

A Survey on Data Processing Issues in Wireless Sensor Networks for Enterprise Information Infrastructure

Kotagiri Ramamohanarao, Lars Kulik, Selvakennedy Selvadurai,
Bernhard Scholz, Uwe Roehm, Egemen Tanin,
Anastasios Viglas, Albert Zomaya, Chris Leckie

May 7, 2006

1 Introduction

Wireless sensor networks (WSNs) are a key technology for a broad range of applications such as environmental and habitat monitoring, traffic control, health monitoring, supply-chain management, smart homes, security, and surveillance systems [16]. A WSN is collection of a large number of inexpensive devices capable of sensing certain physical phenomena, carrying out simple processing tasks, as well as communicating with each other using wireless networking technology in an ad-hoc manner.

As the evolution of sensors follows Moore's law, sensors become smaller, cheaper, and more powerful. This evolution enables the deployment of systems that can consist of hundreds or thousands of sensor nodes. Typically, sensor networks are deployed to gather physical information in a robust and autonomous manner. The data collection can be either continuous or selective, i.e., detect selected events of interest.

Since sensor nodes are small battery-powered devices that are deployed in potentially inaccessible environments, energy efficiency is of paramount importance. Communication, in particular, is an expensive operation for sensor nodes, and is significantly more expensive than data processing. Although the energy efficiency of sensor nodes has certainly improved over recent years with new hardware developments (such as more powerful processors or larger memory), the main issues in wireless sensor networking remain as:

- limited battery power,
- limited memory and CPU resources,
- unreliable, short-range wireless networking, and

- environmental influences on networking and sensing quality.

There are three types of sensor hardware platforms commonly mentioned in the literature, namely *micro-scale sensing devices*, *generic motes*, and *powerful sensing devices*. Micro-sensing devices are tiny almost invisible devices such as SmartDust [74]. These are in the early stages of their development and currently not available from any industry vendors. Devices such as RFID tags are sometimes mentioned as the early incarnations of future micro-sensing devices. However, RFID tags are commonly passive devices. Algorithms and data structures that are discussed in this survey cannot be easily used with RFID tags. Generic sensors, often referred to as *motes*, are available from multiple vendors such as Crossbow and Telos. Most research activities that aim at information processing in WSNs focus on generic sensor devices. Powerful sensing devices are comparable to existing personal computing devices such as PDAs or even PCs. They can serve as gateways for generic sensor nodes or can execute high-bandwidth sensing tasks requiring more processing power than what is available from generic motes. In this survey, the generic sensor node class forms our main focus.

A number of research questions such as energy-efficient networking protocols or low-level programming of motes are already relatively well understood. With TinyOS [87], there is a widely used operating system available on a number of sensor platforms. The development of Sun's J2ME and recently released Squawk will facilitate a wider distribution of standard development platforms for WSNs. We focus, in this report, on higher-level aspects of WSN programming.

Higher-level aspects of WSNs such as efficient in-network processing, programming environments and intrusion detection are key components for enterprise information infrastructures. In-network query and data processing allows to leverage the potential of WSNs to a higher abstraction level. It gives application developers the ability to gather and retrieve information from a WSN using declarative queries instead of low-level programming. Query languages for WSNs support the access and description of information in a sensor network in an *ad-hoc* fashion, enable periodic updates, and allow querying for significant events in the network.

Programming environments facilitate the development, deployment, and execution of applications in WSNs. The state of the art for node-level programming languages is nesC [38]. However, nesC makes the development of applications very tedious for large-scale networks, and the design of higher-level programming models and their efficient implementation remains a major challenge for future research.

Although we are mainly focusing on the in-network processing and programming environment issues, there are many other areas that need to be addressed to realize a more robust technology. Some of these topics are better hardware platforms, efficient localization algorithms, topology management and control, link-aware medium access control protocols and resilient routing protocols. These issues however are out of the scope of this survey for enterprise information infrastructures and thus, not reviewed here.

The remainder of this report is structured into four sections. In the following section, we survey current research directions in in-network processing. Different data gathering protocols are described together with data aggregation techniques. Section 3 provides a review of programming environments for WSNs. A concise overview of unique security aspects of WSNs is given in Section 4. We present likely future directions in the final section.

2 In-Network Processing

In-network processing is widely accepted as a paradigm for energy-efficient data gathering in WSNs. As energy resources and radio ranges are often limited in WSNs, nodes employ multihop routing to report data to gathering node(s). A node that issues a query and collects data is often called a sink. Although it is possible to increase the communication range r of individual nodes, the signal amplitude drops proportional to $1/r^\alpha$, where α is ranging between 2 and 5. Therefore, the energy requirement is prohibitive for larger distances, and nodes use multihop routing to report data to a sink.

Aggregation algorithms for data collection in sensor networks are based on the insight that a sensor node consumes less energy for information processing than for information communication. There are two ways to reduce the communication cost of sensed data: *packet merging* and *partial aggregation*. The communication cost in wireless networks is determined by the number of packets a node has to transmit. Each packet has a fixed size and the size of a sensor record is typically much smaller. Packet merging [93] combines several smaller records into a few larger ones and thus reduces the number of packets and thereby the communication cost. Partial aggregation computes intermediate results at the node level such as the sum or the average of sensor readings. This reduces the need for communication considerably: instead of transmitting the sensor records of each individual node separately to a data center, a node first aggregates the incoming records of the nodes in communication range and then communicates the partial result, such as the current average of sensor readings, to the next node, which in turn aggregates this partial result with its own readings. Since the energy cost for processing information is small compared to the cost for communication, aggregation leads to considerable energy savings.

Aggregation algorithms exploit the fact that there are different types of correlations in sensor readings: spatial correlations, temporal correlations, and data correlations. Spatial correlation is a result of the fact that adjacent sensor readings are typically similar. Sensor readings are temporally correlated if the monitored physical feature has a small variability with respect to the data sampling frequency. Sensor nodes can exhibit data correlations in their sensed data due to their overlapping sensing coverage. When the correlated data is exploited within the network, the sink gathers snapshots of reduced signal data values measured at the sensor nodes, and uses interpolation to derive the signal value at other points in the moni-

tored region.

2.1 Design issues in data gathering

The issues related to the design of a data gathering algorithm is controlled by the following factors:

- Data aggregation
- Number of transmissions
- Data compression
- Topology control

The most critical design issues to affect data gathering is the formation and maintenance of a data aggregation tree. This tree is usually mapped closer to a minimum spanning tree in order to reduce the number of bits transmitted in the network. The formation of this aggregation tree allows *opportunistic* aggregation at appropriate intermediate nodes. In some proposals, any number of values could be aggregated into a single data value, typically uninformed by the correlation structure.

If approximate results are acceptable, reducing the number of transmissions could result in improved energy efficiency and lifetime. Controlling the number of raw samples collected from each node is another design decision that affects the efficiency of data gathering. In order to decide on the number of necessary transmissions, we should determine the correlation graph of the nodes in an efficient and possibly in a distributed manner. Such a graph allows the capture of spatial correlation characteristic. The number of transmissions might also be reduced by exploiting the temporal correlation observed by each node independently through some memory of their past observed values.

When based on the data compression approach, such algorithms focus on reducing the number of bits transmitted to the sink using suitable coding techniques such as single-input or multi-input coding [81]. Using a single-coding strategy, a node encodes its information based on the data of one other nearby neighbor node as part of the routing tree structure toward the sink. On the contrary, multi-coding strategies use all input information from multiple nodes to exploit correlation among several nodes before encoding.

Finally, the use of a topology control scheme, which allows the formation of a spanner, ensures redundant nodes are made to sleep. The redundancy is of the physical connectivity perspective, rather than the application perspective. However, a correlation-informed topology control scheme might allow the total exploitation of both objectives. Thus, using a correlation graph, the redundant nodes are identified and made to sleep while ensuring the network is not partitioned.

