Towards a Unified View on Space and Time in Sensor Networks*

Kay Römer and Friedemann Mattern
Institute for Pervasive Computing
ETH Zurich
{roemer,mattern}@inf.ethz.ch

Abstract

In wireless sensor networks, many applications rely on the ability of sensor nodes to estimate their physical location and time with respect to a common reference scale. This necessity is reflected by the development of a significant number of algorithms for localization and time synchronization for sensor networks in the recent past. However, research on location estimation on the one hand and on time synchronization on the other hand has been largely separated. Despite this, models, requirements, techniques, and algorithms of the two domains are rather similar and in some respects closely related. The purpose of this paper is to make this affinity explicit, with the hope of stimulating a mutual fertilization and enabling a better understanding of both domains.

1 Introduction

In wireless sensor networks – large wireless networks of tiny computing and sensing devices –, space and time play a crucial role, since sensor nodes are used to collaboratively monitor physical phenomena and their spatio-temporal properties. Consequently, a number of techniques and distributed algorithms for location estimation and for time synchronization have been developed specifically for sensor networks. Research in these two domains has been performed by mostly separated research communities.

A closer look on both research domains reveals that there are many similarities[1]. This does affect a variety of aspects of location estimation and time synchronization, ranging from applications and requirements to basic approaches and concrete algorithmic techniques. The purpose of this paper is to point out this close affinity. The hope is that this could lead to a better understanding of both domains and to a mutual fertilization, where results from one domain could be adopted by the other.

The remainder of this paper is structured as follows. In Section 2 we point out uses of space and time in sensor networks. Section 3 presents a common model for location estimation and time synchronization and discusses various requirements and basic approaches to location estimation and time synchronization based on this model. In Section 4 we discuss the structure of distributed algorithms for location estimation and time synchronization. In particular, we will point out that both kinds of algorithms are based on few common structural elements. Section 5 presents various limitations and trade-offs of this class of algorithms and Section 6 concludes the paper.

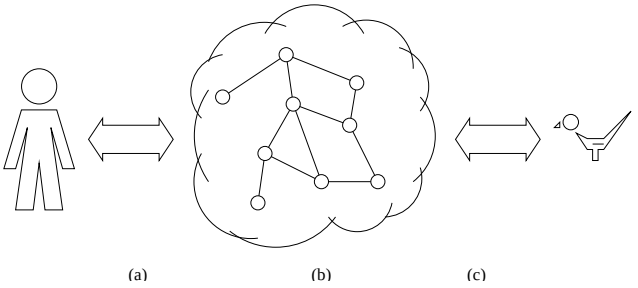

Figure 1: Applications of space and time. (a) interaction of an external observer with the sensor network, (b) interaction among sensor nodes, (c) interaction of the sensor network with the monitored real world.

2 Uses of Space and Time

Figure 1 illustrates three important use classes of space and time in sensor networks. Typically, a sensor network is tasked by and reports results to an external observer (a). A sensor network also interacts with the physical world through distributed sensors and possibly also through actuators (c). Finally, the sensor nodes interact among each other to coordinate distributed computations (b). The following paragraphs will discuss applications of space and time in these three domains.

2.1 Sensor Network – Observer

In many applications, a sensor network interfaces to an external observer for tasking, reporting results, and management. This external observer may be a human operator (as depicted in Figure 1) or a computer system. Tasking a sensor network often involves the specification of regions of

*This work was partly supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant no. 5005-67322.

[1]We acknowledge that there are also significant differences, but this paper focuses on the similarities.

interest in spacetime such as “only during the night” or “the area south of ...”. Since the observer is typically interested in a physical phenomenon of the real world (and not in individual sensor nodes), such spacetime addressing is often preferable over addressing individual nodes or groups of nodes by identifiers.

As a sensor network reports monitoring results to the observer, many spatio-temporal properties of observed physical phenomena are of interest. For example, time and location of occurrence of a reported physical event are often crucial to associate event reports with the originating physical events. Properties such as size, shape, speed, trajectory, density do all refer to the categories time and space.

2.2 Sensor Network – Real World

In sensor networks, many different sensor nodes distributed over an area may be involved in the observation of a single physical phenomenon. One of the key functions of a sensor network is therefore the assembly of many distributed observations into a coherent estimate of the original physical phenomenon – a process known as data fusion. Space and time are key ingredients for data fusion. For example, many sensors can only detect the proximity of an observed object. Higher-level information, such as speed, size, or shape of an object can then only be obtained by correlating data from multiple sensor nodes whose locations are known. The velocity of a mobile object, for example, can be estimated by the ratio of the spatial and temporal distances between two consecutive object sightings by different sensor nodes. As another example, the size and shape of a widespread object can be approximated by the union of the coverage areas of the sensor nodes that concurrently detect the object.

Since many different instances of a physical phenomenon can occur in spatio-temporal proximity, one of the tasks of a sensor network is the separation of sensor samples, that is, the partitioning of sensor samples into groups that each represent a single physical phenomenon. Spatio-temporal relationships (e.g., distance) among sensor samples are a key element for separation.

Spatio-temporal coordination among sensor nodes may also be necessary to ensure correctness and consistency of distributed measurements [9]. For example, if the sampling rate of sensors is low compared to the temporal frequency of an observed phenomenon, it may be necessary to ensure that sensor readout occurs concurrently at all sensor nodes in order to avoid false observation results. This is also an issue for sensor calibration as explained in [2]. Likewise, the spatial distribution of sensors has an impact on the correctness of observation results. For example, in order to estimate the average of a certain physical quantity over a certain physical area (e.g., average room temperature), it is typically not sufficient to simply calculate the average over all sensor nodes covering the area, because then areas with higher node density would be overrepresented in the resulting average value.

It is anticipated that in the future large-scale and complex actuation functions will be realized by coordinated use of many simple distributed actuator nodes that are part of a sensor network. Similar to distributed measurements, spatio-temporal coordination will then also be an important ingredient for consistent distributed actuation.

2.3 Within a Sensor Network

Time and location are also valuable concepts for intra-network coordination among different sensor nodes. As sensor networks are essentially distributed systems, many traditional uses of the concepts of time and location do also apply to wireless sensor networks. Liskov [12] points out a number of uses of time in distributed systems in general such as for concurrency control (e.g., atomicity, mutual exclusion), security (e.g., authentication), data consistency (e.g., cache consistency, consistency of replicated data), and communication protocols (e.g., at-most-once message delivery).

One particularly important example for concurrency control is the use of time-division multiplexing in wireless communication, where multiple shared access to a common communication medium may be realized by assigning time slots with exclusive access to the communicating peers. This may require the participating sensor nodes to share a common view on physical time. A prominent use of spatial information for network coordination is geographic node addressing and routing, where geographic locations replace the use of node identifiers.

A number of approaches intend to improve energy efficiency by selectively switching sensor nodes or components thereof into power-saving sleep modes. In order to ensure seamless operation of the sensor network despite of this, spatio-temporal coordination among sensor nodes may be required. The algorithm presented in [27], for example, extends the lifetime of dense networks by switching off nodes such that the remaining nodes are sufficient to cover the area of interest. To ensure coverage, node locations must be known. Another way of extending network lifetime is to periodically switch off radio transceivers of sensor nodes, since their power consumption is rather high even when only listening to the network. Temporal coordination is required to ensure that activity periods of sensor nodes overlap in time in order to enable communication (see, e.g., [28]).

