Benchmarking In-Network Sensor Query Processing

Qiong Luo Hejun Wu Wenwei Xue Bingsheng He
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{luo, whjnn, wwxue, saven}@cs.ust.hk

Abstract

In-network sensor query processing systems are used for power-efficient sensory data acquisition and aggregation in wireless sensor networks. Due to the cross-layer design of these systems and the resource-limited and noisy nature of WSNs, it is challenging to study the end-to-end performance of these systems in a realistic setting. In this paper, we design and implement a benchmark, Bisque, for this purpose. We identify the components of an SUT (System Under Test), set the network topology, database schema and population, select nine queries, as well as fix the performance metrics and scaling factors for the benchmark. We apply Bisque to TinyDB and its variations with different network routing protocols and aggregation techniques on a WSN emulation platform. Our initial results show both the strengths and the limitations of current-generation WSN query processing systems.

1. Introduction

Recent years have seen exciting progress on in-network query processing techniques [7][16][17][29] for wireless sensor networks (WSNs). This progress is intertwined with the development of network routing protocols such as AODV (Ad-hoc On-Demand Distance Vector) [4] and Directed Diffusion [14]. As sensor query processing systems involve both resource-limited hardware (e.g., the Crossbow MICA2 motes [8]) and new system software (e.g., the Berkeley TinyOS [23]) with a cross-layer design, it is challenging to study the end-to-end performance of these systems in a realistic setting. In this paper, we make the first attempt to develop a benchmark, Bisque (Benchmark for In-network Sensor Query Processing), for this purpose.

Traditional database benchmarks mainly consider the following five factors: (1) the System Under Test (*SUT*), usually the DBMS under evaluation, (2) the database schema and population, (3) the database workload, (4) the performance metrics, and (5) the scaling factors and other parameters. These issues have been discussed in depth in the classic benchmarking handbook [11] for transaction processing systems and have been well covered in later benchmarks, for instance, OO7 [5] for Object-Oriented DBMS, BUCKY [6] for Object-Relational DBMS, XBench [28] and XMark [18] for XML query processing, and Linear Road [1] for Stream Data Management. Nevertheless, benchmarking in-network sensor query processing requires new considerations on all of these five factors as well as on other factors.

First, we identify the SUT. As an in-network sensor query processing system is networked embedded software, its components on a sensor node are tightly integrated with one another. Typically, there are three major layers in such a system from bottom up: the MAC (Media Access Control) layer, the routing layer, and the query layer. We focus on the two upper layers as they are more closely related to query processing. Especially, given the relatively simple query layer in current systems, the routing layer plays a significant role in query processing and calls for further study from a database point of view.

Next, we consider the database and query workload design. As used in most sensor query processing systems, the database schema is a single relational table of fewer than 20 attributes, including both sensory attributes (e.g., *light*, *temperature*) and non-sensory attributes (e.g., *nodeID*). The database is virtual and dynamic in that the data are flowing out of sensor nodes continuously and are not saved in the network. The workload is also simple, as seen in current WSN monitoring applications [3][17]. It is a set of continuous queries with selection, projection, and aggregation operations. For the benchmark database population, we generate synthetic data based on real-world data sets using spatial and temporal interpolation in order to both preserve the characteristics of the real-world data sets and suit the benchmark setup.

We select the performance metrics to be response time, power consumption, and quality of data. These metrics are chosen for real WSN application requirements of an in-network sensor query processing system. In mission-critical applications such as poisonous gas detection or production monitoring, short response time is the key concern. For some outdoor, long-running applications, power consumption is another concern as it may be costly, or impossible to replace the batteries. Finally, data quality is chosen because the environment is noisy and the communication and aggregation techniques may be erroneous.

Moreover, for an in-network sensor query processing system, we set its network topology, scaling factors and

other parameters in Bisque. We design the network to be in a simple but representative n*n grid topology with the network size being one of the scaling factors. The other two scaling factors are the length of the sample interval in a continuous query and the test duration. These scaling factors together determine the amount of data flowing in the network. In addition, wireless transmission range and noise level are set as two system parameters.

With this benchmark design, we implement several SUTs, generate network topologies and data sets at various scales, and test the SUTs on our sensor network emulator VMNet [25]. All SUTs are based on TinyDB [21], a leading and publicly available in-network sensor query processing system, with variations in the routing protocol (TinyDB original, Directed Diffusion [14], or AODV [4]) and in the aggregation scheme (TinyDB original or SKETCH [7]). The network emulator VMNet emulates Crossbow MICA2 motes based wireless sensor networks and allows realistic application-level performance evaluation [27]. All tools and documents are available at the Bisque web site [2] and we expect gradual updates on them with the advances in the state of the art and the growth of our experience.

The remainder of this paper is organized as follows. Section 2 reviews the background and related work on in-network sensor query processing and benchmarking. Sections 3 and 4 present the design and implementation of the benchmark, respectively. We discuss our initial results in Section 5 and conclude in Section 6.

2. Background and related work

The current generation of smart sensor nodes, or motes, is battery-powered tiny devices equipped with limited computation and communication capabilities. The motes communicate with each other using a radio channel in a multi-hop fashion. Figure 1 illustrates a typical setup for in-network sensor query processing. In this figure, the PC runs client-side programs for users to post commands or queries to the sensor network and to receive query results from the network. The sensor node that is connected to the PC is the sink (node). It forwards commands and queries to other nodes and forwards results to the PC. All sensor nodes have network routing and query processing (selection, projection, and aggregation) code. Once a query is injected into the network, these networked nodes run the query epoch by epoch and send results back to the sink in each epoch.

In our work, we focus on continuous (or long-running) queries as opposed to snapshot (or one-time) queries. To study in-network query processing, we categorize these queries into two classes: (simple) data acquisition queries [3][14][17] and aggregation queries [7][13][16][29]. The former simply collects individual sensor readings from all or some nodes in the network, whereas the latter requires some aggregation information (e.g., SUM, MAX) about the sensor readings. In the following, we give an overview of the state-of-the-art in-network processing techniques for these two classes of queries followed by a discussion on related work on benchmarking in general and performance evaluation for sensor query processing in specific.