Accordingly, we could classify the data gathering schemes in the literature into four categories, namely compression-induced schemes, transmission-controlled

schemes, aggregation-tree schemes and topology control schemes. In the following, we describe some of the commonly cited proposals in the first three categories, as topology control schemes generally do not consider the correlation structure. As an alternative to data gathering, storage of data and creation of rendezvous points for data collection by later queries can be considered. We discuss this approach as a separate subsection.

2.2 Gathering with data aggregation

A number of research projects have studied *in-network* query processing algorithms for wireless sensor networks: Cougar from Cornell University, TAG and TinyDB from UC Berkeley, REED from MIT, and Directed Diffusion from UCLA. In contrast to *server-based* approaches (also referred to as *warehouse approaches*), where all sensor data are gathered from the nodes and sent to the sink for processing, in-network aggregation approaches focus on where, when, and how data is aggregated in the sensor network.

Due to the limitations of the computational and memory resources on sensors, no high-level data models are applied, but rather data is typically modeled as a sequence of simple data values. The most common approach is to use a relation-based data model where sensor data is accessed through a single *sensors* table with one column per sensor reading that is horizontally partitioned over all nodes. Conceptually, the *sensors* table is a ‘virtual’, unbounded table: the sensor readings are not physically stored, but rather the data items are transient tuples in a *data stream*.

The view that a sensor network can be seen as a huge distributed database system is proposed by the Cougar system [9, 24, 92, 93]. Cougar offers an SQL-like query interface that abstracts from the physical network layer of the sensor network. Cougar enhances the standard SQL by *DURATION* and *EVERY* clauses [93], which specify the lifetime of a query and the rate of the query answers, respectively. Efficient data collection in Cougar is based on the idea that local computations can reduce unnecessary traffic on the sensor network. Therefore, Cougar uses a query proxy layer in between the network and application layer that sits on each sensor node. A query optimizer generates query plans that determine which nodes are involved in routing (the data flow) and which computations need to be performed at which nodes needs. Each query has a leader node that performs the main computation. Thus, two computation plans are generated for a given query: one plan for the leader node and one plan is disseminated to all sensor nodes that have to participate for a given query. Although, the query plan can ensure an energy-efficient data collection process, the query proxy layer will introduce additional overhead to each sensor node.

COUGAR allows a more object-relational model in that it models sensors as hierarchical Abstract Data Types (ADT) with associated methods [9]. It also assumes a central *sensors* table, but rather than separate columns for each sensor reading, this table encapsulates each sensor as an ADT with appropriate methods

to access the sensor data or to detect events [9].

2.2.1 Tree-based aggregation

Similar to Cougar, TinyDB [65] provides a declarative interface to query an ad-hoc sensor network that supports periodical data acquisition from sensors.

In traditional DBMS, all operators are pull-based: an operator requests data from its children in the plan only when needed. In contrast, in sensor networks data is commonly pushed through the system by the sources. One approach pursued in TinyDB [65] is to connect operators with queues, allowing sources to push data into the queues. However, queues are very limited in size on sensors, so operators must be scheduled dynamically to process the queues in order to minimize their sizes, queue delays and overflows.

In addition to the standard SQL operators AVERAGE, COUNT, MIN, MAX, and SUM, TinyDB offers two further operators: HISTOGRAM and MEDIAN. TinyDB supports in-network query processing using a generic aggregation service called TAG (Tiny AGgregation) [63]. Aggregation in TAG is implemented by a merging function f , an initializer i and an evaluator e . If $\langle x \rangle$ and $\langle y \rangle$ are (multivalued) partial state records, then the resulting partial state record $\langle z \rangle$ can be computed via $f: \langle z \rangle = f(\langle x \rangle, \langle y \rangle)$. The initializer i specifies how to initialize the state record for a single sensor value. The merging function for AVERAGE is given by partial state records $\langle S, C \rangle$ for SUM and the COUNT. If $\langle S_1, C_1 \rangle$ and $\langle S_2, C_2 \rangle$ are two partial state records, then f is given by: $f(\langle S_1, C_1 \rangle, \langle S_2, C_2 \rangle) = \langle S_1 + S_2, C_1 + C_2 \rangle$. The evaluator e computes the actual value of the aggregate. In the case of the AVERAGE operator the evaluator is $e(\langle S, C \rangle) = S/C$.

Some recent projects also introduced the *join* operator into in-network query processors and looked at multiple queries. The REED project from MIT [1] extends the TinyDB query processor with facilities for efficient multi-predicate filtering and a join operator. [88] considers multi-query optimization for aggregate queries on sensor networks, and propose a set of distributed algorithms for processing multiple queries with minimum communication while observing the computational limitations of the sensor nodes.

TAG is an example of a tree-based in-network aggregation algorithm. To gather data within the sensor network, one node, the sink, is appointed to be the root. The root initiates a broadcast sending its ID and its level. All nodes that receive this message and do not have a level assigned, determine their own level as the level in the message plus one. The ID within the message determines the parent for the node receiving this message. The broadcasting of ID and level continues until all nodes have assigned a level and a parent. For a network without loss and in which all nodes participate for a given query, the resulting tree is close to an optimal solution.

In order to optimize the data aggregation process, TinyDB introduces the concept of a Semantic Routing Tree (SRT) [64]. For a given query, only a few nodes typically have to respond. An SRT is an index over a fixed attribute, for example the

temperature sensed by the network, where each parent maintains the range of all values of its children for the attribute A. When a query for the predicate is received by a parent, it forwards the query only when at least one child satisfies the predicate. An SRT optimizes the query forwarding phase of TAG and greatly reduces the number of broadcasts required to reach all of the nodes that satisfy the query. However, the maintenance cost of SRT exceeds its benefit if it has to be maintained for varying attributes [65].

Constructing an efficient aggregation tree to reduce the total bits of transmitted data has been addressed in [29, 40]. In [29], a randomized tree construction algorithm is proposed and shown to perform near-optimal under widely varying scales of correlation for grid network topologies. They term this aggregation scheme opportunistic as data from a node to the sink is always sent over a shortest path. Their key idea is to allow each node to independently decide on its upstream parent toward the sink in a random way, and their data gets aggregated along the way. It was demonstrated that this scheme obviates the need for specialized routing structures.

In this randomized tree construction algorithm, a tree is constructed from random walks originating from each sensor. At each time step, the current node chooses one of the two upstream nodes as its parent. A node waits until both its downstream neighbors have sent their information out. Then, it aggregates the information it sensed locally, and sends it to one of its upstream neighbors. By finding the average expected collision time of two flows, they demonstrated that this randomized algorithm gives a constant factor approximation to the optimum aggregation cost for a given data correlation. The main advantage of this scheme is it is a simple algorithm that guarantees a near optimal aggregation tree for any correlation structure. However, it works only for grid deployments with a restricted channel model. They also assume data compression models specific to aggregation, wherein any n data values can be compressed into a single data value.

Traditional cost metrics do not apply to continuous queries over infinite streams, where processing costs per-unit-of-time is more appropriate [41]. Basically all proposed query languages for WSN support some new rewriting rules (e.g., [64]). Future directions are to support optimization for multiple queries [88], and to more tightly integrate intelligent routing protocols with query optimization/processing [78].

2.2.2 Multi-path aggregation

The main disadvantage of tree based aggregation has been pointed out by later works on multi-path aggregation [6, 69]: trees are susceptible to link loss and node loss in a sensor network. The closer a node is to the sink, the more important becomes a node for tree-based aggregation: if a link or node fails that is close to the sink, the aggregated information of an entire sub-tree could be lost. Multi-path aggregation, however, exploits the benefits of the wireless broadcast advantage that all nodes in communication range can hear a message and propagates the aggregates toward the sink using multiple routes. Multi-path routing becomes as a conse-

quence more robust for node failures or communication loss. In return, a multi-path aggregation algorithm has to deal with redundancy in data aggregation. In order to avoid multiple counting for the same sensor reading while aggregating, a multi-path algorithm can employ duplicate-insensitive sketches [19]. Sketches facilitate a compact representation using statistical information for data sets.