Another service of importance for sensor network applications is temporal message ordering [19]. Many data-fusion algorithms have to process sensor readings ordered by the time of occurrence (e.g., in the approach for velocity estimation sketched above). However, highly variable message delays in sensor networks imply that messages from distributed sensor nodes typically do not arrive at a receiver in the order they have been sent. Reordering messages according to the time of sensor readout requires temporal coordination among sensor nodes.

The close relationship between time and space in the physical world is also reflected by methods for time synchronization and location estimation themselves. For example, methods for location estimation based on the measurement of time of flight or time difference of arrival of certain signals typically require synchronized time. The other way round, location information may also help to achieve time synchronization. This is due to the fact that time synchronization approaches often have to estimate message delays. One component of the message delay is the time of flight of the carrier signal between two nodes, which can be calculated if the distance between sender and receiver and the propagation speed of the carrier signal are known.

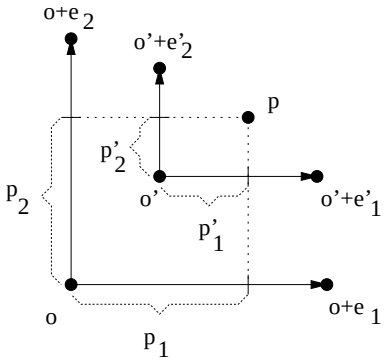

Figure 2: A point p in spacetime and its coordinates p_i and p'_i in two different coordinate systems.

3 Locating Nodes in Spacetime

In this section we develop a common model for location estimation and time synchronization. Using this model, we will discuss various requirements on and different classes of time synchronization and localization.

One possible way to model physical space is to do this as a three-dimensional real-valued vector space. Likewise, physical time can be modeled as a one-dimensional real-valued vector space. These two vector spaces are often combined to form a four-dimensional vector space known as *spacetime*. To indicate points in spacetime, a coordinate system is used, consisting of the vector o (the origin) and four linearly independent vectors e_1, e_2, e_3, e_4 (the axes). To avoid relativistic effects and to simplify our discussion, we assume that a coordinate system has the following properties: $e_4 = (0, 0, 0, t)$, the e_i are mutually orthogonal (i.e., inner product is zero), and $|e_1| = |e_2| = |e_3|$. In other words, the space axes e_1, e_2, e_3 form a Cartesian coordinate system, e_4 is the time axis, and $|e_1|, |e_2|, |e_3|$ and $|e_4|$ are the space and time units, respectively. Any point p in spacetime can now be specified by its coordinates (p_1, p_2, p_3, p_4) with respect to the coordinate system (o, e_1, e_2, e_3, e_4), such that p is given by $o + p_1e_1 + p_2e_2 + p_3e_2 + p_4e_4$.

Under these assumptions, the spatial distance between two points p and q is given by $\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}$, and the temporal distance is given by $|p_4 - q_4|$.

The above model allows a holistic view on localization and time synchronization as follows. If a sensor node is modeled as a point p in spacetime, localization and time synchronization can be considered as determining the current coordinates of p with respect to a given coordinate system. We refer to this process as *locating a sensor node in spacetime*.

Note that it is quite common to use different coordinate systems. However, using a simple coordinate transformation scheme, the coordinates p_i of a given point p can be transformed into coordinates p'_i in a primed coordinate system, as depicted in Figure 2 for a two-dimensional coordinate system.

In the remainder of the paper we will simply use the terms *localization / location* as an abbreviation for localization / location in spacetime. We will use *localization / location in time* and *localization / location in space* when specifically referring to time and space.

Localization in spacetime comes in many different flavors and with many different requirements and practical constraints, which are discussed in the following sections.

3.1 Internal vs. External

With external localization in spacetime, a given coordinate system is used as a reference. With internal localization, there exists no predefined coordinate system. The nodes of a sensor network then have to agree on a single coordinate system, but which one is actually chosen is irrelevant.

Note that external localization is a special case of internal synchronization, since a coordinate transformation can be applied to map coordinates w.r.t. an arbitrary coordinate system used for internal synchronization to coordinates w.r.t. a predefined external coordinate system.

External localization is mostly used when interfacing to the real world and observers, since there are well-established coordinate systems used in daily life such as the coordinate system defined by Universal Transverse Mercator (UTM) space coordinates and Coordinated Universal Time (UTC). For spatio-temporal coordination among sensor nodes, internal localization is often sufficient.

3.2 Global vs. Local

In order to be able to compare two points p and q in spacetime, the coordinates of the two points must be known w.r.t. a single coordinate system. The most obvious way to achieve this is to have all network nodes use a single global coordinate system. In this case, any sensor node can easily compare any two points in spacetime obtained from any two nodes.

However, the use of a single global coordinate system is not the only possible solution. As illustrated in Figure 3, small sets of nodes or even single nodes may use a local coordinate system each. If points in spacetime remain within

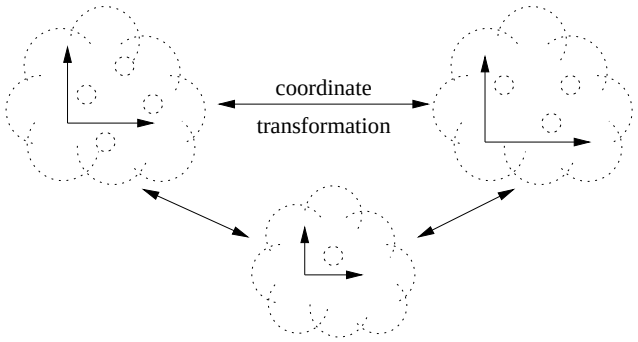

Figure 3: Small sets of nodes may use local coordinate system. If a point in spacetime is passed beyond the scope of such a local coordinate system, a coordinate transformation is applied.

the scope of such a local coordinate system, they can be easily compared. However, if coordinates of points in spacetime are passed across the border between the scopes of two different local coordinate systems, a coordinate transformation must be applied to the point.

When to prefer the one or the other approach depends on the actual application. Maintaining a coordinate system across a set of distributed network nodes requires communication among the participating nodes. However, comparing points within the scope of a single coordinate system then comes for free. Local coordinate systems typically do not require active communication among nodes using different coordinate systems. Passing points across a coordinate system boundary, however, requires to compute a suitable transformation between the two involved coordinate systems, which then has to be applied to the affected points.

Therefore, a reasonable approach would be to cluster nodes based on their interaction patterns, where each cluster has a local coordinate system. If two nodes do frequently exchange points in spacetime, they should end up in the same cluster. If there is strong interaction among all nodes in the network, using a single global coordinate system is likely the better choice.

3.3 Point Estimates vs. Bounds

Actual implementations of localization in spacetime are based on measurements. Since measurements are always afflicted with errors, only estimates of the coordinates of a point in spacetime can be obtained in practice. Despite this, it is often convenient to use such point estimates as if they were correct in the absolute sense.

