the worst case before a sensor event can be delivered. All events are sent along shortest paths to the consumer. Figure 6 shows the shortest paths from the producers of the DG to the consumer for the same setting as in Figure 5.

The simulation setting is as follows: Fixed nodes with a wireless communication range of 20 meters are uniformly placed within an area of 100m x 100m. There is exactly one consumer and a certain percentage of all nodes are assumed to be producers in a delivery group. The producers are placed within a square in the upper left of the whole area, which covers a certain percentage of the whole area. The default parameters for the simulations are as follows:

- Communication range: 20m
- Simulation area: 100m x 100m
- Total number of nodes: 100
- Percentage of producers: 20%
- Percentage of producer sub-area: 100%

Depending on these parameters, we want to examine the amount of energy and the delay required for delivering a single sensor event. Energy is the sum of energy spent by nodes in the network for delivering the event from the producer to the consumer. Delay is the amount of time elapsed between detection of an event in the producer and delivering the respective event notification to the application at the consumer.

We use the number of single-hop message transfers as a measure for energy. In general, energy consumption will not only depend on the number of single-hop message transfers, but also on the size of the messages. However, the messages exchanged by TMOS and the heartbeat protocol are about the same size, since they mostly consist of a time stamp and sensor event data. Therefore, we can compare the relative performances of the two protocols without considering message sizes.

The delay includes network latency and other delays such as Δ for Heartbeat (see Section 3.1). This simulation only considers network latency (measured in number of hops) as a lower bound for total delay. For TMOS, the total delay is dominated by the network latency. For Heartbeat, however, the delay is typically dominated by Δ.

We performed 3 sets of simulations, each varying one of the parameters total number of nodes, percentage of producers, and producer sub-area while keeping the other parameters fixed. A single simulation run examines the average amount of energy and average network delay for delivering a single sensor event to the consumer in a fixed network topology for TMOS and the heartbeat protocol as described above. Each of the three simulation sets determines the average of 100 such runs. The network topology is changed between each of the runs by randomly placing the nodes (producers) in the simulation area (producer sub-area).

Figure 7 (8) shows energy (delay) depending on the number of nodes, on the producer area, and on the producer percentage, respectively. In the examined settings, TMOS consumes between 28% and 67% of the energy spent by Heartbeat. While the network delay of Heartbeat is mostly independent of the examined parameters, the network delay of TMOS grows linearly with node number and producer percentage due to the resulting increasing length of the ring. Note, however, that the total delay of Heartbeat is typically dominated by the Heartbeat interval Δ (see Section 3.1), such that comparing the network delays of Heartbeat and TMOS does not reflect the actual relative performance. Section 6 discusses ways to reduce TMOS' network latency.

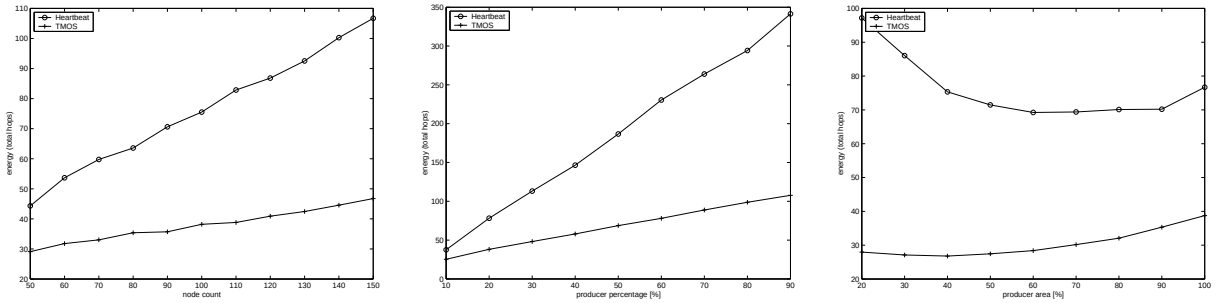

Figure 7: Simulated energy consumption for varying node number (left), varying DG area (middle), and varying producer percentage (right).