It is often claimed that multi-path aggregation algorithms are as energy efficient as tree-based aggregation algorithms [69], because each node only has to transmit a message once, in the same way as in any tree-based aggregation algorithm. However, a crucial factor missing in this equation: the receive time for each communicating sensor is increased. Recent studies on the energy consumption of wireless sensor nodes report that the energy requirement for the receive mode is only slightly lower compared to the energy cost for the transmission mode [13], i.e., a sensor node dissipates almost the same amount of energy in receiving data as in transmitting data. In multi-path aggregation each node expects to receive data from all nodes in communication range that have a higher level than the listening node. Therefore, the duration for receiving data can be considerably longer compared to a tree-based aggregation algorithm, where each node only has to listen to its direct children, a set that can be a significantly smaller.

Finding for each aggregation operator a duplicate-insensitive algorithm that guarantees a desired accuracy is a key challenge for multi-path routing schemes. The obvious approach to address this challenge would be to include control information with each aggregated message. The control information contains meta information such as the node IDs, which participated in the creation of the aggregate. This meta information can be used by each forwarding node to suppress duplication. Such an approach would have the same accuracy as an aggregation algorithm on a routing tree. The limited storage and processing capabilities of sensor nodes, however, render such a scheme impractical for larger sensor networks. Thus, all multi-path schemes integrate much cheaper probabilistic Order and Duplicate Insensitive (ODI) methods of the sketch theory [19, 32]. Therefore, current approaches in multi-path aggregation focus more on the development of approximate ODI algorithms for aggregation operators than on actual aggregation techniques for sensor networks.

2.2.3 Clustered in-network aggregation

Clustered in-network aggregation approaches exploit the spatial correlation of sensor readings to preserve node energy [72, 91, 95]. Spatial correlation in sensed data refers to the fact that sensor readings in close proximity are typically similar. Spatial correlation is a frequent phenomenon, in particular for attributes such as temperature, pressure, or humidity [45]. If a selective query has to retrieve an aggregate such as the average temperature in a certain area, then nearby nodes typically have similar readings and are geographically clustered in terms of the similarity of their sensor readings.

The Clustered AGgregation (CAG) and Geographic Adaptive Fidelity (GAF)

approaches, use the fact for spatially correlated nodes form clusters so that only one node needs to respond an aggregation query [91, 95]. Similarly, in static clustering [72] the network is statically partitioned into grid cells. For each grid cell one node is appointed as a cluster head node that acts as a gateway node. In each cell data is routed via a local shortest path tree, aggregated at the local gateway and then communicated directly to the sink.

Adaptive query processing techniques determine an optimal position for those nodes that aggregate information from different clusters. Eddies [5] proposes a dynamic aggregation scheme.

Clustered approaches are based on the idea, similar to SRTs, that not all nodes participate in an aggregation query directly. Different clusters require a variable node participation. However, the current clustered approaches are based on the assumption that participation is only determined by spatial correlation. In general, it is too restrictive to assume that participating nodes for every selective query will always be geographically clustered. Current clustered in-network aggregation approaches do not take into account that a selective query may not always translate into geographically clustered node participation.

As an example for a hierarchical aggregation scheme, we discuss LEACH (Low-Energy Adaptive Clustering Hierarchy) [46]. LEACH is one of the pioneers of cluster-based protocols. It forms random disjoint clusters among sensor nodes that are closely located, based on the received signal strength. Clusterheads (CHs) are dynamically elected using a stochastic approach to allow equal share of the energy-intensive CH role. Each CH acts as a data aggregator as well as a router for its cluster members. LEACH works in two stages: cluster setup and steady state. During the cluster setup phase, each node independently decides against a threshold whether it should be become the next CH. During the steady state, each regular node transmits data directly to its CH, which performs lossy aggregation without the notion of correlation. LEACH is a distributed scheme that does not require any global knowledge. It is able to ensure energy load balancing within the network to prolong the network lifetime. However, it performs aggregation without regard to the correlation structure of its nodes.

2.3 Transmission controlled gathering schemes

2.3.1 The clustered aggregation technique

CAG can also be categorized as a transmission controlled-based gathering technique. CAG operates in two phases: query and response. During the query phase, CAG forms clusters using a user-defined threshold. A regular node joins a cluster if its current reading is within a range computed from the threshold. Otherwise, it becomes a CH. During the response phase, only CHs transmit aggregated data every epoch forwarded along a routing tree. The rest of the nodes do not participate in further data generation, but may involve in forwarding if it is on the routing path. When the next query phase starts, the CHs might change except for the sink that

always becomes the first CH every time. The final aggregated value at the sink is shown to be within constant error from the aggregation over all sensor readings.

The main advantage of this scheme is that most redundant nodes could sleep until the next query phase. Moreover, the cluster setup process is simple and requires very limited overhead only. However, CAG has a number of shortcomings. Firstly, it assumes only a simple correlation structure through simple edges. Secondly, a single aggregated value is obtained from the network to represent the entire region profile. Finally, for some threshold, all nodes may be forced to be CHs resulting in a flat data gathering scheme.

2.3.2 Distributed correlation-dominating set scheme

In [45], the authors introduced techniques to exploit spatial correlations by selecting a small subset of sensors called correlation dominating set, whose data values are sufficient to interpolate value at all points with sufficient accuracy. A hypergraph is used to represent the correlation structure. As computing the minimum dominating set problem is NP-hard, they proposed some approximation algorithms. To select a connected correlation-dominating set of small size, they proposed a set of energy-efficient distributed algorithms and centralized algorithms.

The basic distributed algorithm works as follows. Firstly, each node assigns itself a priority. Then, each node collects k -hop neighborhood information, which may be gathered during the data gathering process using piggybacking and broadcast data transmissions. If a node could be inferred from a set of other nodes within some error threshold and this node is not required to preserve a connected graph, this node is redundant and could go to sleep. The set of nodes left active forms the connected correlation-dominating set.

The main advantage of this algorithm is that all redundant nodes are made to sleep, thus enabling significant energy savings. Secondly, a near optimal spatial correlation structure is formed in a distributed fashion to capture the correlations quite accurately. However, in collecting neighborhood information, a piggybacked message might tend to infinity in a very dense network. Furthermore, as each data message is flooded to all nodes over k -hops, these nodes have to be active for a sufficiently long time, which results in high energy usage.

2.4 Compression-induced gathering schemes

In all these schemes, a suitable encoding scheme is used to control the number of bits transmitted to sink. However, the most significant disadvantage in such an approach is that the need for all nodes to participated in all collection rounds.

2.4.1 Chou et al.'s distributed compression scheme

In [17], the authors proposed the distributed compression algorithm with the corresponding adaptive predictive algorithm for the sinks. The encoding is achieved in

a completely blind manner without any side information of the correlated sensors. It obviates the need for the sensors to exchange their data among themselves to obtain their correlation structure. Instead, the correlation structure among the nodes is tracked by the sink using a predictive algorithm. Their encoder is also able to support multiple compression rates.

In order for the sink to construct the correlation structure, the sink goes through an initialization phase where it gathers decoded data from every sensor for a certain number of rounds. The correlation structure among the nodes is then captured through a linear predictive model, and the readings are modeled as i.i.d Gaussian random variables. The number of rounds is chosen to be large enough to allow for the algorithm to converge. They derive a loose error bound on the estimation error probability based on the Chebyshev's inequality.

Once an estimate of the prediction coefficients is made, the sink determines the number of bits required for each sensor node to encode their data. The sink then requests each sensor node to compress and send its data. In order to respond to changes to the monitored environment, the sink keeps updating the correlation coefficients to reflect the current conditions. It also decodes the compressed sensor data by finding the nearest (in hamming distance) codeword in a specified codebook.

The main advantage of this scheme is that it is flexible to capture spatio-temporal correlation structure without requiring any state to be maintained at the individual nodes. The encoding is also done without any side information of other nodes. Furthermore, the decoding error is bounded. However, it is not a truly distributed scheme. The correlation structure is computed with the global knowledge of the network at a centralized data gathering node, i.e., the sink. Also, at each round after the initialization, the sink needs to inform each sensor individually their rate of compression. Therefore, a sink-to-any routing structure is required to support such communications in a multihop environment against the typical routing graph rooted at the sink.