Another approach is to explicitly deal with errors in measurements by specifying bounds on the actual coordinates of a point in spacetime, where one assumes that the true value lies within the bounds. Common ways of specifying bounds are bounding boxes and spheroids. In the special case of (one-dimensional) time, both map to intervals.

Both point estimates and bounds have advantages and disadvantages that influence the choice of one over of the other. Basically, point estimates are convenient to use due to the simplicity of point arithmetic and because statements in terms of the abstract spacetime model can be directly applied to the point estimates. However, the use of point estimates may lead to wrong results. For example, for two point estimates $\hat{p}$ and $\hat{q}$ where $\hat{p}_4 < \hat{q}_4$ holds, p may despite of this actually represent a later point in time than q.

While the use of explicit bounds is often more complex and inconvenient, and sometimes rather imprecise, errors like the above one can be avoided. However, the use of bounds also introduces situations where it is impossible to decide on a certain expression. For example, if bounding boxes (i.e., intervals) are used to represent points in time, it cannot be determined whether one point is earlier than another if the corresponding intervals overlap. While the introduction of such undecidable situations may seem undesirable from a technical point of view, they explicitly represent fundamental limitations of the system and alert the application or user about it, instead of making arbitrary and potentially wrong decisions.

Yet another approach to deal with the imprecision of localization algorithms is the use of probability distributions over spacetime. However, due to the practical difficulty of dealing with probability distributions, this approach is currently not used in distributed algorithms for sensor networks.

3.4 Points vs. Distances

In the previous discussions we assumed that points in spacetime originating from different nodes in the sensor network are compared. However, there are also applications where individual nodes locally measure distances between points in spacetime and where it is sufficient to compare distances measured at different sensor nodes.

As an example, consider an application where sensor nodes measure the time during which a certain phenomenon can be sighted and where the sighting durations at different sensor nodes must be compared (e.g., to estimate the acceleration of a mobile object). Here, the actual points in time when the phenomenon appeared or disappeared are irrelevant. However, it is important that different sensor nodes measure the same duration given identical physical stimuli.

Obviously, measuring and comparing distances is a special case of measuring and comparing points in spacetime, since a distance can be easily calculated when the points are given w.r.t. a common coordinate system.

3.5 Scope and Lifetime

A *scope* defines a subset of nodes where localization in spacetime is required. A *lifetime* defines a time interval during which localization is required. The two extremes are everywhere/continuous (e.g., time synchronization to coordinate access to the communication channel) and on-demand, where localization is only performed where and when actu-

ally needed (e.g., to estimate the location of a an instantaneous physical event).

Both lifetime and scope requirements can vary from application to application and may change dynamically and in unpredictable ways. In many sensor network applications, scope and lifetime are correlated with the occurrence of the observed physical phenomena. For example, to locate an object moving through a sensor network, nodes that can detect the object might define the scope and the lifetime.

With everywhere/continuous localization, the localization procedure is performed permanently on all nodes, such that an up-to-date estimate of the current location in spacetime is immediately available whenever requested by the application. With on-demand localization, the localization procedure is performed only where (i.e., on a certain node) and when the application requests the current location in spacetime. The result is only available after a delay caused by the execution of the localization algorithm.

The overheads of the two approaches depend on the frequency of the application requesting localization. If rarely requested, on-demand localization may be more efficient. If frequently requested, continuous localization is likely to be more efficient. In sensor networks, where activity is often triggered by the occurrence of rare physical events, the on-demand approach is certainly a promising technique for achieving resource efficiency.

3.6 Precision

A localization algorithm yields a point estimate or bounds on a sensor node's actual position p in spacetime. Precision is a measure for how well this result matches the ground truth locations p of nodes across the network over time. For algorithms returning point estimates, the *instantaneous* precision for a given node at a given point in time is usually expressed in terms of the distance between the point estimate and p. Algorithms that return bounds are error-free if p is actually enclosed by the bounds. However, the precision of bound-based algorithms can be expressed by the uncertainty of the bounds (e.g., the volume of a bounding box, the length of an interval).

To derive the overall precision of an algorithm within the scope and during lifetime, the instantaneous precisions of the nodes have to be combined. The combined precision then has to be accumulated over time to arrive at a single value that characterizes precision. Common ways of combining instantaneous precision values of many nodes are maximum, average, and standard deviation. A variant often found in the literature is the maximum error after removing a given percentage (e.g., 5%) of the largest errors. The combined precision typically improves during the execution of an algorithm and approaches a stable value in the steady state. The combined precision in the steady state can be used to express the overall precision of an algorithm.

Requirements on the precision may heavily vary from application to application. This applies both to quality and quantity. With respect to quality, an application might require a certain average precision, other applications may request a certain maximum error. The requirements on the distribution of precision over the network and over time may also vary from application to application. With respect to quantity, required precision is closely related to the temporal frequency and spatial detail of the phenomena that require localization in spacetime. For localization in time, precision requirements range from a maximum error of few micro seconds (e.g., for controlling access to the communication channel) to seconds or even minutes (e.g., for activating a sensor network during certain times of the day). With respect to location, precision requirements range from a maximum error of some centimeters (e.g., locating a shooter [22]) to tens or even hundreds of meters (e.g., locating an animal herd).

As mentioned in Section 3.5, the scope of localization in sensor networks is often defined by a set of collocated sensor nodes that cooperate in monitoring a close-by event in the real world. For this kind of application, the precision among this set of collocated nodes typically must be high. However, the precision among nodes which are far apart in space may not be important. We will return to this issue in Sections 5.1.

3.7 Other Quality-of-Service Aspects

Besides the aspects discussed so far, a number of additional QoS characteristics of localization in spacetime are of practical relevance. Two prominent examples are robustness and security. A robust localization algorithm delivers correct location estimates even in the presence of well-defined, accidental failures. Another aspect is secure verification of location estimates, where spoofed locations can be detected (see, e.g., [4]).

4 Distributed Algorithms for Localization in Spacetime

Many practical distributed algorithms for localization in space (e.g., [1, 7, 14, 16, 20, 21]) and time (e.g., [5, 8, 13, 23, 24]) are based on a few common structural elements. In this section we point out these structural elements and discuss various concrete instances of these elements found in existing algorithms.

Consider Figure 4. Part (a) shows two kinds of nodes: black reference nodes with known locations and white client nodes with unknown locations. In part (b), a gray client node measures its distance Δ_i from a number of neighboring reference nodes. Using the location S_i of the references and the measured Δ_i, the gray node infers its own location in spacetime. The node can now also act as a reference for other client nodes in subsequent iterations of the algorithm as illustrated in part (c). Eventually, all nodes should be able to measure distances to a sufficient number of neigh-

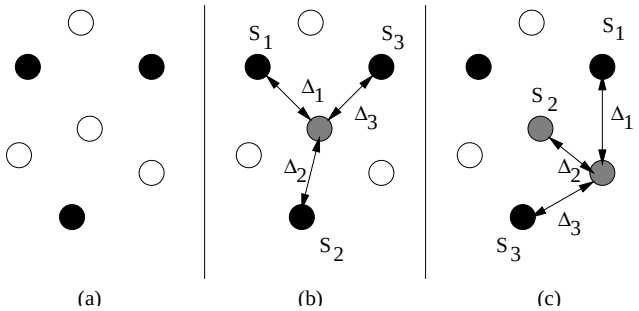

Figure 4: Client nodes infer their location in spacetime by measuring spatio-temporal relationships Δ (e.g., Euclidean distance, message delay) to black reference nodes with known locations S in spacetime. The process is iteratively applied. The figure shows from the left to the right, a sequence of three snapshots.

boring reference nodes in order to estimate their location in spacetime.