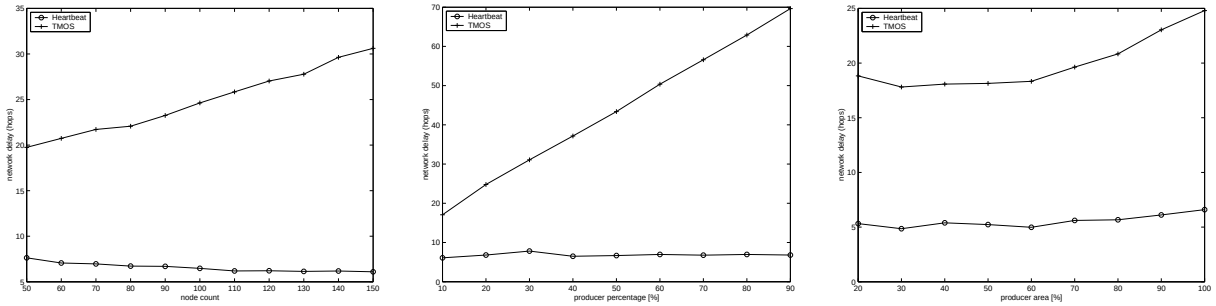

Figure 8: Simulated network delay (in hops) for varying node number (left), varying DG area (middle), and varying producer percentage (right). The numbers for Heartbeat do not include additional delays caused by Δ (see Section 3.1).

Note that this first simulation only considered sensor event delivery. However, both the Heartbeat protocol and TMOS require a setup phase to (1) discover nodes in the delivery group and to (2) setup message routing. (1) can be assumed to be the same for Heartbeat and TMOS. For Heartbeat, (2) includes finding (shortest) paths from all DG producers to the consumer. For TMOS, (2) includes setting up the ring as described in Section 4.2. However, the costs of (2) heavily depend on the underlying basic routing scheme. A more thorough evaluation will have to examine these costs and the MAC-layer effects described in Section 2.1.

6 DISCUSSION

In this section we want to discuss some properties and potential improvements of the TMOS scheme for temporal message ordering in sensor networks.

The correctness of TMOS is guaranteed by the properties of the logical ring used for message transfers. We assumed that the basic routing algorithm provides FIFO channels between pairs of nodes. The ring consists of a closed chain of such pairs of nodes. By taking care when forwarding messages in the producers, we can easily extend the FIFO property to any pair of non-adjacent nodes on the ring.

Due to this extended FIFO property, the two copies m_2 and m_2' of a sensor event sent along the two half-rings connecting a producer with the consumer will "shift" earlier sensor events out of the ring. Consider the two copies m_1 and m_1' of an earlier sensor event (i.e., $m_1.t < m_2.t$) generated by a different producer. At least one of the copies m_1 or m_1' will be shifted out of the ring by either m_2 or m_2'. In Figure 9, for example, m_1' will be shifted out of the ring by m_2' (the consumer is depicted by a circle, producers are depicted by squares).

Formally speaking, this property can be expressed as follows. Let $R(P, C)$ be the set of paths from producer P to consumer C on the ring, i.e. $|R(P, C)| = 2$. Each path $R \in R(P, C)$ is a list of nodes $\{n_1, n_2, ..., n_{|R|}\}$ on the ring with $n_1 = P$ and $n_{|R|} = C$. Then the following property holds:

$$\forall P_1, P_2 \in DG : \exists R_1 \in R(P_1, C), R_2 \in R(P_2, C) : R_2 \sim R_1 \quad (2)$$

where $R_2 \sim R_1$ is true iff the path R_2 is a suffix of path R_1. In words: for any two producers P_1 and P_2 there exist paths R_1 from P_1 to C and R_2 from P_2 to C, such that R_2 is a suffix of R_1. A sensor event sent by P_1 will then shift out one copy of all earlier sensor events sent by P_2 due to the FIFO property. Since this applies to all $P_2 \in DG$, the arrival of the second copy of a sensor event sent by P_1 guarantees that at least one copy of all earlier sensor events have already arrived at the consumer. We can therefore achieve temporal message ordering by the approach described in Section 4.