2.4.2 Approximated Slepian-Wolf coding

In [21], the authors investigated the interaction between the correlation structure and the routing tree to transport the data. They considered two approaches. The first considers both compression and routing independently, where each node has the global correlation structure to perform compression using Slepian-Wolf coding. In the latter, source coding is performed with some side information obtained through explicit communications as part of the transport toward the sink. Both approaches were tackled optimally in centralized schemes as well as through approximated distributed solutions for scalability. Here, we focus on the proposed distributed solutions.

Initially, they compute the shortest-path tree using Bellman-Ford. Then, from the neighborhood information, each node finds the set of nodes closer to the sink. A node codes its data with respect to its entropy conditions on this set. This im-

plies that nodes closer to the sink have higher entropy than farther nodes. Thus, the nearer nodes perform more compression by sending lesser bits over longer distances.

For the explicit communication approach, the reduction in data size at a node depends on its position within the routing tree. However, when data in the network is strongly correlated, solving the problem of forming a spanning tree with minimal sum of parts from the leaves to the sink is shown to be similar to multiple traveling salesmen problem. This is NP-hard [22]. A number of heuristics were proposed based on well-known local search techniques such as simulated annealing and greedy search.

The main advantage of using the approximated Slepain-Wolf algorithms is that it is able to capture the correlation structure more accurately in an opportunistic manner. However, it is unclear how its prediction error bound would be, and its performance in a more realistic environment.

2.4.3 MEGA and LEGA

In [81], the authors focused on single-input coding strategies as they feature several advantages against multi-input coding strategies. Mainly, the former coding approach does not have any timing requirements unlike the latter approach. Since the problem of finding a minimum-energy data gathering tree is NP-complete for optimal self-coding, they proposed an approximation algorithm. This algorithm, termed LEGA, is based on shallow light tree that combines the properties of the spanning tree and the shortest path tree. For foreign coding, they introduced the algorithm termed the minimum-energy gathering algorithm (MEGA) that gives an optimal solution for data gathering by reducing it to the problem of finding a MST in a digraph.

For optimized foreign coding, MEGA uses the superposition of a directed MST and shortest path tree (SPT) rooted at the sink. Firstly, MEGA computes a SPT using Dijkstra's algorithm. As the weight of each link is proportional to the transmission of one bit data, the resulting tree is an energy-minimal tree. Secondly, it computes a parent node to encode the packet for a node using direct MST. The weights of the MST edges depend on the distance and the correlation coefficient of the involved nodes. Once a packet is encoded by this parent, it is sent toward the sink on the SPT. A distributed version of this algorithm is also given based on a distributed algorithm of directed MST. The main advantage of this approach is that it is usable on asynchronous networks as well as no timing assumptions are required. However, the correlation structure is very simplified and the state required to build and maintain dual adaptive trees is substantial.

Since the problem of constructing an efficient data gathering tree in a self-coding model is shown to be NP-complete, the same authors introduced an approximation algorithm within a constant factor. This algorithm uses the shallow light tree. The encoding starts from the k -hop neighbor of the sink and proceeds further toward the leaves. Once a node encodes its data, it forwards the encoded

data toward the sink, and also broadcasts its raw data to farther nodes. Upon receiving this side information, these nodes self-encode and then repeats the above step. The main advantage of this scheme is its simplicity. However, it involves considerable overhead to allow explicit communication for the coding purpose. In a dense network, even though some side information has become available, a node may receive much more redundant data.

In all the above schemes, it is evident that all nodes participate in each round of data collection albeit with reduced bits of information. Since the dominant energy usage is more attributed to the power amplification component to allow a data bit to reach the receiver, a more energy efficient approach would be to decrease the number of transmissions altogether.

2.5 Data-centric storage and indexing

A key difference between a classical network and a sensor network is that the distinction between data and its address is not important. Sensor nodes can fail and nearby nodes can take over the functions of failing sensors. A data-centric view does not rely on a node's ID but takes the sensed information, the type of the sensed information, or the location of the sensor readings into account. An example of a data centric view on storage is given by Directed Diffusion [47]: the sensor reading of a node is locally stored at that node. Nodes can publish sensor readings (events) and nodes interested in those events can subscribe to it. However, the cost for finding events can be $O(n)$ for n sensor nodes. Thus, this approach does not scale to larger sensor networks.

Data-centric storage (DCS) stores and names data with respect to its attribute(s). Similar to directed diffusion, the network address, where the data is stored, is not relevant. In DCS, the data is stored by name: all sensor-based data with the same name (of the same type) are stored at one node in the sensor network. This node does not need to be the node that originally sensed this data. This schema avoids flooding the network to find the data, because the query can be sent directly to the node storing the desired information. DCS distributes the storage load among the sensor nodes.

To enable efficient data access in a sensor network, an index is required. Most indexing approaches assume a flat architecture, where each node has the same role. Geographic Hash Tables (GHTs) [79] implement DCS. A GHT constructs an in-network index for efficient data access that distributes the data among the sensor nodes. The GHT approach requires that each node knows its own location, for example using GPS or a location service [58]. A hash function maps attributes to node locations. GHT associates with each datum a key and uses a hash table to map keys to nodes storing the requested data. The main disadvantage of GHTs are their inability for range queries, which limit their usability for indexing spatial data.

To be able to perform multi-dimensional range queries, several approaches have been suggested: DIFS, *fractional cascading*, and DIMS. DIFS [44], a Distributed Index for Features in Sensor networks, is based on a multi-rooted quadtree

for partitioning the deployment layout. DIFS modifies the concept of GHTs in order to support range queries. Similar systems such as fractional cascading [36] and DIMS [59] (Distributed Index for Multidimensional data), which is a locality-preserving hashing scheme, have been developed to address range queries. Since data cannot be stored permanently on sensor nodes due to their limited storage capabilities, older data must be discarded over time. This process of data aging is supported by DIMENSIONS [35], an approach for multi-resolution summaries over spatio-temporal data.

TSAR [27], Tiered Storage ARchitecture, endorses a two-tiered architecture instead of a flat architecture for storing and indexing sensor data. The approach distinguishes two types of nodes: proxies that perform indexing and manage meta data, and normal sensor nodes. Each proxy is responsible for a group of normal sensor nodes. The motivation for the two tiered architecture is two-fold: first, future sensor networks could be multi-tiered and second, indexing could be too complex for normal sensor nodes. The maintenance of hash tables and the number of updates could turn out to be energy demanding for normal sensor nodes. Therefore, the TSAR approach suggests to store the sensor readings at the normal nodes but access this information via distributed indices maintained at the proxies. For the distributed index structure, TSAR proposes the use of *sparse interval skip graphs*. An interval skip graph is a distributed data structure that enables the efficient search for all intervals that contain a particular point or range of values. The data structure requires only $O(\log n)$ time for accessing the first interval that matches a particular value or range, and constant time for each successive interval. Each sensor transmits interval summaries of data as an interval with the minimum and maximum sensed value to a proxy. Sparse interval skip graph can efficiently support spatio-temporal and range queries.

[25] presents a peer-to-peer indexing structure, a kind of distributed R-tree called peer-tree, that supports energy- and time-efficient executions of spatial queries in WSN. [18] discusses how to take advantage of data redundancy in sensor networks and proposes techniques to process spatio-temporal region queries in WSN. Finally, in [26], the query answers are estimated using a statistical model for the sensors' readings, where the model captures the redundancy and correlation in sensor measurements.

3 Programming Environments

The intricacies associated with WSN application development make support in the form of a WSN programming environment highly desirable. Ideally, a WSN programming environment will provide support for all of the above aspects. This claim neglects the fact that in the context of WSNs these aspects must be considered areas of active research (which precludes the existence of turn-key solutions). The combination of embedded, distributed and real-time aspects in WSNs induces additional difficulties. These difficulties arise from the non-orthogonality of measures

taken to support different aspects of the system (e.g., switching sensor nodes to a dormant state reduces power consumption while it complicates distributed algorithms that attempt to maintain shared global time).