The meaning of the symbols Δ and S has to be interpreted in a rather broad sense here. S is any state information of a node that is relevant to the localization algorithm. Examples for S are time, location, orientation, and address of a node. S may also include confidence values that characterize the precision of the respective bits of state information. Δ is a spatio-temporal relationship between a client node and one or more reference nodes. Examples include Euclidean distance, hop distance, message delay, and angle with respect to the orientation of the client. Δ may also include confidence values.

A pair (S, Δ) can be interpreted as a *constraint* on the possible spacetime locations of a client node. For example, if S is a location of a reference node in space and Δ its Euclidean distance, then the location of the client node is constrained to the hull of a sphere with radius Δ centered at S. As we will show in Section 4.2, a constraint may also involve multiple reference nodes, such that Δ is a relationship among a client node and any number of reference nodes (e.g., client node is closer to reference 1 than to reference 2). Also, reference nodes need not be network neighbors of the client node.

A second structural element of localization algorithms is a procedure for *combining multiple constraints*. As pointed out above, a single constraint limits the possible locations of a client node, but the resulting solution space often does not satisfy precision requirements. Hence, multiple constraints have to be combined (e.g., intersected) to further cut down the solution space (e.g., to a single point in spacetime).

A third important component of localization algorithms are rules to *select constraints*. In dense networks with many reference nodes, there is a large set of possibilities for obtaining constraints that involve different sets of reference nodes. While a large number of constraints may result in very precise location estimates, the overhead for combining such numerous constraints may be prohibitive. Hence, the goal is to select a small number of tight constraints that is sufficient to achieve a certain precision. This selection process is not trivial, as it depends on a number of parameters such as the precision of the state information of the individual reference nodes, but also on a particular combination of reference nodes. Also, certain reference nodes may only become available after they have estimated there location themselves. Often, an overlay structure (e.g., spanning tree, clustering) is constructed to ease this selection process. For example, a client node may use its parent in a spanning tree as a reference node. Essentially, constraint selection can be interpreted as the approach an algorithm takes to structure localization in multi-hop networks (i.e., across space).

The fourth important element of localization algorithms is an approach to *maintain localization over time*, since a single estimate of a node's location in spacetime is quickly invalidated due to the progress of time and due to mobility. The conceptually simplest approach to this problem is to repeat a one-shot localization frequently.

Last but not least, a *bootstrapping mechanism* is needed to provide initial reference nodes that act as seeds for distributed localization algorithms.

A concrete algorithm for localization in spacetime can often be considered as a combination of concrete instances of the above five categories and additional "glue" elements. Many practical algorithms consist of several phases in order to improve precision or other performance metrics. In each phase, different instances of the five categories may be used. For example, several algorithms consist of a first phase to obtain rough location estimates for all nodes. In a second phase, the so-called refinement phase, these initial estimates are further improved.

In the following sections we will discuss the above structural elements in more detail.

4.1 Bootstrapping

Obtaining constraints typically requires a number of reference nodes with known locations in spacetime. Bootstrapping consists in providing such initial reference nodes with location estimates. The most commonly used approach to solve the bootstrapping problem is the provision of so-called anchor nodes which are able to estimate their locations by means of an out-of-band localization mechanism such as GPS. While anchors are a natural way to solve the bootstrapping problem and allow for good precision due to providing location "fixpoints" throughout large networks, they also come with a significant overhead: a certain portion of the nodes must be equipped with additional hardware (e.g., GPS receivers) and an additional infrastructure is often needed (e.g., GPS satellites). We will discuss issues with anchors in more detail in Section 5.1.

It is also possible to solve the bootstrapping problem without the use of anchors. Consider for example Figure 5, where three nodes 1, 2, and 3 with mutual Euclidean distances d_{12}, d_{23}, d_{13} are depicted. The nodes define a coordinate system as follows. The origin is given by the position of node 1. The positive x axis is given by a ray

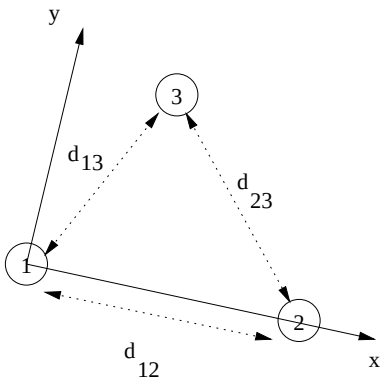

Figure 5: Three non-collinear nodes with known mutual distances d_{ij} define an unambiguous coordinate system for two-dimensional space.

starting at node node 1 passing through node 2. The positive y axis is given by a ray starting at node 1 that is orthogonal to the x axis and that extends into the half plane (defined by the x axis) that contains node 3. In this coordinate system, the coordinates of the three nodes are $(0, 0)$, $(0, d_{12})$, and $(X, \sqrt{d_{12}^2 - X^2})$, respectively, with $X = (d_{12}^2 + d_{13}^2 - d_{23}^2)/2d_{12}$ (e.g., [3, 17]).

Note that a coordinate system constructed this way is not unambiguous, any other coordinate system could have been used as well. Hence, in contrast to anchor-based approaches, anchor-free approaches are not suitable for external localization (cf. Section 3.1). Also, the coordinate system changes when one of the initial reference nodes moves, invalidating the location estimates of all nodes whose positions have been estimated with respect to this coordinate system. The precision of anchor-based algorithms is often superior to anchor-free approaches, since anchor nodes may be distributed over the network to act as fixpoints for localization. With anchor-free approaches, nodes far away from the reference nodes that define an initial coordinate system may experience significant imprecision due to accumulating errors.

4.2 Obtaining Constraints

The general form of a constraint is $(\{S_1, ..., S_N\}, \Delta)$, where N reference nodes and their respective state information S_i is involved. Δ represents a spatio-temporal relationship among these reference nodes and the client node. While S_i are typically retrieved from a reference node by means of a message exchange, Δ is usually a measured quantity. In most cases, Δ is either represented by a point estimate (e.g., distance = X) or by bounds (e.g., distance $> X$ and/or distance $< Y$).

For localization in time, the delay of network messages is typically used as a foundation for Δ measurements. A single message exchange from a reference to a client node can be used to define a lower bound (i.e., either 0 or a known minimum message delay) on the temporal distance between reference and client. A round-trip message exchange between client and reference can be used to derive lower and upper bounds on the temporal distance (e.g., [18]). The average of these bounds can be used as a point estimate of the temporal distance (e.g., [8, 24]). A broadcast message is usually received almost concurrently by a set of receivers. Hence, a broadcast message that is received by a reference node and a client node can be interpreted as a virtual message from the reference to the client with a negligible delay (e.g., [6]).