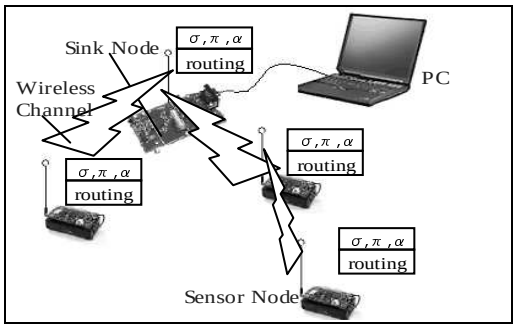


Figure 1. A typical setup of in-network sensor query processing

2.1 Data acquisition queries

As a simple data collection process, a data acquisition query relies on the routing layer to execute the task. Consequently, routing protocols are the major factor for the performance of simple data acquisition queries. Due to the resource-limited and noisy nature of sensor networks, their routing protocols are aimed at efficient power consumption and reliable data transmission. Three representative routing protocols for data acquisition queries are AODV [4] (used in Cougar [29]), Directed Diffusion [14], and the one in TinyDB. [17]

AODV is a reactive routing protocol for ad-hoc mobile networks. It builds a route between two nodes only on the demand of the source node. This protocol has been adopted and enhanced in Cougar to support their query proxy layer for sensor networks. Directed Diffusion, or DD in short, is a data-centric communication paradigm that integrates application-specific semantics into the routing layer. In DD, data is named as attribute-value pairs and is disseminated from source nodes to the node of an interest (query) along multiple paths (multi-paths) for reliability. Each receiving node adaptively reinforces (either positively or negatively) its neighbors based on the data received from them. In comparison, TinyDB uses a cost-based single path routing protocol to build a routing tree, and proposes a number of techniques for data acquisition queries in sensor networks, such as ordering of sampling and predicates and prioritizing packet delivery.

2.2 Aggregation queries

Different from simple data acquisition queries, aggregation queries are not only related to the low-level routing protocols, but are also concerned with the in-network aggregation schemes adopted in the high-level query (application) layer. Two representative in-network aggregation techniques are TAG [16] in TinyDB and SKETCH [7].

In the TAG approach [16], an aggregate is computed bottom-up in the routing tree and many optimization techniques are used to improve the performance, e.g., snooping the shared radio channel, hypothesis testing and adaptive network topology maintenance. Most recently, Considine et al. [7] studied approximate aggregation techniques to enable fault-tolerant, multi-path routing. This goal is achieved by converting duplicate-sensitive aggregates, such as COUNT and SUM, into duplicate-insensitive sketches before transmission. Although this approach may increase communication overhead, the authors argue that their techniques are indifferent to the failures in nodes and in network links. Consequently, the quality of data is improved in comparison with TAG.

Additionally, both Cougar and Directed Diffusion considered aggregation processing. Cougar's approach is to divide an aggregation query plan into a number of flow blocks [29] whereas Directed Diffusion investigated packet merging and duplicate elimination (not exactly SQL aggregates) [13].

2.3 Benchmarking and performance evaluation

Benchmarking is a long-lasting research topic in the database community [11]. There have been a number of successful domain-specific benchmarks proposed in the literature, for example, the TPC-x family benchmarks [24] for transaction processing. Later benchmarks such as BUCKY [6], Linear Road [1], OO7 [5], XBench [28], and XMark [18] are also excellent examples. We learn from the design methodology of these previous domain-specific benchmarks, and design and implement our own for in-network sensor query processing. We consider both data-centric routing and data aggregation schemes in our benchmark and aim at creating a realistic and representative setting for the performance study. To our best knowledge, Bisque is the first benchmark for in-network sensor query processing.

There have been a number of performance studies on sensor networks using real-world deployment, including the GDI (Great Duck Island) project [12] and the research deployment in BBQ [9]. GDI deployed sensor networks in outdoor environments mainly to study the power consumption and data loss rate of the deployed network. The research deployment in BBQ was used to evaluate the power consumption and accuracy of their model-driven data acquisition techniques. In comparison, our work formalizes the key issues such as SUTs, topology, metrics, and scaling factors for general WSN query processing. With this formalization as the basis, we study the performance of different techniques under various configurations.

Finally, there are also a few performance studies on sensor networks in simulation/emulation environments. The two that are most relevant to our work are those using PowerTOSSIM [19], a TinyOS simulator, and the TAG simulator [7][16]. The study done on PowerTOSSIM [19] instrumented TinyOS code to measure the power consumption of a set of simple programs (for example, the Blink application that makes the lights on a sensor node blink) and a simple TinyDB query running on the simulator. In comparison, the TAG simulator was used to study the power consumption of TAG and the centralized aggregation [16], as well as the result accuracy and the total packet size of SKETCH [7]. Similarly, we use our sensor network emulator to conduct realistic, application-level performance evaluation. However, our focus is to design a benchmark for in-network query processing and this benchmark can be used with other evaluation platforms.

3. Benchmark design

Bisque is designed as a general benchmark for in-network query processing systems. The purpose of Bisque is to reveal the main performance characteristics of these systems while keeping the major performance factors controllable. The key elements of Bisque include SUTs, network topology, data schema and population, query workload, performance metrics, and scaling factors. In the following, we present the formalization of these elements.

3.1 System under test

The system-under-test (SUT) in Bisque is an *in-network sensor query processing system*. A three-layer illustration of a typical SUT is shown in Figure 2.

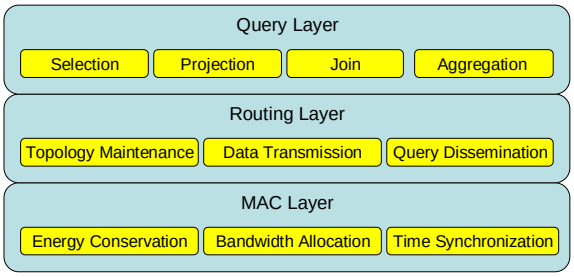


Figure 2. Three-layer illustration of an SUT

The MAC (Medium Access Control) layer of an SUT manages the wireless, multi-hop radio communication channel of a WSN. Its tasks include energy conservation, bandwidth allocation, and time synchronization. The routing layer maintains the network topology to disseminate queries from the sink node into the network and to transmit sensor readings in the network. Finally, the query layer executes declarative, SQL-style database queries. In this layer, relational query operators are applied to the sensor readings before the data are transmitted by the routing layer. The query layer is also responsible for metadata management and power-aware query optimization.