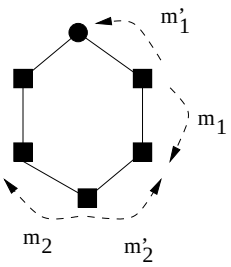

Figure 9: Correctness of TMOS: message m_2' will shift message m_1' out of the ring.

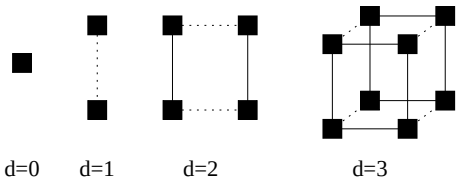

Figure 10: Recursive construction of hypercubes.

The use of a ring has some additional advantages. All producers of a DG use the same topology for forwarding messages. Routing messages along a ring therefore only requires neighborhood knowledge in the nodes and does not require additional routing information to be included into the sensor events. Additionally, sending two copies of each sensor event results in some amount of robustness to node failures as described in Section 4.3.

One potential drawback of TMOS is that the ring can become quite large for delivery groups with many nodes, which results in long paths between producers and the consumer. However, this can be fixed by using other topologies with characteristics similar to the ring. One such topology is the hypercube.

Hypercubes can be recursively constructed using the following rule. A single node is a hypercube with dimension 0. A hypercube with dimension d is obtained by taking two hypercubes with dimension $d-1$ and connecting the corresponding vertices as depicted by the dotted lines in Figure 10 for $d = 0, ..., 3$. Note that a hypercube with dimension 2 is a ring with four nodes.

By arranging the nodes of a delivery group on a hypercube instead of a ring, the maximum path length for the delivery of sensor events can be reduced at the cost of a higher energy consumption due to the increased message parallelism. The latter will also result in increased robustness in case of node failures. Figure 11 shows an example for sensor event delivery on a hypercube. Again, the consumer is depicted by a circle, producers are depicted by squares. However, there are a number of open questions, such as the construction of energy efficient hypercubes or routing of sensor events on hypercubes such that the path suffix property 2 holds. We are currently investigating these issues.

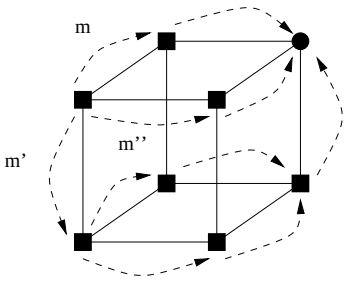

Figure 11: Sample sensor event delivery on a hypercube.

7 CONCLUSION

Temporal message ordering is an important component of sensor network applications, where individual sensor readings are fused in a time-dependent manner in order to obtain high-level sensing results.

We pointed out that sensor networks suffer from long and highly variable network delays due to the properties of the used communication technologies and due to network dynamics. Moreover, typical sensor network applications require immediacy, robustness, scalability, and energy efficiency. Due to these requirements and characteristics of sensor networks, solutions for temporal message ordering in traditional computer networks are not suited for sensor networks.

We introduced TMOS, a temporal message ordering scheme for wireless sensor networks. TMOS uses delivery groups to reduce the overhead involved in temporal message ordering. Using distributed TSP-heuristics, an ring topology is constructed, which is used to deliver sensor events in an energy-efficient and scalable way. TMOS can be used on top of a wide variety of basic routing algorithms such as DSR and AODV.

According to simulations, TMOS can save up to 70% of the energy spent by the heartbeat protocol in the examined settings at the cost of an increased network latency. On the other hand, the overall delay of traditional message ordering algorithms is bound by "artificial" delays. We discussed potential ways to reduce TMOS' network latency by using topologies like hypercubes instead of rings.

Currently we are working on a more thorough simulation and analysis of TMOS. TMOS is currently being used in a prototype vehicle tracking sensor network based on Smart-Its sensor nodes [32].

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, March 2002.

[2] K. P. Birman and T. A. Joseph. Reliable Communication in the Presence of Failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[3] J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *OSDI 2002*, Boston, USA, December 2002.