For localization in space, distance-dependent properties of propagating signals (e.g., sound, radio) such as received signal strength or time of flight are typically used as a foundation for Δ measurements. Two common forms of constraints are based on Euclidean distances (e.g., bounds or point estimates for the distance from a reference) and angles (e.g., bounds or point estimates for the direction of arrival of a signal from a reference). Rough distance bounds can be exploiting the fact that communicating nodes cannot be farther apart than the maximum communication range (e.g., [1]). Angle measurements typically require more complex hardware such as directional antennas or antenna arrays. Two common constraints that involve multiple references are "closer to" relationships (i.e., client is closer to reference 1 than to reference 2) and distance differences (i.e., client is X meters closer to reference 1 than to reference 2).

4.3 Combining Constraints

A single constraint can be interpreted as a region in spacetime that contains the location of a client. Combining multiple constraints typically consists of two steps. In a first step, "bad" constraints are eliminated from the set of available constraints. One example of such bad constraints are outliers that represent a region in spacetime that does not overlap with the regions defined by some or all other constraints. After this step, the remaining constraints should have a non-empty intersection that contains the prospective location estimate of the client.

In a second step, the intersection or a point in the intersection of the remaining constraints is computed. In many cases, this can be achieved analytically, for example by solving an equation system. In some cases, a closed-form solution cannot be derived or the computational overhead may be prohibitively high. An approximative solution that trades off computational overhead for memory is to subdivide the solution space into pixels, where each pixel has an associated bit that is set to "1" initially. For each available constraint, the pixels that are outside the region represented by the constraint are set to "0". Eventually, the remaining "1" pixels form an approximation of the intersection region (e.g., [10]).

Due to measurement errors, it may happen that there is no sufficiently large subset of constraints with a non-empty intersection. This is often the case if the constraints use Δ relationships that are point estimates (e.g., distance = X)

rather than bounds. Here, an optimization problem may be set up by requiring the solution point to minimize a certain error metric. A commonly used error metric is the distance between a point and a constraint, which is defined as the minimal distance to any point contained in the region defined by the constraint. A typical objective function for the optimization problem is then to minimize the sum of the squared distances between the solution point and each constraint.

For localization in time, constraints define regions that are either points or intervals. In case of intervals, the intersection interval can be computed as the maximum of the lower bounds and the minimum of the upper bounds. In case of point estimates, the average of all constraints is typically used, which minimizes the sum of the squared error distances. If confidence values are available for the constraints, a weighted average may be used instead (e.g., [20, 23]).

Let us consider some commonly used examples for constraint combination in algorithms for localization in space. A very simple approach is based on centroids (e.g., [1]), where multiple distance-bound constraints are given (i.e., distance from reference is at most X). Here, each constraint defines a sphere. A point close to the intersection of such a set of spheres can be obtained by computing the centroid of the locations of the according reference points.

Another commonly used approach is multilateration to combine multiple distance constraints, where the region defined by each such constraint can be interpreted as the hull of a sphere. Multilateration finds the intersection point of a set of at least 4 spheres in three-dimensional space. In case of exactly four spheres, a linear equation system can be derived and solved to find the intersection point. For more than 4 constraints a minimum-square-error optimization problem can be derived, also resulting in a linear equation system (e.g., [15, 20, 21]).

One further approach is based on triangle tests, where a node performs a check to see whether it is located inside the triangle formed by three reference nodes (e.g., [10]). Such a test is based on the following property: a node located outside the triangle can be moved such that the distances to all anchors are either all increased or all decreased simultaneously. In contrast, all movements of a node located inside the triangle will increase the distance to some anchors and decrease the distance to some anchors simultaneously. In dense networks, the following approximative triangle test can be used: a node is assumed to be in the triangle, if no neighbor of the node is further from or closer to all three anchors simultaneously. For this test, "closer to" constraints are used.

4.4 Selecting Constraints

At each point during the execution on a localization algorithm, a certain set of reference nodes are available for each client node. Using these reference nodes, a number of "good" constraints must be selected out of the large set of possible constraints. This selection is based on the quality of the state information of the reference nodes, on the quality of the spatio-temporal relationship and on temporal aspects. For example, a better set of references may become available in a future iteration. However, the algorithm might not be able to proceed if a node chooses to wait for better references to become available, since the node itself then cannot act as a reference for other nodes.

There are two different ways to approach this problem. *Structured* approaches first construct an overlay topology that controls selection of reference nodes and triggers client nodes to start measurements. A common overlay topology are trees. For each anchor, a spanning tree of the network is constructed with the anchor at the root. A client node becomes active as soon as its parent has estimated its location and can thus act as a reference (e.g., [5, 8, 24]). Another typical overlay topology are clusters, where nodes in a cluster establish a local coordinate system and estimate their locations in terms of this reference grid. Adjacent clusters must share a number of nodes to allow for the derivation of a coordinate transformation between these clusters (e.g., [3, 6]).

While such structured approaches ease the selection of reference nodes, there is an additional overhead for constructing and maintaining the overlay topology. For example, if nodes fail or move, the overlay topology has to be updated to reflect this change. In contrast, *unstructured* approaches do not explicitly construct an overlay topology (e.g., [13, 21]). Instead, each node actively monitors its neighborhood for a sufficient set of references to become available. While this approach avoids the overheads of topology construction, it introduces an overhead due to a potentially large number of constraints.

Approaches for localization in time often use structured approaches, since a small number of constraints is usually sufficient to achieve the requested level of precision. With localization in space, significant measurement errors and a high degree of freedom due to the three dimensions of space typically requires the use of many constraints. Hence, many approaches for localization in space are unstructured.

4.5 Maintaining Localization over Time

A single run of a localization algorithm allows each node to estimate its location in spacetime at a certain point in time. However, as time progresses, the precision of this one-shot estimate may decrease quickly due to node mobility or due to the progress of time. Obviously, the algorithm can be executed one more time to obtain up-to-date estimates. The resulting precision over time then depends on the frequency of execution. However, since each execution of the algorithm takes a certain amount of time, this frequency cannot be arbitrarily increased. Hence, the maximum precision over time is also limited. Alternatively, if a certain target precision is requested by the application, the execution fre-

quency may be calculated to be just high enough to provide the requested precision (see, e.g., [24]). For localization in space it is also possible to limit re-execution to nodes that have changed their location (e.g., [20]) in the meantime.

One way to further improve precision over time is the use of sensors to measure the location in spacetime locally without referring to other nodes. This technique is also known as *dead reckoning*. Hardware clocks, for example, are dead-reckoning devices for estimating the current time. Accelerometers may be used to measure movements and can hence provide estimates of the current position in space (see, e.g., [25]). However, dead-reckoning devices typically suffer from significant errors that accumulate over time and can therefore only be used to bridge the short gap between two consecutive runs of a localization algorithm. For example, typical hardware clocks suffer from an unknown clock drift between 10 and 100 parts per million. After one minute, the deviation from real time is then between 0.6 and 6 milliseconds. For location estimation using accelerometers, there is a quadratic relationship between acceleration-measurement errors and errors in the computed location estimate.