Programming environments for WSNs implement programming paradigms. A programming paradigm implies a conceptual model for programmers, and it is related to a certain abstraction level. For example, does the programmer think in terms of system-level programming techniques (such as events, message passing and synchronization), or does he focus on a more problem-oriented view of the whole WSN application? Accordingly, current WSN programming environments can be grouped as follows.

- *Node-level programming environments* support the development of node-centric WSN applications. In a node-level programming environment programmers think in terms of how single (sensor) nodes operate in the context of the WSN application. A node-level programming environment can be just a programming language with a component library, or it can be a node-centric operating system with hardware and networking abstractions. Node-level programming may extend to the execution platform, and provide the programmer with a mixed-mode execution environment. Mixed-mode facilitates execution of code either as binary or as byte code. Binary code achieves high execution speed, whereas byte code can be compressed efficiently (thus reducing memory footprint).
- *High-level programming environments* abstract from single sensor nodes by providing programming paradigms that consider the WSN application as a whole.

In particular, *state-centric programming environments* obey the fact that intrinsic to WSN applications is the notion of states of physical phenomena, together with models of their evolution over space and time. The state of an application may be composed of a number of state variables distributed among several sensor nodes. Inputs observed through the WSN application advance the state through sequential state updates. State-centric programming environments abstract from computational aspects related to concurrency, responsiveness, networking and resource management, which results in a higher abstraction level than node-level programming environments.

WSNs can be envisioned as distributed *databases* that collect and index physical measurements and serve queries from users and network-external and internal applications. WSN databases provide a logical data-centric view of the network and abstract from physical storage and communication issues.

The majority of WSN programming language environments and constituents are related to the node-level. We discuss these approaches in Sections 3.1–3.4. High-level approaches are treated in Section 3.5. We draw our conclusions in Section 3.6.

3.1 Programming languages

NesC [38, 37] is the de-facto standard for application and system programming in WSNs. NesC is a domain specific language designed for WSN applications. The programming paradigm of nesC is based on a static component model. The language incorporates event-driven execution and a flexible concurrency model. Due to several restrictions in the language, it is possible to conduct extensive program analyses and optimizations such as data-race detection and aggressive function inlining. The basic compilation steps are as follows: First, a nesC program is translated by the nesC compiler. Program analyses and optimizations are performed. The output of the nesC compiler is a C program. Second, the C output of the nesC compiler is translated to a binary code by the GCC compiler [39].

NesC has been employed to implement a significant number of WSN applications. TinyOS, which is the state-of-the-art operating system for WSNs (see Section 3.2), has been written in NesC as well.

A nesC extension named galsC was introduced in [15]. The programming language galsC is a dataflow language [2] that implements the TinyGALS [14] programming model. A dataflow program in this model consists of a set of processing units called actors. Actors provide ports to produce and receive data; they capture the concurrency inherent in a system. Asynchronous event-driven execution is a special case of dataflow models where each actor is triggered by every incoming event. The globally asynchronous and locally synchronous (GALS) mechanism is a method to build event-triggered concurrent execution from thread-unsafe (nesC) components.

Although nesC and galsC are very popular, they are not the only programming languages used for embedded systems programming. Due to the resource limitations in small embedded systems, assembly language is still commonly used. However, due to the complexity of modern applications and the increase in computational power and memory size, there has been a shift from assembly-language to high-level languages. New compilation techniques are necessary to cope with quite often contradicting objectives such as code size, power consumption, and runtime [56].

Prominent languages for embedded systems are Ada [48], C [82], and Java [43]. Ada and C are full-fledged imperative programming languages. A survey conducted by the Gartner Group reports that more than 78% of all embedded systems firmware and application developers use the C programming language [28]. C covers even niches such as signal processing applications [54].

In contrast to C, Ada has been designed for safety critical systems. It is a strongly typed language and the semantics of the language has been clearly defined. Ada was later extended with object-oriented programming constructs and mechanisms for distributed systems programming. The latest revision of the Ada programming language is Ada 2005. It includes a component library, Java-like interfaces and major improvements for real-time and high-integrity systems.

The Java language [43] is very successful in many application domains such as

web applications. It is also used in embedded systems [86].

A scripting language for WSN applications has been introduced in [10]. Scripts execute node-level state machines that are mainly driven by external events. The scripting language provides high-level commands for data acquisition, communication, and migration of scripts to other nodes.

3.2 Operating systems

For programming environments operating systems play a significant role. Operating systems manage memory, processes, and network communication. For wireless sensor nodes it is important that the operating can cope with these tasks but has a small footprint considering the resource constraints.

TinyOS [87] is an operating system (OS) specifically designed for WSNs. It features a component-based architecture which enables rapid innovation and implementation while minimizing memory footprint as required by the severe memory constraints inherent in sensor networks. As a result, TinyOS does not support advanced features such as preemptive task scheduling and sophisticated synchronization primitives. TinyOS is entirely written in nesC [38].

In the past several commercial operating systems for embedded systems have been introduced, e.g., QNX [77], Windows CE [66], VxWorks, LynxOS, VRTX, pSOS and Nucleus RTX. QNX is one of the first true commercial microkernel operating systems. Under QNX every driver, application, protocol stack, and file system runs outside the kernel, in the safety of memory-protected user space. As a result, virtually any component can fail and be automatically restarted without affecting other components or the kernel. No other commercial RTOS provides such protection. Windows CE [66] and its successors (e.g., Pocket Windows, Windows XP Embedded, etc.) are mainly used for Smartphones and PDAs, which can also be regarded as embedded systems.

Linux as the most successful open source operating system branched off to several embedded systems variants such as μ Clinux [71] and Real-Time Linux [80]. μ Clinux is intended for microcontrollers without a memory management unit. Real-Time Linux is a hard real-time system beneath the Linux kernel. It treats the Linux kernel as a low-priority real-time task.

There is a clear trade-off between the functionality of an embedded systems OS and its resource consumption. TinyOS as the most simplistic OS has a footprint of only 400 bytes (core OS code and data!). Footprints of more advanced real-time kernels can get as big as several hundred kilobytes, or even megabytes (in the case of Windows XP Embedded).

3.3 Virtual machines and mixed-mode execution

Instead of translating high-level language code to binary code, it can be translated to byte code that is then interpreted by a virtual machine (VM). A VM abstracts from the properties of the underlying hardware which makes program execution

hardware-independent. In comparison to other implementation techniques of programming languages, VMs have the advantages of portability, ease of implementation and fast edit-compile-run cycles. Examples of VM-based architectures are Java's JVM [61], Prolog's WAM [3] and Smalltalk's VM [42]. VMs are lightweight which makes the induced code size overhead tolerable for WSNs [57]. This overhead is further alleviated by the fact that byte code is well-suited for compression, which reduces the overall memory footprint of the application (see Section 3.4).

For the development of VMs a series of tools [30, 49, 33] has been proposed. Vmgen [30] assists the programmer with the implementation, debugging and performance measurement of VMs. Vmgen generates the code for the interpreter engine. It uses several techniques to improve the VM performance, e.g., threaded code [7], scheduling the dispatch of the next instruction, keeping the top-of-stack in a register, combining instructions to super-instructions, eliminating stores of unchanged stack items, and improving the branch predication accuracy.

Mixed mode execution for C is presented in [75], mixed mode execution for Java was introduced in [67].

3.4 Code space optimizations

Small-scale embedded systems such as sensors are severely restricted by the amount of available memory (e.g., the WeC Berkeley Mote [96] contains only 8 kilobytes of program memory and 512 bytes of RAM). Memory is a scarce commodity for the following reasons.