[4] J. Elson and K. Römer. Wireless Sensor Networks: A New Regime for Time Synchronization. In *Workshop on Hot Topics in Networks (Hotnets-I)*, Princton, USA, October 2002.

[5] H. Garcia-Molina and A. Spauster. Ordered and Reliable Multicast Communication. *ACM Transactions on Computer Systems*, 9(3):242–271, August 1991.

[6] R. Hayton. *OASIS: An Open Architecture for Secure Interworking Services*. PhD thesis, University of Cambridge, 1996.

[7] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *SOSP 2001*, Lake Louise, Banff, Canada, October 2001.

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. In *ASPLOS 2000*, Cambridge, USA, Nov. 2000.

[9] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *6th Intl. Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, USA, August 2000.

[10] X. Jia. A Total Ordering Multicast Protocol Using Propagation Trees. *IEEE Transactions on Parallel and Distributed Systems*, 6(6):617–627, June 1995.

[11] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[12] D. S. Johnson and L. A. McGeoch. Experimental Analysis of Heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, 2002.

[13] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Emerging Challenges: Mobile Networking for Smart Dust. *Journal of Communications and Networks*, 2(3):188–196, September 2000.

[14] O. Kasten and M. Langheinrich. First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network. In *Workshop on Ubiquitous Computing and Communications, PACT 01*, Barcelona, Spain, October 2001.

[15] J. P. Kearns and B. Koodalattupuram. Immediate Ordered Service in Distributed Systems. In *9th Int. Conference on Distributed Computing Systems (ICDCS 89)*, pages 611–618, Newport Beach, USA, June 1989.

[16] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(4):558–565, July 1978.

[17] M. Mansouri-Samani and M. Sloman. GEM – A Generalised Event Monitoring Language for Distributed Systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4(25), February 1997.

[18] F. Mattern. Virtual Time and Global States in Distributed Systems. In *Workshop on Parallel and Distributed Algorithms*, Chateau de Bonas, October 1988.

[19] G. J. Nelson. *Context-Aware and Location Systems*. PhD thesis, University of Cambridge, 1998.

[20] T. P. Ng. Ordered Broadcasts for Large Applications. In *IEEE Intl. Conf. on Distributed Computing Systems*, June 1991.

[21] C. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 99)*, New Orleans, USA, February 1999.

[22] S. Quaireau and P. Laumay. Ensuring Applicative Causal Ordering in Autonomous Mobile Computing. In *Workshop on Middleware for Mobile Computing (Middleware 2001)*, Heidelberg, Germany, November 2001.

[23] K. Römer. Time Synchronization in Ad Hoc Networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 01)*, Long Beach, CA, October 2001.

[24] D. J. Rosenkrantz, R. E. Strearns, and P. M. Lewis II. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM Journal on Computing*, 29:563–581, 1977.

[25] A. Schiper, J. Eggli, and A. Sandoz. A New Algorithm to Implement Causal Ordering. In *Workshop on Distributed Algorithms*, pages 219–232, Nice, France, 1989.

[26] Y. C. Shim and C. V. Ramamoorthy. Monitoring and Control of Distributed Systems. In *First Intl. Conference of Systems Integration*, pages 672–681, Morristown, USA, 1990.

[27] C. Skawratananond, N. Mittal, and V. K. Garg. A Lightweight Algorithm for Causal Message Ordering in Mobile Computing Systems. Technical Report TR-PDS-1998-11, Parallel and Distributed Systems Group, University of Texas at Austin, November 1998.

[28] I. Stojmenovic and X. Lin. Power aware localized routing in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1122–1133, November 2001.

[29] B. Warneke, M. Last, B. Leibowitz, and K. S. J. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. *IEEE Computer Magazine*, 34(1):44–51, January 2001.

[30] M. Woo, S. Singh, and C. S. Raghavendra. Power-Aware Routing in Mobile Ad Hoc Networks. In *Mobicom 98*, Dallas, USA, October 1998.

[31] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *IEEE Infocom 2002*, New York, USA, June 2002.

[32] Smart-Its Project. www.smart-its.org.