Another way of improving the precision is *prediction*, where based on location estimates from the past a current estimate is computed. Besides the past behavior, prediction requires a model of how a node can move through spacetime. With respect to time, such a model is rather simple as real-time progresses at a constant rate. The situation gets more complicated for space, where nodes can move in complex patterns. However, it is often possible to derive constraints on the possible locations (e.g., only on roads), bounds on speed and acceleration. For example, if there is an upper bound on the speed of a node, we can derive bounds on the possible locations of a node at time t_1 given the node's location at time $t_0 < t_1$. Technically, prediction can be achieved by fitting a curve (often a polynomial with low degree) to a set of locations in spacetime observed in the recent past. As with dead reckoning techniques, prediction often experiences significant errors which are mainly subject to the accuracy of the used model.

Many time synchronization algorithms (e.g., [6, 13]) use a combination of prediction and dead reckoning. Basically, the hardware clock (i.e., a dead reckoning device) is used to implement progression of time. However, due to clock drift hardware clocks may run faster or slower than "synchronized time". To compensate this clock drift, the relative rate difference between synchronized time and the hardware clock in the recent past is used to predict the rate difference in the future. The predicted rate difference is then used to compensate the error due to a wrong rate of the hardware clock. Common approaches to implement such a rate compensation are linear regression and phase-locked loops. With linear regression, a line is fitted to to a set of time estimates from the past, where the slope of the line characterizes the rate difference. With phase-locked loops, the phase offsets between synchronized time and hardware clock are integrated over time and used to control the frequency of the hardware oscillator.

4.6 Examples

We conclude this section by giving some examples for localization in space and time in sensor networks from the literature. We present two anchor-based algorithms and two anchor-free algorithms, in each case one for localization in time and one for localization in space. Please note the similarities among each pair of algorithms.

4.6.1 Anchor-Based Algorithms

Flooding Time-Synchronization Protocol [13]. The node with the lowest node ID is elected as the anchor whose local time serves as a reference time for synchronization. If this node fails, then the node with the lowest ID in the remaining network is elected as the new anchor. The anchor periodically broadcasts a synchronization message that contains its current local time. Nodes which have not received this message yet use the message contents to derive a constraint and broadcast the message to its neighbors. Each node collects eight such constraints and uses linear regression on these eight data points to estimate time offset and rate difference to the anchor. The algorithm is repeatedly executed to maintain synchronization over time.

Ad Hoc Positioning System [15]. It is assumed that a fraction of the nodes are anchors with known locations. The algorithm supports several localization methods, we will sketch the method "dv-distance" here, where nodes must be able to measure distances to neighbors in order to derive a distance constraint. Anchors regularly broadcast a message containing their location. In addition, this message contains a field that holds an estimate of the distance from the anchor which is initialized to zero. Nodes which have not received this message yet measure the distance to the sender, which is then added to the distance field in the message to obtain an estimate of the node's distance from the anchor. Then, the message is broadcast to the neighbors of the nodes. Eventually, each node will end up with distance estimates for at least 4 anchors and uses multilateration to compute its location. If a node changes its location, it listens to the broadcasts again in order to update its location estimate.

4.6.2 Anchor-Free Algorithms

Reference-Broadcast Synchronization [6]. This algorithm denotes some nodes as beacon nodes which frequently broadcast messages to a set of client nodes that should be synchronized (cf. Section 4.2). The clients that receive such a broadcast then exchange their respective reception times to obtain mutual constraints. Each client collects multiple such constraints and uses linear regression to compute relative time offsets and rate differences to the

other client nodes. The offset and rate difference between a pair of client nodes defines a coordinate transformation between the local time scales (i.e., coordinate systems) of these nodes. To extend this scheme to multi-hop networks, the network is clustered such that a single beacon can synchronize all nodes in its cluster. Gateway nodes that participate in two or more clusters independently take part in the reference-broadcast procedure of all their clusters. By knowing offsets and rate differences to nodes in all adjacent clusters, gateway nodes can compute coordinate transformations between all adjacent clusters. Time synchronization across multiple hops is then provided by transforming clock readings between the local time scales (i.e., coordinate systems) of the nodes.

Self-Positioning Algorithm [3]. In the first phase of this algorithm, each node measures distances to its neighbors and broadcasts these distances to its neighbors. After this, each node knows the distance to each of its neighbors and the distances between some pairs of its neighbors. Then each node constructs a local coordinate system using two of its neighbor nodes (cf. Section 4.1). In the second phase, coordinate transformations are computed between the coordinate systems of adjacent nodes. In the third phase, a global coordinate system is selected and coordinate transformations are computed to transform the local coordinate systems to this global system. For this, a set of nodes called the Location Reference Group (LRG) is elected such that the degree of mobility of the centroid of these nodes is small in order to avoid frequent adjustments of the global coordinate system. The global coordinate system is then defined by the average of the local coordinate systems of the nodes in the LRG (i.e., origin is centroid of the origins of the individual coordinate systems, axis vectors are averages of the axis vectors of the individual coordinate systems).

5 Limitations and Trade-offs

Sensor networks are subject to various challenges that have to be met by algorithms for localization in spacetime. In the following paragraphs we briefly summarize these challenges, before discussing various trade-offs and limitations of algorithms with respect to these challenges.

Infrastructure. In many applications, sensor networks have to be deployed in remote, unexploited, or hostile regions. Sensor networks therefore often cannot rely on sophisticated hardware infrastructure. For example, under dense foliage or inside buildings, GPS cannot be used since there is no free line of sight to the GPS satellites.

Energy and Other Resources. Sensor-network applications often require that sensor nodes be small and cheap. This has a number of important implications. First of all, the amount of energy that can be stored in or scavenged by small devices is typically very limited due to the low energy density of currently available and foreseeable technology. To ensure longevity despite this limited energy budget, energy-efficient design both in hardware and software becomes a dominating goal. Additionally, computing, storage, and communication capabilities of individual sensor nodes are rather limited due to size and energy constraints.

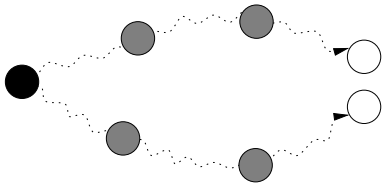

Figure 6: Collocated nodes (white) may end up with a large relative error due to using different chains of reference nodes (gray).

Network Dynamics. Due to their deployment in the physical environment, sensor networks are subject to a high degree of network dynamics. Sensor nodes can be mobile (e.g., [11]), die due to depleted batteries or due to environmental influences, and new sensor nodes may be added at any point in time. This results in frequent changes in the network topology and in temporary network partitions. Mobile nodes can transport messages across partition boundaries by storing a received message and forwarding it as soon as a new partition is entered. The end-to-end delay of such message paths is very unstable and hard to predict.

Configuration. After initial deployment, it is often infeasible to physically access the sensor nodes for hardware or software maintenance. The large number of nodes also precludes manual configuration of individual nodes. While traditional networks such as the Internet do also consist of a large number of nodes, there are also many human network administrators who each care for a manageable number of computers. With sensor networks, however, a small number of human operators, if any, may be responsible for thousands or even millions of sensor nodes.

5.1 Anchor Infrastructure

The number, distribution, and arrangement of anchor nodes in a network is a key parameter for the performance of an anchor-based localization algorithm. In this section, we discuss various aspects of such an anchor infrastructure.

In order to obtain precise location estimates, the anchors must define an unambiguous coordinate system at the least. With respect to time, the local time scale of any single node defines such an unambiguous time coordinate system. For space, at least four anchors are required. Three anchors typically result in two possible coordinate systems, but one of them can often be excluded due global constraints about possible locations of nodes.