This layered SUT architecture in Figure 2 is an abstraction of current in-network sensor query processing systems, but not a requirement of Bisque. The components of different layers may mix with one another in a specific implementation of an SUT.

3.2 Network topology

As in common practice, the network topology in Bisque is defined with three factors: the area, the node positions, and the network links. The sensor nodes are deployed at the specified positions in a two-dimensional area of a specified shape and size. Given the wireless transmission range of each node, a network link exists between two nodes if their distance is within the transmission range of each node and no obstacles block the wireless signal between them.

To simplify performance analysis, similar to the previously adopted topologies [7], Bisque defines the area as a square of width *w*, in which *m*m* nodes (including a sink) are deployed at the cross points of an *(*m*-1)*(m-1)* grid with the cell width *a= w/(m-1)*. Without causing confusion, we call this grid an *m*m* grid in the remainder of the paper. The position of each node is denoted as coordinate (*x*,*y*), with the node at the lower left corner being (*0*,*0*) and the node at the upper right corner (*w*,*w*). The sink resides at or near the center of the area: if *m* is odd, it is at (*w/2*,*w/2*); otherwise, it is at (*(m-2)a/2*, *(m-2)a/2*).

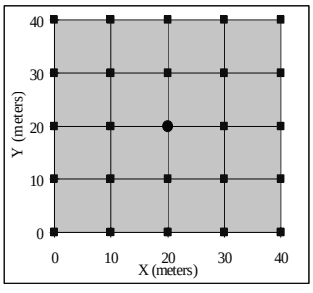


Figure 3. A 5*5 grid deployment

Figure 3 shows an example of a 5*5 grid deployment. The sink is shown as the dot in the center. If the transmission range of all nodes is 15 meters, the maximum number of hops from a node to the sink is 2 (e.g., from (0,0) to the sink (20,20)).

3.3 Database schema and population

The Bisque database schema consists of a single virtual relational table named *sensors*. Such virtual table abstraction has been widely used by previous research work on sensor databases [17][29]. The schema contains both *sensory attributes* that represent common types of sensors today (e.g., temperature, light and microphone sensors), and *non-sensory attributes* (e.g., nodeID, timestamp, and location attributes). We omit the schema for space consideration. It is available in the Bisque specification [2].

Because sensor networks are tightly embedded in the physical world, sensory data collected from a specific network deployment usually reflect the spatio-temporal trends and correlations of the phenomena being monitored in the environment. Therefore, we consider using a real-world data set as the Bisque database population.

There are many data sets publicly available in the sensor network research community, e.g., the Intel Lab data set [15] and the GDI [12] data set. After analyzing these real-world data sets, we pick the Intel Lab data set as the recommended data set for Bisque. The reason is that this data set comes with a detailed network topology diagram and has a sufficient number of readings for each node in the topology.

In order to fit the data set to different sample intervals, network topologies and test durations, we have designed a data generator based on a simple spatio-temporal interpolation model. Given an original data set and the user requirements for the target data set (e.g., the locations of the nodes, the sample interval, and the duration of the data set), the data generator outputs a synthetic data set. The code and description of this data generator is available at the Bisque Web site [2].

Using the original Intel Lab data set as the input to our data generator, we generate three synthetic data sets of different scales (small, medium and large) as the default Bisque database populations. The three data sets contain sensor readings for a 9-node (3*3), 25-node (5*5), and 49-node (7*7) grid network topology, respectively.

The first reading of each node in a data set has the same timestamp. Each reading contains four fields: *nodeid*, *timestamp*, *light* and *temp*. In a data set, there are 1000 readings for each node (except for the sink) with a fixed interval of one second. Therefore, there are totally 8K, 24K and 48K sensor readings in the three data sets.

To give a rough idea about the spatial and temporal characteristics of our data sets, Figure 4 shows the mean

values of the light readings of each node in the 49-node data set, and Figure 5 the light readings of five representative nodes in the 49-node data set over time.

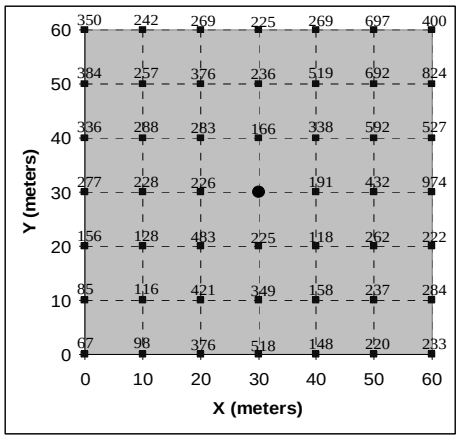


Figure 4. Mean light readings of nodes in the 49-node data set

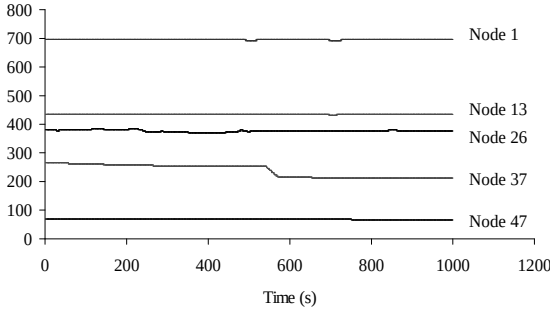


Figure 5. Light readings of five nodes in the 49-node data set

3.4 Query workload

The goal of the Bisque query workload design is to reveal the performance characteristics of in-network query processing techniques. The query workload attempts to include: (1) queries that are most common and useful for real-world sensor network applications, and (2) queries that are simple but can serve as basic building blocks for more complex queries.

In this section, we present the nine SQL queries in the current version of Bisque query workload. In the workload, Q1-Q5 are data acquisition queries (Section 3.4.1) and Q6-Q9 are aggregation queries (Section 3.4.2). All queries in the workload are continuous queries. For the simplicity of presentation, the SAMPLE INTERVAL clause of all queries is omitted. The sample interval of these queries is one of the scaling factors in Bisque, which is discussed separately in Section 3.6.