- *Power consumption:* the transistor circuitry constituting program and data memory consumes power, which in turn reduces the battery lifetime of the system. To achieve low power consumption, the system's silicon area (and hence the available memory) must be kept at a minimum.
- *Limited physical space:* with WSN applications it is usually not desirable to increase the size of sensor nodes in order to accommodate additional memory circuitry.
- *Production costs:* additional memory circuitry increases die and wafer sizes which in turn increases the production costs of sensor nodes. However, to make large-scale WSN applications affordable and to keep the WSN-related cost relative to the total product cost low, WSN nodes are meant to become a mass product in the near future.

Memory constraints severely limit the sizes of prospective applications. For this reason significant effort has been spent to minimize the memory footprint of embedded systems applications.

In the realm of Java, compression rates of up to 85% of the original program size have been achieved [76, 90, 11]. Instead of sophisticated compression schemes alternate representations of the VM code (e.g., trees) have been used in [51].

Techniques for binary code have been introduced in [23, 20, 52, 60, 34, 50, 55]. The main technique is to split the code into various portions and to compress them with different compression schemes. Even the instructions are split in op-code and parameters. This gives further opportunity to remove redundancies.

In [53] compression (i.e., Huffman codes) is incorporated into the VM instruction fetch.

Compilation techniques for embedded systems have only started to emerge in the past few years (cf. [68, 97, 84, 12]). They can reduce power consumption and memory footprint of applications through optimizations, either during compilation or as a post-compilation step. Program optimizations heavily depend on program properties derived through advanced static program analysis techniques [31, 8, 70].

3.5 High-level approaches

Issues addressing *where* and *when*, rather than *how*, to perform sensing, computation, and communication, are vital for the performance of a WSN application. These nonfunctional aspects of the application are related to concurrency, timeliness, responsiveness, distributed systems aspects, networking, and resource management. Node-level programming paradigms support these aspects only to the extent foreseen by the application programmer.

State-centric approaches attempt to provide programming paradigms that offer meaningful abstractions for nonfunctional aspects of the application. The aim is to enable the programmer to think in terms of physical phenomena and signal processing instead of communication protocols, concurrency, and network management. This is even more important for future WSN applications, where developers will likely be domain experts rather than networking or OS experts.

Central to state-centric programming paradigms is the fact that WSN applications as a whole model states of physical phenomena, together with the phenomena's evolution over space and time. This requires to address the following concerns.

- How is the global system state decomposed into state variables?
- Where are state variables stored?
- How are state variables replicated?
- How are the events (i.e., sensor inputs) acquired?
- Where and how are the system state, state transitions and output computed?

PIECES (Programming and Interaction Environment for Collaborative Embedded Systems) [62, 96] is a programming environment that implements the state-centric programming paradigm. With PIECES the programmer decomposes the global system state into a hierarchy of independently update-able parts called pieces. Every piece has associated one computational entity (called principal) that

manages the state of the piece. Although other principals may cache copies of pieces, there exists exactly one master copy for each piece in the system. This ensures system-wide consistency of state update, but it introduces single points of failure (PIECES does neither provide state machine replication [4] nor failure detectors).

Principals can form collaboration groups. Members of a collaboration group contribute to a state update. Collaboration groups allow principals to address other principals through their roles rather than their names or logical addresses. The abstraction of collaboration groups shields designers from complicated communication protocol management and event-handling issues.

Principals are free to roam the WSN; at any given time a principal is hosted by exactly one WSN node. Mobility lets principals follow physical phenomena spatially, which minimizes latency and communication overhead. At the moment PIECES exists in the form of a modeling and simulation framework implemented in Java and Matlab.

WSNs can be envisioned as distributed databases that collect and index physical measurements and serve queries from users and network-external and internal applications. WSN databases provide a logical data-centric view of the network and abstract from physical storage and communication issues. This abstraction induces a loss of efficiency, which is somewhat compensated by highly increased ease of use. Examples of WSN database systems include Cougar [9] and TinyDB [64].

Generic role assignment is introduced in [83] to assign user-defined roles to sensor nodes. Roles are assigned based on the capabilities of the sensor nodes.

3.6 Summary

Programming environments for WSNs need to cover aspects from areas as diverse as embedded systems, distributed systems, and real-time systems. Resource constraintness can be considered the foremost characteristic that affects WSN application development. Measures taken to reduce resource consumption often conflict with the distributed nature of WSNs and have adverse effects on timeliness.

Contemporary programming environments offer programming paradigms on different levels of abstraction. Node-level programming environments force the programmer to think in terms of how single sensor nodes operate in the context of the WSN application. High-level programming environments abstract from single sensor nodes by providing programming paradigms that consider the WSN application as a whole. Both abstraction levels have advantages and disadvantages: node-level programming exposes the underlying system constraints and gives the programmer full access to every aspect of the implementation. However, this abstraction level requires the programmer to effectively deal with every such aspect. Non-functional requirements such as resource-awareness, timeliness, fault-tolerance, communication and concurrency are all left to the programmer.

High-level programming approaches are meant to lift the programmer's abstraction level above the before-mentioned technical issues. The intended benefit is

to keep programmers from re-inventing (complicated) wheels and to enable them to focus on the functional requirements of WSN applications.

Current high-level approaches cover only small fractions of the issues they are meant to handle. In particular, real-time issues and fault tolerance of WSN applications are left out. At the moment high-level approaches implicitly assume a simplistic fault model, where a *fail silent* property is assumed for sensor nodes. A given high-level approach cannot be used if it misses out on a required technical issue that cannot be manually implemented due to the enforced abstraction level. As an example, consider WSN database systems: query interfaces provided by such systems do not allow the programmers to address real-time issues. If timeliness is not already guaranteed by the database system, there is no way to incorporate it manually. In other cases the provided abstraction is less preclusive: although PIECES does not provide state machine replication, the provided abstraction does not preclude a manual state machine replication mechanism.

4 Intrusion Detection

The development of WSNs offers the promise of a flexible, low cost solution for monitoring critical infrastructure. In any application involving critical infrastructure, there is the potential risk of malicious attacks on this infrastructure, either for financial gain or as a terrorist act. These networks have a critical role to play in detecting these attacks, and thus can become a target for attack in its own right. However, the problem of detecting attacks on WSNs has not been well addressed in the literature.

A key attraction of WSNs is their ease of installation and operation. However, security is one of the key challenges to creating a robust and reliable network [73]. Currently, most research on security in WSNs has focused on prevention techniques, such as secure routing protocols, and cryptography and authentication techniques [89]. These security mechanisms are usually the first line of defense. However, experience with the Internet has shown that flaws in these protocols are continuously being found and exploited by attackers [94]. WSN protocols are faced with additional challenges due to complexities such as the error-prone wireless channel, unpredictable node movement and unreliable node operation. These challenges create considerable potential to exploit weaknesses in the network. Consequently, we cannot rely on intrusion prevention techniques alone. In practice, Intrusion Detection Systems (IDSs) are needed to detect both known security exploits and even novel attacks that have yet to be experienced.

Intrusion detection is the problem of identifying misuse of computer systems and networks [85]. Most IDSs apply signature-based techniques. In general, signature-based techniques test for features of known network attacks. This raises the question of how to learn these features for known attacks, and how to detect new attacks. It is difficult to use supervised learning in this context, since labeled training data is expensive to produce. More importantly, it is difficult to detect new

types of attacks whose signatures may differ from those in its signature set. This has motivated research into unsupervised learning techniques, which do not require labeled data and are able to detect previously ‘unseen’ attacks.

Instead of learning the signature of attack traffic, unsupervised anomaly detection techniques focus on learning the signature of normal traffic. Unsupervised learning techniques do not require the data to be labeled, nor do they require the data to be purely of one type, i.e., normal or attack traffic. This is a significant benefit over the supervised learning approach.

5 Conclusions and Future Directions

This report discusses a number of high-level issues to be resolved in realizing a complete in-network processing solution for WSNs, mainly related to data gathering and aggregation, storage and indexing, query processing and data models, programming environments and intrusion detection.