However, such a minimum number of anchors is often not sufficient. Energy considerations and interference issues often limit the effective range of anchors. With radio communication, for example, the energy consumption grows with range to the power of k, where typically $2 \leq k \leq 4$. Hence, in large networks with small anchor range, typically a significant portion of nodes cannot directly obtain constraints for a sufficient number of anchor nodes. In this case an iterative approach can be applied, where nodes first estimate their locations using anchors and then act as "secondary" references for other nodes. As measurement errors accumulate along such chains, the error in the estimated location is the larger, the more iterations are required (i.e., the larger the distance to the anchor is). Depending on the precision of the Δ_i, this error can be significant. With some approaches for measuring Δ_i used in practice (e.g., measuring Euclidean distance based on received radio signal strength), the error can be as high as 50% in realistic settings [20].

One particular problem with using a minimum number of anchors is that collocated nodes may end up with large relative errors due to using different chains of reference nodes as depicted in Figure 6. This can be problematic, since collocated sensor nodes often cooperate in observing a nearby physical event and thus may need a very small relative error. For example, estimates of the distance between the collocated nodes may include significant errors if the nodes use different paths. Local refinement procedures as described in [20] can somewhat improve the local consistency.

To achieve a reasonable precision, typically a large number of anchors is required, such that the maximum distance of any client node from a sufficient number of anchors is small. In [20], between 5% and 10% of all nodes, and in [21], between 10% and 20% of all nodes are anchors. However, a large number of anchors requires an out-of-band mechanism for providing the anchors with precise location estimates. Such an out-of-band mechanism may be a serious drawback, since it typically implies additional hardware infrastructure, and special hardware must be attached to the sensor nodes. One typical example for such an out-of-band mechanism is GPS with its satellite infrastructure and resulting constraints, where anchor nodes must be equipped with expensive and energy-intensive GPS receivers.

In order to ensure that each node in the network has a sufficient number of neighbors which can act as anchors, the network must have a certain minimum density. In other words, each node in the network must have a certain minimum number of neighbors. This also implies that nodes at the edge of the network (with a lower number of neighbors) typically experience a reduced precision. Network density is particularly important if the collected constraints are loose, since then many constraints are needed to achieve precise location estimates. In [20], for example, each node has an average of 7 to 12 neighbors in order to achieve a reasonable precision.

The arrangement of the anchors is also of importance for the achieved precision of localization in spacetime. Obviously, anchors should be evenly distributed across the sensor network in order to ensure that any node has a sufficient number of anchors in its vicinity. However, also the relative arrangement of anchors with respect to each other has an influence on the localization accuracy. For example, collinear anchors (i.e., anchor nodes that fall on a line in 3D space) and also approximately collinear anchors result in significantly reduced precision for localization in space. This is illustrated for localization in 2D space in Figure 7. In part (a), the error e_2 in distance measurement results in a small error in the estimated location (dotted circle) w.r.t. the actual location of the node (solid circle). In part (b), where anchor nodes are almost collinear (i.e., fall on a point in 2D space), the same error e_2 results in a much larger error in the estimated location.

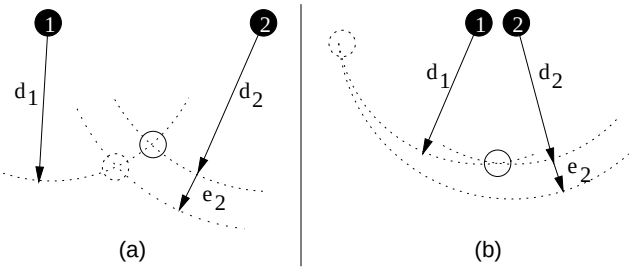

Figure 7: Localization error is much larger if anchor nodes are almost collinear.

In summary, a minimum set of anchors may lead to poor precision for iterative algorithms in large networks. A reasonable precision typically requires a much larger number of anchor nodes and an out-of-band mechanism for bootstrapping anchor locations with high precision. The actual deployment of the anchor nodes requires precautions to avoid collinearity. Depending on the quality of the constraints, networks must have a certain minimum density in order to achieve a reasonable precision.

5.2 Energy and Other Resources

As noted in Section 3.5, it is quite common that applications do only require a very limited scope and lifetime of localization, where actual scope and lifetime requirements depend on the occurrence of events in the physical environment. Hence, a significant amount of resources and energy could be saved if localization is only performed where and when needed and with the required precision.

However, algorithms for localization in spacetime are often not well suited for on-demand localization. This is due to two main reasons. Firstly, localization of a single node typically requires the cooperation of many other nodes to act as references for obtaining constraints. For on-demand localization, (recursively) providing a sufficient number of reference nodes on-demand would be needed. However, managing this process is a complex task. For example, as noted in the previous section, the number and relative arrangement of the anchors must be considered by such a

mechanisms, as this is crucial for the achieved precision. Moreover, such selective localization may induce significant management overheads.

Secondly, many algorithms require a significant amount of convergence time for achieving the requested precision. For example, [13] reports a convergence time of 10 minutes in a network of only few tens of nodes. Hence, if a node requests localization, a significant amount of time will elapse before a location estimate with sufficient precision can be provided.

5.3 Network Dynamics

At the beginning of Section 5 we mentioned the various effects of network dynamics found in sensor networks. These effects may have a significant impact on the performance and applicability of localization algorithms.

An important implicit assumption of many localization algorithms is that *before* the location of a node can be estimated, the node must obtain constraints involving a sufficient number of reference nodes. These references in turn must also be able to obtain constraints from a sufficient number of other reference nodes, and so on. Overall, in order to locate a node, there must be "constraint paths" from a client node to a sufficient number of initial references. This typically means that a sufficient portion of the network must be connected before localization can be performed.

However, a number of application projects (e.g., [11]) explore settings, where sensor nodes are mobile and network connectivity is sporadic. Such a situation is illustrated in Figure 8. Nodes 1 and 2 collect sensor readings while being disconnected from the network. After some time, node 2 sends its collected data to node 3 (e.g., a mobile base station as in [11]) while in communication range. Later on, when node 2 has already left communication range of node 3, node 1 sends its collected data to node 3 while in communication range. Node 3 now is to fuse data from nodes 1 and 2, which requires the locations in spacetime of nodes 1 and 2 while they collected the data. Note that there is no network connection between nodes 1 and 2 at any point in time. In such settings, many localization algorithms may not be applicable due to the above implicit connectivity assumption. This problem is addressed, for example, in [17, 18].

As noted in the previous section, many algorithms take a significant amount of convergence time before delivering the requested precision. It is typically assumed that the network remains stable during the execution of the algorithm. However, node mobility and other effects of network dynamics may invalidate this assumption. This may lead to significantly increased convergence times (see, e.g., [23]), or may also prevent the algorithm from converging at all. In the latter case, the effective precision may reduce significantly.

Some algorithms construct an explicit overlay topology as noted in Section 4.4. Network dynamics may break these topologies or make them inefficient. Maintaining these topologies under a high degree of network dynamics may become an unacceptable resource overhead.