3.4.1 Data acquisition query workload

Q1: Single Sensory Attribute Projection

```
SELECT   nodeid, light
FROM     sensors
```

Q1 is the simplest query in the workload and allows a basic performance comparison of different routing protocols of the SUTs. It projects a single sensory attribute for all nodes in the network. The non-sensory attribute *nodeid* is also projected in order to identify what node a tuple is from. Since what sensory attribute to project does not affect the relative performance of SUTs, *light* is chosen as an arbitrary sensory attribute to project in this query.

Q2: Projection of Multiple Sensory Attributes

```
SELECT   nodeid, light, temp
FROM     sensors
```

Q2 projects multiple sensory attributes for all nodes in the network. The purpose of this query is to investigate the effect of number of sensory attributes on different routing protocols. The number of attributes projected in the query can be increased as necessary.

Q3: Single Sensory Attribute Projection and Selection

```
SELECT   nodeid, light
FROM     sensors
WHERE    light > C
```

Q3 studies the performance of selection queries on a sensory attribute. In comparison with Q1, this query adds a WHERE clause with a selection predicate on the projected *light* sensory attribute.

In each epoch (sample interval) of the query, only those nodes whose current light readings satisfy the predicate will send out their data towards the sink even though all nodes in the network must acquire their own light readings. The set of nodes that satisfy the predicate may vary from epoch to epoch depending on the data.

The parameter C in the predicate is a user-specified constant value. It can be changed to achieve different selectivities of the predicate.

Q4: Conjunctive Selection on Multiple Sensory Attributes

```
SELECT   nodeid, light, temp
FROM     sensors
WHERE    light > C1
AND      temp > C2
```

The query condition of Q4 is the conjunction of multiple selection predicates on sensory attributes. This query is used to investigate the predicate ordering issue in query evaluation. The number of predicates involved in the selection condition can be increased as necessary. C_1 and C_2 are two user-specified constant values.

Q5: Disjunctive Selection on Multiple Sensory Attributes

```
SELECT   nodeid, light, temp
FROM     sensors
WHERE    light > C1
OR       temp > C2
```

Q5 differs from Q4 in that its query condition is disjunctive. The techniques of handling disjunctive conditions, if present, are evaluated by this query.

3.4.2 Aggregation query workload

Q6: Duplicate-Insensitive Simple Aggregation

```
SELECT   MAX(light)
FROM     sensors
```

Q6 tests the performance of the aggregation schemes for duplicate-insensitive aggregates. All nodes in the network participate in the aggregation process.

Q7: Duplicate-Sensitive Simple Aggregation

```
SELECT   SUM(light)
FROM     sensors
```

Q7 tests the performance of the aggregation schemes for duplicate-sensitive aggregates. The duplicate-sensitivity of the aggregate requires extra effort in multi-path routing in order to ensure the correctness of query results.

Q8: Aggregation with Sensory Attribute Selection

```
SELECT   AVG(light)
FROM     sensors
WHERE    light > C
```

In comparison with Q6 and Q7, Q8 adds a selection predicate on the aggregation attribute. The predicate selects a subset of the nodes in the network to participate in the aggregation and this subset may change over epochs of the query depending on the data.

Q9: Aggregation with GroupBy Clause

```
SELECT      AVG(light), loc_x
FROM        sensors
GROUP BY    loc_x
```

Q9 adds a GROUP BY clause to an aggregation query, which results in more communication cost in the network in comparison with Q6-Q8. The increased communication cost is because multiple partial aggregates, each of which is for one group, are transmitted in each epoch.

3.4.3 Discussion

We have considered several other complex queries as candidates to be added to the future version of our Bisque query workload. An example is the following query Q10, which adds a HAVING clause to Q9.

Q10: Aggregation with GroupBy and Having Clauses

```
SELECT      SUM(light), loc_x
FROM        sensors
GROUP BY    loc_x
HAVING      SUM(light) < C
```

Unfortunately, when we used either VMNet or real mote network to test these complex queries, there were often severe problems during the query execution. The problems include high packet loss rates, heavily inaccurate query results, and difficulties in finding a multi-hop route to the sink for distant nodes. We believe these problems are mainly due to the computation and communication resource constraints on the current generation of sensor motes.

3.5 Performance metrics

We select the following three performance metrics for Bisque to evaluate the expected battery life, the query response time, and the result data quality.

(1) Power Consumption within an Interval (*PT*). It is the average accumulated power consumption of a node that processes the query within a time interval. This metric determines the battery life of sensor nodes.

(2) Response Time (*RT*). For a given query, the response time is the average time interval between two consecutive query results arriving at the sink. Note that, given a fixed time duration, the number of results generated by different SUTs may be different for the same query on the same data set and the same network topology. This difference is due to the differences in the routing and query processing techniques of the SUTs, and it affects the response time inversely.

(3) Relative Error Rate (*RE*). Assume at time t, the returned result is V_t and the correct sensory result is C_t. If there are M results produced between time t_1 and t_2, we compute the average result error rate *RE* in Equation 1 as in previous work [7]. The relative error rate measures the accuracy of the query results.

$$RE = \frac{1}{M}\sum_{t=t_1}^{t_2} abs\left(\frac{V_t - C_t}{C_t}\right) * 100\ \%. \quad \text{.....Equation 1}$$

3.6 Scaling factors and other parameters

We define three scaling factors in Bisque:

(1) Number of Nodes, or network size. It is the number of nodes in the WSN, including the sink. This factor affects response time directly. It often takes the sink longer to receive a query result from a node in a larger network than in a smaller one because it may take more hops and

the network contention may be heavier in the larger network.

(2) Sample Interval of Continuous Queries. The sample interval is specified in a continuous query. A node running the query generates a data tuple every sample interval. If the sample interval is short, the node may have to work at a full speed and may have no time to sleep.

(3) Test Duration. The test duration of each query is set based on the sample interval of the query. Bisque specifies the duration to be *10* times long as the sample interval, which is sufficient to get stable performance results.

The two other system parameters are *transmission range* and *noise level*. The transmission range is the maximum distance between a source and a destination node within which the radio signals of the source node can successfully reach the destination node. It determines the link between two nodes in a network. The noise level of an environment describes the noise strength in the wireless channel of a WSN. As it can be measured in an environment and can be classified to describe typical application environments, it is often used to estimate the Bit Error Rate (BER) of the wireless communication [27].