The problem of efficient data gathering of correlated data in WSNs is a challenging one but promises a large saving in data transmission costs. This is particularly advantageous to sensor nodes where energy constraints are rather severe. In a periodic monitoring scenario, any form of compression or aggregation may result in significant energy savings. Based on the proposals in the literature, we classified them into categories such as compression-induced schemes, transmission-controlled schemes, aggregation-tree schemes and topology control schemes. Compression-induced schemes aim to reduce the number of bits transmitted after obtaining some information about correlations. However, they require all nodes to participate in every data collection round. As for the transmission-controlled schemes, they aim to reduce the number nodes involved in data gathering. After learning about the correlation structure, all redundant nodes are forced to sleep and the sink just predicts their values. However, the learning phase may prove to be quite overhead intensive. The aggregation-tree-based schemes aim to form an efficient aggregation tree that reduces the number of bits drastically. Such a structure should ideally be mapped onto the routing structure to realize opportunistic in-network aggregation. As for the topology control, most schemes are mainly correlation-unaware and the redundancy is entirely decided from physical connectivity point of view. In addition, data storage can be viewed as an alternative method to data collection by creating rendezvous points for queries and data.

Most of these algorithms are suited for periodic monitoring applications. However, it is unclear how they would perform in an event-driven application with real-time constraints. As most of these schemes require a substantial learning phase, delivering such event information may be challenging to achieve within its delay constraints. Also, the introduction of actuation mechanisms into WSNs to allow swift event responses may require significant redesign of such data gathering algorithms.

Designing an effective data processing scheme also requires modifications on

nearly every part of a traditional query processor. Many interesting problems are introduced, such as adding time, order, and windowing to data models and query languages. The combination of push-based and pull-based operators in a network poses additional complexity. Other important issues that need to be addressed for data processing are handling and integration of user-defined functions and aggregations, adaptive query re-optimization and data routing, and distributed in-network query processing.

Programming environments for WSNs need to cover aspects from areas as diverse as embedded systems, distributed systems, and real-time systems. Resource constraints can be considered the foremost characteristic that affect WSN application development. Measures taken to reduce resource consumption often conflict with the distributed nature of WSNs and have adverse effects on timeliness.

Contemporary programming environments offer programming paradigms on different levels of abstraction. Node-level programming environments force the programmer to think in terms of how single sensor nodes operate in the context of the WSN application, e.g., nesC. High-level programming environments abstract from single sensor nodes by providing programming paradigms that consider the WSN application as a whole. However, current high-level approaches cover only some aspects such as resource awareness, distributed programming, real-time issues and fault tolerance.

References

- [1] D. J. Abadi, S. Madden, and W. Lindner. REED: Robust, efficient filtering and event detection in sensor networks. In *Proceedings of the 31th International Conference on Very Large Data Bases (VLDB)*, pages 769–780, 2005.
- [2] W. B. Ackerman. Data flow languages. *IEEE Computer*, 15(2):15–25, 1982.
- [3] H. Ait-Kaci. *Warren’s Abstract Machine: A Tutorial Reconstruction*. MIT press, Cambridge, 1991.
- [4] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. John Wiley and Sons, Inc., second edition, 2004.
- [5] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 261–272. ACM Press, 2000.
- [6] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 515–526, 2004.
- [7] J. R. Bell. Threaded Code. *Communications of the ACM*, 16(6), June 1973.

- [8] J. Blieberger and B. Burgstaller. Eliminating Redundant Range Checks in GNAT Using Symbolic Evaluation. In *Proceedings of the Ada-Europe International Conference on Reliable Software Technologies*, pages 153–167, June 2003.
- [9] P. Bonnet, J. Gehrke, and P. Seshadri. Towards Sensor Database Systems. *Lecture Notes in Computer Science*, 1987:3–14, 2001.
- [10] A. Boulis and M. B. Srivastava. A Framework for Efficient and Programmable Sensor Networks. In *Proceedings of the 5th IEEE Conference on Open Architectures and Network Programming*, 2002.
- [11] Q. Bradley, R. Horspool, and J. Vitek. JAZZ: An efficient compressed format for java archive files. In *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, page 7, 1998.
- [12] B. D. Bus, B. D. Sutter, L. V. Put, D. Chanet, and K. D. Bosschere. Link-time optimization of ARM binaries. In *Proceedings of the 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '04)*, pages 211–220. ACM Press, 2004.
- [13] M. Cardei and J. Wu. Energy-efficient coverage problems in wireless ad hoc sensor networks. *Computer Communications*, 29(4):413–420, February 2006.
- [14] E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: A Programming Model for Event-Driven Embedded Systems. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC '03)*, pages 698–704. ACM Press, 2003.
- [15] E. Cheong and J. Liu. galsC: A Language for Event-Driven Embedded Systems. In *Design, Automation and Test in Europe (DATE'05)*, volume 2, pages 1050 – 1055, 2005.
- [16] C. Chong and S. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, August 2003.
- [17] J. Chou, D. Petrovic, and K. Ramachandran. A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks. In *Proceedings of the 22nd Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 1054–1062, 2003.
- [18] A. Coman, M. A. Nascimento, and J. Sander. Exploiting redundancy in sensor networks for energy efficient processing of spatiotemporal region queries. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 187–194. ACM Press, 2005.

- [19] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, pages 449–460. IEEE Computer Society, 2004.
- [20] K. D. Cooper and N. McIntosh. Enhanced code compression for embedded RISC processors. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 139–149, 1999.
- [21] R. Cristescu, B. Beferull-Lozano, and M. Vetterli. On network correlated data gathering. In *Proceedings of the 23rd Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2571–2582, 2004.
- [22] R. Cristescu and M. Vetterli. Power efficient gathering of correlated data: optimization, np-completeness and heuristics. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):31–32, 2003.
- [23] S. Debray and W. Evans. Profile-guided code compression. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI '02)*, pages 95–105. ACM Press, 2002.
- [24] A. J. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. The Cougar project: A work-in-progress report. *ACM SIGMOD Record*, 32(4):53–59, December 2003.
- [25] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P 2003)*, pages 32–39. IEEE Computer Society, 2003.
- [26] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor network. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada*, pages 588–599. Morgan Kaufmann Publishers, 2004.
- [27] P. Desnoyers, D. Ganesan, and P. Shenoy. TSAR: a two tier sensor storage architecture using interval skip graphs. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, pages 39–50. ACM Press, 2005.
- [28] eMedia Asia Ltd. and Gartner, Inc. Embedded Systems Development Trends: Asia. <http://www.eetasia.com>, 2005.
- [29] M. Enachescu, A. Goel, R. Govindan, and R. Motwani. Scale free aggregation in sensor networks. In *First International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, 2004.
- [30] M. A. Ertl, D. Gregg, A. Krall, and B. Paysan. *vmgen* — a generator of efficient virtual machine interpreters. *Software—Practice and Experience*, 32(3):265–294, 2002.

- [31] T. Fahringer and B. Scholz. *Advanced Symbolic Analysis for Compilers*, volume 2628. LNCS, Springer-Verlag, 2003.
- [32] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.
- [33] B. Folliot, I. Piumarta, and F. Riccardi. A Dynamically Configurable, Multi-language Execution Platform. In *Proceedings of the 8th ACM SIGOPS European Workshop*, pages 175–181, September 1998.
- [34] C. W. Fraser, E. W. Myers, and A. L. Wendt. Analyzing and compressing assembly code. *ACM SIGPLAN Notices*, 19(6):117–121, 1984.
- [35] D. Ganesan, D. Estrin, and J. Heidemann. Dimensions: why do we need a new data handling architecture for sensor networks? *SIGCOMM Computer Communication Review*, 33(1):143–148, 2003.
- [36] J. Gao, L. J. Guibas, J. Hershberger, and L. Zhang. Fractionally cascaded information in a sensor network. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 311–319, New York, NY, USA, 2004. ACM Press.
- [37] D. Gay, D. Culler, and P. Levis. nesC Language Reference Manual, September 2002.
- [38] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI2003)*, pages 1–11. ACM, 2003.
- [39] GNU Software Foundations. Gnu Compiler Collection. <http://gnu.gcc.org>.
- [40] A. Goel and D. Estrin. Simultaneous optimization for concave costs: single sink aggregation or single source buy-at-bulk. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 499 – 505, Baltimore, Maryland, 2003.
- [41] L. Golab and M. T. Özsu. Issues in data stream management. *ACM SIGMOD Record*, 32(2):5–14, June 2003.
- [42] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.
- [43] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Java Series. Sun Microsystems, 1996.
- [44] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. Difs: A distributed index for features in sensor networks. In *First IEEE International*