5.4 Configuration

Localization algorithms may require a number of configuration parameters whose values differ from node to node. Typical examples for these parameters are the set of reference nodes to use, network link calibration parameters (e.g., minimum delay of a wireless link), and sensor calibration parameters (e.g., for distance measurements).

While some of these parameters can be automatically configured and adapted, this is not so easy for other parameters. For example, calibration may be a particularly tricky issue in sensor networks, because typical low-cost sensors used on sensor nodes are very sensitive to environmental parameters such as temperature and humidity. If these sensors are exposed to a harsh and dynamic physical environment, the output of the sensors includes significant errors. Additionally, sensor orientation, wear, and dirt lead to systematic but dynamically changing errors. In [26], for example, the authors observed an average error of approx. 75% for distance measurements with uncalibrated sensors based on time of flight of an ultrasound signal. Hence, calibration parameters often cannot be statically configured, but must be dynamically updated to reflect the changing setup.

6 Conclusion

In this paper, we showed that space and time are closely related in the context of sensor networks: in terms of applications, models, requirements, basic approaches, and in terms of concrete algorithmic techniques.

To even go a step further, we can consider localization in both time and space as *sensor calibration* problems. A sensor is a physical device that takes a certain physical quantity as input and produces a variable electrical signal as output that is usually converted to a digital number using an analog-to-digital converter. Calibration then consists of enforcing a certain mapping between the observed physical quantity and the sensor output. For example, if a sensor is exposed to a temperature of T°C, then the sensor should output T. Localization in space can then be considered as the task of calibrating a location sensor to a given coordinate system. Localization in time can likewise be considered as the task of calibrating a time sensor to a given coordinate system. Note that hardware clocks can be considered as sensors that measure the period of a physical process with periodic behavior (e.g., an oscillating quartz).

It would be worthwhile to examine the use of known techniques from time synchronization and node localization in the more general context of calibration. It particular, it is quite likely that many of the observations in this paper could be generalized to calibration. For example, the classification in Section 3 can be directly transferred to calibration. It is also thinkable that distributed calibration algorithms will

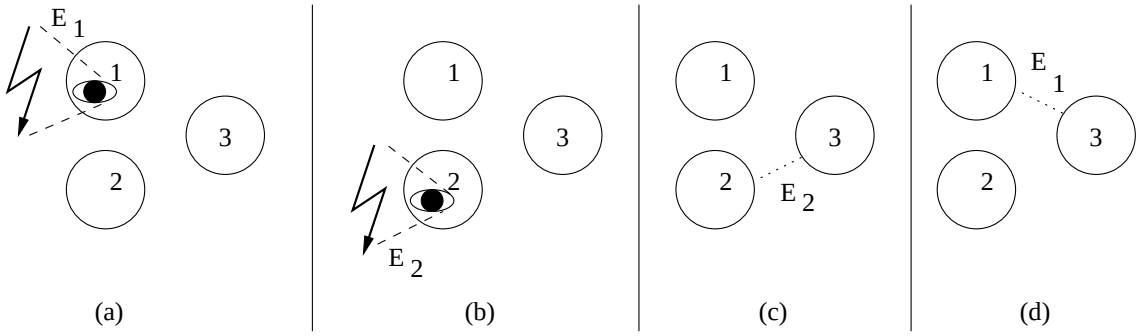

Figure 8: Message transport across partition boundaries. (a-b) Sensor nodes 1 and 2 collect sensor readings while disconnected from the network. (c-d) Later, sensor nodes 1 and 2 report their findings to node 3 for data fusion. At no point in time there is a network connection between node 1 and 2.

consist of structural elements similar to those we identified in Section 4.

References

[1] N. Bulusu, J. Heideman, and D. Estrin. GPS-less Low Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications*, 7(5):28–34, October 2000.

[2] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. A Collaborative Approach to In-Place Sensor Calibration. In *IPSN*, Palo Alto, USA, April 2003.

[3] S. Capkun, M. Hamdi, and J. P. Hubaux. GPS-free Positioning in Mobile Ad Hoc Networks. In *34th International Conference on System Sciences*, Hawaii, January 2001.

[4] S. Capkun and J. P. Hubaux. Secure positioning of wireless devices with application to sensor networks. In *Infocom*, Miami, USA, March 2005.

[5] H. Dai and R. Han. Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(1):125–139, January 2004.

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

[7] L. Evers, S. Dulman, and P. Havinga. A distributed precision based localization algorithm for ad-hoc networks. In *PERVASIVE 2004*, pages 269–286, Vienna, Austria, April 2004.

[8] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[9] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with Irregular Spatio-Temporal Sampling in Sensor Networks. *SIGCOMM Computer Communication Review*, 34(1):125–130, 2004.

[10] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Mobicom*, San Diego, USA, September 2003.

[11] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proc. ASPLOS X*, San Jose, USA, October 2002.

[12] B. Liskov. Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing*, 6(4):211–219, 1993.

[13] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *SenSys*, Baltimore, USA, November 2004.

[14] A. Nasipuri and K. Li. A directionality based location discovery scheme for wireless sensor networks. In *WSNA*, Atlanta, USA, September 2002.

[15] D. Niculescu and B. Nath. Ad hoc positioning system (aps). In *GLOBECOM*, San Antonio, USA, November 2001.

[16] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Mobicom 2000*, Boston, USA, August 2000.

[17] H. Ritter, J. Schiller, and T. Voigt. Demand-based Location Determination in Wireless Sensor Networks. In *Adjuct Proc. EWSN 2004*, Berlin, Germany, January 2004.

[18] K. Römer. Time Synchronization in Ad Hoc Networks. In *MobiHoc 2001*, Long Beach, USA, October 2001.

[19] K. Römer. Temporal message ordering in wireless sensor networks. In *IFIP Mediterranean Workshop on Ad-Hoc Networks*, pages 131–142, Madhia, Tunisia, June 2003.

[20] C. Savarese, J. M. Rabaey, and K. Langendoen. Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks. In *USENIX Annual Technical Conference*, Monterey, USA, June 2002.

[21] A. Savvides, C. C. Han, and M. Srivastava. Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. In *Mobicom 2001*, Rome, Italy, July 2001.

[22] G. Simon, A. Ledezczi, and M. Maroti. Sensor Network-Based Countersniper System. In *Proc. SenSys*, Baltimore, USA, November 2004.

[23] W. Su and I. F. Akyildiz. Time-diffusion synchronization protocol for sensor networks. *IEEE/ACM Transactions on Networking*, 2004. To appear.

[24] J. v. Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *WSNA*, Atlanta, USA, September 2003.

[25] E. Vildjiounaite, E. J. Malm, J. Kaartinen, and P. Alahuhta. Location Estimation Indoors by Means of Small Computing Power Devices, Accelerometers, Magnetic Sensors, and Map Knowledge. In *PERVASIVE 2002*, Zurich, Switzerland, August 2002.

[26] K. Whitehouse and D. Culler. Calibration as Parameter Estimation in Sensor Networks. In *WSNA*, Atlanta, USA, September 2002.

[27] Y. Xu, J. Heidemann, and D. Estrin. Geography-Informed Energy Conservation for Ad-Hoc Routing. In *MobiCom*, Rome, Italy, July 2001.

[28] 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.