4. Implementation

As the first step of our performance study using Bisque, we selected four well-known SUTs proposed in the literature, called AODV, DD, SKETCH, and TinyDB, respectively. For fair comparison, all techniques (e.g., AODV, DD and SKETCH) are re-implemented to run on TinyOS [23] unless the techniques are released together with the latest TinyOS stable version (1.1.0) (e.g., ACQP [17] and TAG [16]). We have tested all of our implementations using small-scale sensor networks before we evaluate their performance on VMNet. Our results on the emulated networks (ESNs) from VMNet are similar to those on real WSNs of the same configuration.

4.1 SUTs

To control the scale and complexity of the initial performance study, we used the CSMA type MAC layer provided in TinyOS 1.1.0 [17] for all four SUTs. At this stage, we focus on investigating the performance impact of the different aggregation schemes and routing protocols adopted in the SUTs. For reliability, we compressed the query messages of TinyDB to allow a query to be transmitted in one message. Without this compression, the query messages sometimes need to be re-sent to reach the nodes that are far from the sink, which has non-deterministic performance effects. In addition, we added two attributes, *loc_x* and *loc_y*, into the query layer to enable querying the locations of nodes. In short, the major difference among TinyDB, AODV, and DD is in their routing layers. The difference between TinyDB and SKETCH is SKETCH's new aggregation scheme and multi-path routing. In the following, we present the implementation of the SUTs in detail.

TinyDB. We adopted the implementation of TinyDB in TinyOS 1.1.0 package and made the necessary modifications to allow benchmarking. This release of TinyDB includes a subset of the ACQP and TAG implementation.

We chose the version 1.1.0 over the more recent released version 1.1.13 for the code stability consideration. At the current stage, we focus on the widely used single-path cost-oriented routing protocol in the version 1.1.0, and do not compile and link other routing protocols (e.g., MintRoute [26]) released in the later versions for TinyDB. As illustrated by our experimental results in Section 5.2, with an initial but stable implementation, the performance differences between TinyDB and other SUTs are already revealed using Bisque.

AODV. There is an AODV implementation, Tiny AODV, for an early version of TinyOS in the public domain [20]. We adopted the core source code of TinyAODV and modified it to run with the current-version of TinyDB query parser. The two modifications are (1) changing the old message format to match the new TinyOS message format, and (2) adding support for the location and nodeID attributes.

DD. Directed Diffusion was initially implemented and tested using the ns-2 simulator [14]. The authors later developed a micro-edition of DD running on MICA motes, which is called TinyDiffusion [22]. However, the implementation of TinyDiffusion contains only a small subset of the original design ideas presented in the paper. Considering this limitation, we chose to develop our own version of DD without using the publicly-available TinyDiffusion code.

We have implemented the core components of DD for MICA2 motes and TinyOS. This includes the interest and data caching, periodical interest refreshing, gradient set-up and maintenance, initial exploratory source data with low rates, multi-path routing, path reinforcement and negative reinforcement.

We have made our best effort to follow the exact design ideas of DD. Nevertheless, in order to facilitate benchmarking, we have made several minor modifications. We describe these modifications in order.

First, DD was designed and implemented using an animal tracking application example in the original paper. In comparison, the Bisque query workload consists of declarative database queries. Consequently, we reuse the high-level query layer source code of TinyDB and only replace the routing components of TinyDB with that of DD we developed.

Second, the authors considered a peer-to-peer network scenario in the original paper of DD, in which every node in the network can be both a data source and a sink. In comparison, Bisque is targeted for an in-network sensor query processing scenario, where all nodes in the network are source nodes but there is only one sink connected to the base station. Therefore, we disable the sink module of DD on all nodes in the network except for the node connected to the base station.

Finally, DD is based on multi-path routing and always assumes that aggregates are duplicate-insensitive. In order to ensure the correctness of query results for duplicate-sensitive SQL aggregates, such as COUNT and SUM, we change the multi-path routing of DD to single-path routing for these aggregates (Q7-Q9 in Bisque query workload).

SKETCH. We implemented SKETCH [7] on top of TinyDB for three aggregation operators, including SUM, COUNT, and AVG. We followed most of the implementation strategies proposed in the original SKETCH paper [7]. For benchmarking, we have made the following modifications:

(1) We modified the routing layer of TinyDB to enable multiple paths, since SKETCH was proposed to run in multi-path sensor networks.

(2) Instead of using *20* 16-bit bitmaps in the original SKETCH paper [7], we used *10* 16-bit bitmaps for sketches. With the original compression scheme in SKETCH, this setting yields a good tradeoff between data accuracy (quality) and power consumption in our experiments.

(3) We used the hash function proposed by Flajolet [10], as we have no information about the exact hash function used in the original SKETCH.

4.2 Testbed

We used our VMNet [25] network emulator to perform the experiments. It emulates MICA2 motes based WSNs and reports performance results on power consumption, execution time as well as relative error rate in query results. The emulation is trace-driven and the queries run in the virtual (emulated) time. For instance, if the sample interval of a query is 10s and the readings of each node in the input data set are one second apart, VMNet will take one out of every 10 readings of nodeID *k* from the input data set and feed it to node *k* in each sample interval.

We have performed extensive experiments to validate VMNet. The results show that VMNet is able to estimate power consumption and execution time realistically. The relative performance difference between different network configurations reported in VMNet is close to that measured in real WSNs. The mechanisms of estimating the power consumption and the execution time of nodes in an emulated sensor network (ESN) are described in detail in our technical report [27]. The computation of other metrics for Bisque is described in the following.

First, the query response time. VMNet logs the timestamps of sensory results at each node. The query response time *RT* is computed using Equation 2, where *n* is the total number of results within the predefined test duration, and *Ti* the timestamp of the *i*th result received at the sink.

$$RT = \frac{1}{n-1}\sum_{i=2}^{n} T_i - T_{i-1} \quad \text{.....Equation 2}$$

Second, the relative error rate. The relative error rate is computed using the input data set and the logged query results. According to Equation 1 in Section 3.5, the relative error rate needs the correct result *Ct* of a query at timestamp t, and the returned result *Vt*. Since each tuple in the data set bears a timestamp, VMNet uses a tool to run the query on the data set directly and computes the correct result *Ct* at timestamp *t*. The returned result *Vt* is logged in VMNet at the virtual time *t*. Thus, the error rate can be computed.