- Workshop on Sensor Network Protocols and Applications*, pages 163–173. IEEE Computer Society, May 2003.
- [45] H. Gupta, V. Navda, S. Das, and V. Chowdhary. Efficient gathering of correlated data in sensor networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '05)*, pages 402–413. ACM Press, 2005.
 - [46] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660 – 670, 2002.
 - [47] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking (TON)*, 11(1):2–16, February 2003.
 - [48] ISO/IEC 8652:1995. Ada 95 Reference Manual.
 - [49] R. A. Kelsey and J. A. Rees. A tractable Scheme implementation. *Lisp and Symbolic Computation*, 7(4):315–335, 1994.
 - [50] D. Kirovski, J. Kin, and W. H. Mangione-Smith. Procedure based program compression. In *Proceedings of the Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture*, pages 204–213, 1997.
 - [51] T. Kistler and M. Franz. A tree-based alternative to Java byte-codes. *International Journal of Parallel Programming*, 27(1):21–33, Feb. 1999.
 - [52] M. Kozuch and A. Wolfe. Compression of embedded system programs. In *Proceedings of the 1994 IEEE International Conference on Computer Design: VLSI in Computer & Processors (ICCS '94)*, pages 270–277. IEEE Computer Society, 1994.
 - [53] M. Latendresse and M. Feeley. Generation of fast interpreters for huffman compressed bytecode. In *Proceedings of the 2003 Workshop on Interpreters, Virtual machines and emulators (IVME '03)*, pages 32–40, New York, NY, USA, 2003. ACM Press.
 - [54] K. W. Leary and W. Waddington. DSP/C: A standard high level language for DSP and numeric processing. In *Proceedings of the Int. Conference on Acoustics, Speech and Signal Processing*, pages 1065–1068. ACM, 1990.
 - [55] H. Lekatsas and W. Wolf. Code compression for embedded systems. In *Proceedings of the 35th Design Automation Conference (DAC 1998)*, pages 516–521, 1998.
 - [56] R. Leupers. *Code Optimization Techniques for Embedded Processors: Methods, Algorithms, and Tools*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

- [57] P. Levis and D. Culler. Maté: A tiny virtual machine for sensor networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, pages 85–95, Oct. 2002.
- [58] J. Li, J. Jannotti, D. D. Couto, D. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130. ACM Press, 2000.
- [59] X. Li, Y. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pages 63–75. ACM Press, 2003.
- [60] S. Liao, S. Devadas, and K. Keutzer. Code density optimization for embedded DSP processors using data compression techniques. In *Proceedings of the 16th Conference on Advanced Research in VLSI (ARVLSI '95)*, pages 272–285. IEEE Computer Society, 1995.
- [61] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Second edition, 1999.
- [62] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. State-Centric Programming for Sensor and Actuator Network Systems. *IEEE Pervasive Computing Magazine*, 2(4):50–62, October 2003.
- [63] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.
- [64] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA*, pages 491–502. ACM Press, June 2003.
- [65] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30(1):122–173, March 2005.
- [66] I. Microsoft. Windows CE. <http://www.microsoft.com/ce>.
- [67] G. Muller, B. Moura, F. Bellard, and C. Consel. Harissa: A flexible and efficient Java environment mixing bytecode and compiled code. In *Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems*, pages 1–20, Portland, OR, USA, June 1997. Usenix.

- [68] M. Naik and J. Palsberg. Compiling with Code-Size Constraints. *Transactions on Embedded Computing Systems*, 3(1):163–181, 2004.
- [69] S. Nath, P. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 250–262. ACM Press, 2004.
- [70] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
- [71] Open Software Foundations. *μlinux*. <http://www.uclinux.org/>.
- [72] S. Patten, B. Krishnamachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN'04)*, pages 28–35. ACM Press, 2004.
- [73] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, June 2004.
- [74] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.
- [75] T. A. Proebsting. Optimizing an ANSI C Interpreter with Superoperators. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '95)*, pages 322–332. ACM Press, 1995.
- [76] W. Pugh. Compressing java class files. In *Proceedings of the 1999 SIGPLAN Conference on Programming Language Design and Implementation*, pages 247–258, 1999.
- [77] QNX Software Systems. Rtos. <http://www.qnx.com>.
- [78] M. Rabbat and R. Nowak. Distributed optimization in sensor networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN 2004), Berkeley, California, USA, April 26-27, 2004*, pages 20–27. ACM Press, 2004.
- [79] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with GHT, a geographic hash table. *Mobile Networks and Applications (MONET)*, 8(4):427–442, August 2003.
- [80] Real Time Linux Foundation, Inc. Real time linux. <http://www.realtimelinuxfoundation.org/>.
- [81] P. v. Rickenbach and R. Wattenhofer. Gathering correlated data in sensor networks. In *Workshop on Foundations of mobile computing*, pages 60–66, Philadelphia, PA, USA, 2004.

- [82] D. M. Ritchie, B. W. Kernighan, and M. E. Lesk. The C programming language. Comp. Sci. Tech. Rep. No. 31, Bell Laboratories, Murray Hill, New Jersey, October 1975. 31 Superseded by B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1988.
- [83] K. Römer, C. Frank, P. J. Marrón, and C. Becker. Generic Role Assignment for Wireless Sensor Networks. In *Proceedings of the 11th ACM SIGOPS European Workshop*, pages 7–12, Leuven, Belgium, Sept. 2004.
- [84] U. P. Schultz, K. Burggaard, F. G. Christensen, and J. L. Knudsen. Compiling Java for Low-end Embedded Systems. *SIGPLAN Not.*, 38(7):42–50, 2003.
- [85] S. Snapp, J. G. Dias, T. Goan, L. Heberlein, C. Ho, K. Levitt, B. Mukherjee, S. Smaha, T. Grance, D. Teal, and D. Mansur. DIDS Distributed Intrusion Detection System — motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, Washington, DC, 1991.
- [86] Sun Microsystems, Inc. Java card technology, 1997. <http://java.sun.com/products/javacard>.
- [87] TinyOS 2.0 Working Group. TinyOS. <http://www.tinyos.net>.
- [88] N. Trigoni, Y. Yao, A. J. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *Proceedings of the 1st IEEE International Conference on Distributed Computing in Sensor Systems, Marina del Rey, CA*, volume 3560 of *LNCS*, pages 307–321. Springer Verlag, 2005.
- [89] J. Undercoffer, S. Avancha, A. Joshi, and J. Pinkston. Security for sensor networks. In *Proceedings of the 2002 CADIP Research Symposium*, October 2002.
- [90] L. R. von der Clausen, U. P. Schultz, C. Consel, and G. Muller. Java bytecode compression for low-end embedded systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22(3):471–489, 2000.
- [91] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom '01)*, pages 70–84. ACM Press, 2001.
- [92] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, September 2002.
- [93] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR2003), Asilomar, CA, USA, January 5-8*, pages 233–244, 2003.

- [94] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proceedings of ACM SIGMETRICS*, pages 138–147, June 2003.
- [95] S. Yoon and C. Shahabi. Exploiting spatial correlation towards an energy efficient clustered aggregation technique (CAG). In *IEEE International Conference on Communications (ICC 05)*, pages 82–98, 2005.
- [96] F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Elsevier/Morgan-Kaufmann, 2004.
- [97] M. Zhao, B. Childers, and M. L. Soffa. Predicting the Impact of Optimizations for Embedded Systems. In *Proceedings of the 2003 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '03)*, pages 1–11. ACM Press, 2003.