5. Experiments

5.1 Experimental setup

We used three PCs, each of which was equipped with a Pentium® 3.2GHz CPU, 1GB RAM, and a 40GB hard disk. The operating system of the three PCs was Red Hat Linux 9.0. All PCs ran experiments independently. Because VMNet estimates response time and power consumption using instruction-level emulation of sensor nodes, different speeds of PCs do not affect the experimental results.

The target sensor nodes that VMNet emulated in our experiments were Crossbow MICA2 motes [8]. The sensor board was MTS300CA and the processor board MPR 410CA. The sink consisted of an MIB510 interface board and an MPR410CA processor board. The version of TinyOS and TinyDB in the experiments was 1.1.0.

We used 9, 25, and 49-node emulated sensor networks (ESNs). The 5*5 ESN is shown in Figure 3, and 3*3 and 7*7 ESNs are similar to the 5*5 ESN. We intentionally kept the network size to be under 100 to study the detailed performance of the SUTs. Testing on large-scale networks with multiple sinks is one direction of our future work.

The noise level of the environment was configured to be -13 dB, which was an experiential value for a typical lab environment as ours. We set the path loss parameters to get a transmission range of 15 meters. This transmission range setting came from our experience with that of the MICA2 sensor motes.

We set the remaining scaling factors as follows. When the sample intervals of queries are 2s, 10s and 30s, the

corresponding test durations are 20s, 100s and 300s. We select the length of the interval as 60s when measuring the power consumption within an interval (*PT*), denoted as *PT(60s)*.

For the constant parameters in the Bisque queries, we set C = 130 in Q3 and Q8; C_1 = 130 and C_2 = 590 in Q4 and Q5. As a result, the selectivities of Q3 and Q8 are both 90%, and those of Q4 and Q5 are 68% and 96%, respectively. We define the *epoch selectivity* as the average of the query selectivities for all nodes in an epoch. The maximum and minimum selectivities of each query for all epochs and all nodes are within a 3% difference of the epoch selectivity. We excluded low-selectivity queries because we found that the performance differences between the SUTs were insignificant when few nodes had results. We designed high selectivities in Q3, Q5, and Q8 to show the effects of additional computation in queries with predicates when the amount of data transmitted is not changed much. In Q4, we designed the medium selectivity to study the effects of selectivity on the performance of the SUTs.

The SUTs were started at the same time after VMNet started. Because the sink is usually equipped with an external power supply and without a sensor board attached, we excluded the sink from the performance evaluation. Since SKETCH is designed for duplicate-sensitive aggregation queries, we did not evaluate its performance for duplicate-insensitive aggregation queries (e.g., Q6) or data acquisition queries.

5.2 Experimental results

As the first step, we focus more on comparing the general performance trends of different SUTs than on studying the characteristics of individual SUTs. Instead of flooding the reader with results on all combinations of setups, we identify four main factors for the performance comparison and describe their effects in order.

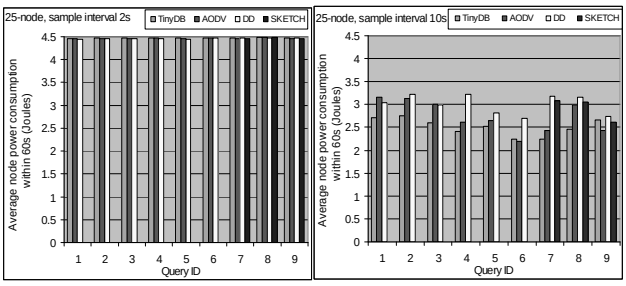
5.2.1 Sample interval

We tested the performance of the SUTs using queries of different sample intervals with other factors fixed. We found that a sample interval longer than 2 seconds was necessary in order to show the performance differences between SUTs on current-generation WSNs. This requirement was most prominent for the comparison on power consumption within a fixed duration (*PT*).

As shown in Figure 6a, when the sample interval was two seconds (2s), the power consumption within 60s (*PT(60s)*) of each SUT differed less than 1% and the average *PT(60s)* of the SUTs was about 4.465 Joules. When the sample interval was 10s, the power consumption decreased significantly for all SUTs and the difference between SUTs became considerable. The reason is that, with a short sample interval of 2s, the nodes keep active [17] (no sleeping) and the power consumption in operations other than sleeping does not differ much [8]. In contrast, a longer sample interval (e.g., 10s and 30s) allows the sensor nodes to sleep for a period after their tasks are finished within each sample interval. Furthermore, the shorter the sample interval, the heavier the network traffic, as each node with satisfying query results tries to send one result per sample interval.

Figure 6b shows that on average, TinyDB was the most power-saving SUT when the sample interval was 10s. Its average node power consumption was 13% and 9% less than that of DD and AODV, respectively. This power saving in TinyDB is mainly due to its routing tree setup. As TinyDB keeps a neighbor table that caches the information about the parent node and children nodes, the route managing overhead is reduced after the routing tree is set up.

In our experimental results, an even longer sample interval of 30s did not affect the relative difference in the power consumption of the SUTs and its average power consumption of each SUT was about 25% less than that of 10s. The longer sample intervals of 10s and 30s did not affect the relative difference between SUTs was because, as we found out in the source code of TinyDB, the operation flows were similar for long sample intervals and the difference was determined by the length of the sleeping time [21]. Therefore, in the following experiments, we fixed the sample interval to be 10s, which was sufficient to show the relative difference between the SUTs.



a. sample interval 2s **b. sample interval 10s**

Figure 6. Average *PT*(60s) of the 25-node ESN

5.2.2 Network size

We next measured the performance of the SUTs running in different sizes of ESNs. The results show that the network size affected both response time and data quality. The average power consumption did not differ much (less than 8%) among 9, 25, 49-node ESNs, even though it was

slightly more in a larger network than that in a smaller one. The small difference was because the number of hops was small (3 at the maximum) and the distance between nodes was the same for all ESNs.

We pick Q4 and Q5 as the representative data acquisition queries to show the effects of network size. Figure 7 shows that the 49-node ESN had the longest query response time, and the 25-node ESN the shortest for all SUTs. The reason is as follows. In processing data acquisition queries, each node tries to send its query result to the air when it has one. Because the selectivities among different ESNs were similar, the 9-node ESN had the fewest nodes sending query results within a sample interval, whereas the 49-node had the most. When the number of result-sending nodes increased, the response time would decrease in general. This was true when the network size increased from 9 to 25. However, because the network contention was more severe in the 49-node ESN, especially around the sink, the sink node received fewer query results than that in the 9-node and 25-node ESNs. Consequently, the 49-node ESN got the longest response time.

Compared with other SUTs in Figure 7, AODV always had the shortest response time in any size of ESNs. This is because the transmission mechanism of AODV is more reliable than that in other SUTs. In AODV, a source node posts requests to other nodes for forwarding its result. Other nodes will reply if they are ready for receiving. The source node will not send its data until it receives the reply. In contrast, nodes in TinyDB and DD send data directly without going through the request-reply process first. This may result in network contention and packet loss. Consequently, there were more results arriving at the sink within a sample interval in AODV than in other SUTs and the response time in AODV was the shortest.

The response time of the aggregation queries was similar for all SUTs because the network traffic was not heavy due to in-network aggregation. Hence, the sink got an aggregation query result every sample interval no matter what the SUT was.

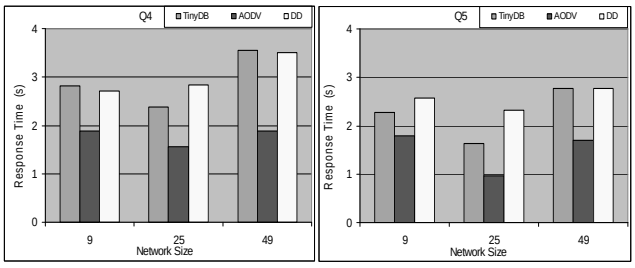


Figure 7. Average *RT* of the ESNs

The relative error rates (RE) of queries other than Q7 (SUM) were all smaller than 5% on different sizes of ESNs and are therefore omitted from discussion. Figure 8 presents the relative error rates of Q7 in the three ESNs. For the three SUTs other than SKETCH, the relative error rates of Q7 increased with the increase of the network size. The reason is mainly related to packet loss. When the network size increases, the network contention becomes more severe and thus more packets are lost in the network.

Interestingly, SKETCH had the smallest *RE* in the 49-node ESN but the largest in the 25-node ESN. Two factors are involved; one is the packet loss rate, and the other the approximation error in the SKETCH hash function. According to the property of the hash function, the approximation error is the lowest if the data distribution is uniform. The more nodes in the network, the better the uniformity of data distribution in the Bisque data set. Therefore, the approximation error of the hash function becomes smaller when the network size increases.

In the 9-node ESN, although the approximation error of the hash function was the largest, its packet loss rate was the smallest among the ESNs and dominated the data quality. Hence, its data quality was better than that in the 25-node network. In contrast, in the 49-node network, although the packet loss rate was the largest, the approximation error decreased sharply and dominated the data quality. Consequently, the 49-node ESN got the smallest relative error rate.

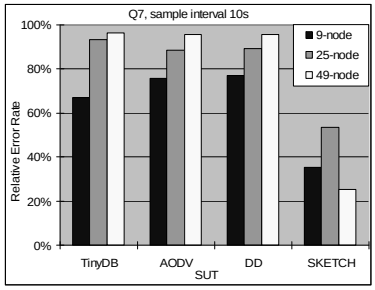


Figure 8. Relative error rate of the SUTs running Q7

5.2.3 Aggregation scheme

In the following, we discuss the effects of the two in-network aggregations schemes: TAG in TinyDB and SKETCH. Although our implementation of AODV and DD included the TAG scheme, we exclude them from discussion, because the effects of TAG in them are not well differentiated due to their complex path management mechanisms. As power consumption is a major concern for aggregation, we focus on comparing the power consumption of aggregations queries Q6-Q9 in TinyDB and SKETCH.

Q6-Q9 in Figure 6b shows the effect of in-network aggregation. In TinyDB, Q6-Q8 consumed less power to

process than acquisition queries on average. This indicates that the in-network aggregation technique decreases wireless communication and thus saves power consumption, since the partial aggregation in internal nodes avoids forwarding all data to the sink [16][29].

However, TinyDB cost a similar amount of power in Q9 to those in acquisition queries. This exception is mainly due to the group by clause in Q9, which requires multiple groups to send their results to the sink. If an internal node finds that its children and itself are not within the same group, it does no partial aggregation but forwards the results instead. Hence, there are network paths that have no partial aggregation for a group-by query, which increases the power consumption. In addition, Q9 is more complex and needs more computation to process than other queries, which also increases the power consumption.

Figure 6b also shows that SKETCH cost about 32% more power in Q7 and Q8 than TinyDB. This extra power consumption was due to the multi-path routing and the additional computation of the SKETCH algorithm. However, SKETCH achieved the lowest relative error rate in Q7 as shown in Figure 8. This indicates that SKETCH improves the data quality of duplicate-sensitive queries at the price of more power consumption.

5.2.4 Queries

After studying the average performance of SUTs in processing all Bisque queries, we examine the performance of individual queries on the SUTs.

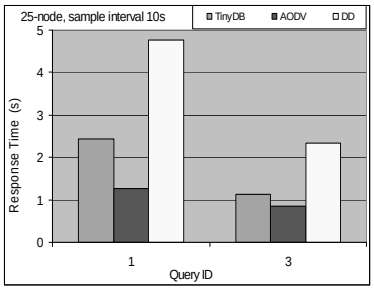


Figure 9. Average *RT* of Q1 and Q3 in the SUTs

We have seen the power consumption of all queries in Figure 6b and the response time of two representative queries Q4 and Q5 in Figure 7. In Figure 9 we further show the response time of Q1 and Q3. Putting together results in Figure 6b and Figure 9, we see that even though Q1 is the simplest query in the Bisque query workload, it did not necessarily lead to the least *PT* and *RT*. Especially, Q3, which differs from Q1 only with a selection predicate of 90% selectivity, achieved much shorter response time and slightly smaller power consumption. Since Q3 produces 10% fewer results than Q1, its network contention was less severe than Q1's. Consequently, more query results of Q3 arrived at the sink than those of Q1 in each sample interval, which translates into a shorter response time for Q3. Also because of the lighter network contention, Q3 consumed less power than Q1.In general, the difference in the power consumption between acquisition queries was small in Figure 6b. Q3-Q5 consumed about 13% less power than Q1 and Q2, on average. The reason is that Q1 and Q2 require all nodes to send results whereas Q3-Q5 have selection predicates. Among Q3-Q5, Q4 had the smallest selectivity and consumed the least power on TinyDB and AODV. However, DD consumed more power in processing Q4 than that in Q5. The reason, as we found in the experiments, is the larger overhead in path management when running Q4 than Q5 on DD. Because the sensory data keep changing, some nodes sometimes have satisfying data and sometimes not. This causes frequent changes of reinforced paths, which costs additional power to manage, as happened in Q4.

The smaller selectivity of Q4 leads to a larger response time than that of Q5 on average (Figure 7). This effect was because there were much fewer query results of Q4 than that of Q5 within a sample interval, as the selectivity of Q4 was about 30% smaller than that of Q5 on average.

On average, the aggregation query Q6 consumed the least power among all queries (Figure 6b). Two reasons are involved. First, as stated in the previous sections, in-network aggregation decreases the power consumption. Second, Q6 is the simplest aggregation query in the Bisque query workload since it queries only one attribute and the operation is MAX, which is simpler to compute than AVG and SUM.

Finally, Q7 had the largest relative error rate among all aggregation queries. This is because Q7 computes SUM of data for all nodes, which is more sensitive to packet loss than queries with other aggregates (MAX and AVG).

5.3 Discussion

Having analyzed the performance of the SUTs with different factors varied, we summarize the main performance results, which are consistent with those in the previous work [7][14][16][17].

(1) Long sample intervals save power consumption. TinyDB was the best power-saver and DD the weakest.

(2) AODV had the shortest response time, 28% faster than other SUTs on average.

(3) SKETCH significantly improved the relative error rate of the duplicate-sensitive aggregation queries.

In addition to the performance results, we find that even though the electric current in transmitting was the largest [17][27], the nodes in ESNs spent 80%-95% of their energy in listening and receiving. Since a significant share of this

power consumption in listening and receiving was wasted (called overhearing), this finding points out a possible direction to further improving the power consumption of WSNs.

6. Conclusion and future work

In this paper, we present the design and implementation of Bisque, a benchmark for in-network sensor query processing. We have selected a number of SQL queries in Bisque to represent various data acquisition and aggregation workloads in a realistic WSN environment. We have implemented several existing in-network sensor query processing techniques for these two types of queries and have conducted experiments to evaluate and compare their performance. As our purpose in this work is to evaluate our benchmark design and to gain the insights into representative query processing techniques, we have not included other SUTs at this stage.

The design and implementation of a sensor query processing benchmark is a brand-new topic for us and there is little previous work in the field. Fortunately, we learn from previous domain-specific benchmarks and focus on the embedded nature of WSNs such as resource limitation and cross-layer design. In addition, the resources on WSN research in the public domain, such as TinyOS and TinyDB code bases and forums, have been great help for our implementation. Following this tradition, we have put our benchmark together with its implementation on the Web [2]. It is our hope that our benchmark will interest and help more people in the community to work on sensor query processing.

Future work includes analyzing our experimental results in greater detail, designing and testing more comprehensive query workloads, and evaluating the techniques using larger-scale platforms.

References

[1] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear Road: A Stream Data Management Benchmark. VLDB, 2004.

[2] Bisque. http://www.cs.ust.hk/bisque

[3] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Querying the Physical World. IEEE Personal Communications, Vol. 7, No. 5, 2000.

[4] Ian D. Chakeres and Elizabeth M. Belding-Royer. AODV Routing Protocol Implementation Design. WWAN, 2004.

[5] Michael J. Carey, David J. DeWitt, and Jeffrey F. Naughton. The OO7 Benchmark. SIGMOD Record, Vol. 22, No. 2, 1994.

[6] Michael J. Carey, David J. DeWitt, Jeffrey F. Naughton, Mohammad Asgarian, Paul Brown, Johannes E. Gehrke, and Dhaval N. Shah.. The BUCKY Object-Relational Benchmark. SIGMOD, 1997.

[7] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate Aggregation Techniques for Sensor Databases. ICDE, 2004.

[8] Crossbow Inc. http://www.xbow.com

[9] Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein, and Wei Hong. Model-Driven Data Acquisition in Sensor Networks. VLDB, 2004.

[10] Philippe Flajolet and G. Nigel Martin. Probabilistic Counting Algorithms for Data Base Applications. Journal of Computer and System Sciences, Vol. 31, Issue 2, 1985.

[11] Jim Gray (Editor). The Benchmark Handbook for Database and Transaction Processing Systems. 2nd edition, Morgan Kaufmann, 1993.

[12] Habit Monitoring on Great Duck Island. http://www.greatduckisland.net

[13] Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. ICDCS, 2002.

[14] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. MOBICOM, 2000.

[15] Intel Lab Data. http://berkeley.intel-research.net/labdata

[16] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. OSDI, 2002.

[17] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The Design of an Acquisitional Query Processor for Sensor Networks. SIGMOD, 2003.

[18] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. XMark: A Benchmark for XML Data Management. VLDB, 2002.

[19] Victor Shnayder, Mark Hempstead, Borrong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. SenSys, 2004.

[20] TinyAODV. http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/tinyos/tinyos-1.x/contrib/hsn

[21] TinyDB. http://telegraph.cs.berkeley.edu/tinydb

[22] TinyDiffusion. http://www.isi.edu/scadds/papers/tinydiffusion-v0.1.pdf

[23] TinyOS. http://www.tinyos.net

[24] TPC (Transaction Processing Performance Council). http://www.tpc.org/

[25] VMNet. http://www.cs.ust.hk/vmnet

[26] Alec Woo, Ternence Tony, and David Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. SenSys, 2003.

[27] Hejun Wu, Qiong Luo, Pei Zheng, and Lionel M. Ni. VMNet: Realistic Emulation of Wireless Sensor Networks. Technical Report HKUST-CS05-05, HKUST, 2005.

[28] Benjamin Bin Yao, M. Tamer Özsu, and Nitin Khandelwal. XBench Benchmark and Performance Testing of XML DBMSs. ICDE, 2004.

[29] Yong Yao and Johannes Gehrke. Query Processing for Sensor Networks. CIDR, 2003.