SensorBench: Benchmarking Approaches to Processing Wireless Sensor Network Data

Ixent Galpin
Dpto. de Ingeniería
Universidad Jorge Tadeo Lozano
Bogotá, Colombia
ixent@utadeo.edu.co

Alan B. Stokes
School of Computer Science,
University of Manchester
Manchester, UK
stokesa6@cs.man.ac.uk

George Valkanas
Dept. of Informatics and Telecommunications
University of Athens
Athens, Greece
gvalk@di.uoa.gr

Alasdair J. G. Gray
Dept. of Computer Science,
Heriot-Watt University
Edinburgh, UK
a.j.g.gray@hw.ac.uk

Norman W. Paton
School of Computer Science,
University of Manchester
Manchester, UK
norm@cs.man.ac.uk

Alvaro A. A. Fernandes
School of Computer Science,
University of Manchester
Manchester, UK
alvaro@cs.man.ac.uk

Kai-Uwe Sattler
Databases and Information Systems Group
Ilmenau University of Technology
Ilmenau, Germany
kus@tu-ilmenau.de

Dimitrios Gunopulos
Dept. of Informatics and Telecommunications
University of Athens
Athens, Greece
dg@di.uoa.gr

ABSTRACT

Wireless sensor networks enable cost-effective data collection for tasks such as precision agriculture and environment monitoring. However, the resource-constrained nature of sensor nodes, which often have both limited computational capabilities and battery lifetimes, means that applications that use them must make judicious use of these resources. Research that seeks to support data intensive sensor applications has explored a range of approaches and developed many different techniques, including bespoke algorithms for specific analyses and generic sensor network query processors. However, all such proposals sit within a multi-dimensional design space, where it can be difficult to understand the implications of specific decisions and to identify optimal solutions. This paper presents a benchmark that seeks to support the systematic analysis and comparison of different techniques and platforms, enabling both development and user communities to make well informed choices. The contributions of the paper include: (i) the identification of key variables and performance metrics; (ii) the specification of experiments that explore how different types of task perform under different metrics for the controlled variables; and (iii) an application of the benchmark to investigate the behavior of several representative platforms and techniques.

SSDBM '14, June 30 - July 02 2014, Aalborg, Denmark
Copyright 2014 ACM 978-1-4503-2722-0/14/06$15.00.
http://dx.doi.org/10.1145/2618243.2618252

Categories and Subject Descriptors

[**Information systems**]: Database performance evaluation; [**Networks**]: Sensor networks

General Terms

Performance, Experimentation

Keywords

Benchmarks, Query processing, Wireless Sensor Networks

1. INTRODUCTION

Wireless sensor networks (WSNs) can be used to support a variety of applications, for example, in environment monitoring or agriculture (e.g. [6, 16]). However, the resource-constrained nature of the sensor nodes, which tend both to have limited computational capabilities and readily drained batteries, means that in-network applications must make judicious use of these resources. As a consequence of the high energy costs of wireless communications, directly shipping all sensed data out of the network is likely to lead to network lifetimes that are much shorter than when some data filtering, aggregation or analysis is carried out in the network, and additional benefits can be gained by effective decision-making relating to routing and caching.

To support the efficient use of WSNs, database researchers have developed both in-network data analysis techniques (e.g. [11, 32, 35]) and sensor network query processors (e.g. [2, 7, 18, 22, 24]). As a result, there are now many different proposals that occupy different positions in a complex design space. Publications on individual proposals invariably include evaluations that report performance for specific tasks on particular platforms. However, these evaluations differ in terms of the tasks carried out, the networks over which they are carried out, the variables that are controlled, and the values that are reported. As a result, it can be difficult to compare proposals in terms of their functionality or performance on the basis of published results, and it can also be difficult to ascertain the impact of different design decisions.

This paper proposes a benchmark for comparing the performance characteristics of different *data processing techniques* for WSNs. The term *data processing technique* encompasses a broad spectrum, including bespoke, hand-crafted software for particular analyses, and more general purpose query processing engines. The benchmark is intended to support analysis of the performance of new and existing proposals, by providing a collection of representative tasks over different network topologies, with a range of control variables that reflect the requirements of different deployments and users. In addition, by providing a standard collection of tasks and variables, the benchmark provides a foundation for more systematic comparisons between existing techniques and new proposals.

The contributions of this paper are as follows:

1. The identification of a collection of tasks, variables and performance metrics that represent the functional and non-functional requirements of a wide range of common applications.

2. The specification of experiments that capture various of the trade-offs that characterize and constrain sensor network deployments.

3. The application of the benchmark to analyse several different and representative data processing techniques, including a sensor network query processor (SNEE [7]) and hand crafted algorithms.

The benchmark has been developed by groups that have experience developing and evaluating both (rather different) sensor query processors and bespoke in-network algorithms. As discussed in Section 5, the benchmark subsumes many of the published evaluations of data processing techniques for WSNs, and the public availability of scripts to support the running of the benchmark and presentation of results should enable its cost-effective application by the wider community[1].

The remainder of the paper is structured as follows. Section 2 discusses various desiderata that a benchmark for WSNs should exhibit including its scope and limitations, and identifies and motivates the selection of the controlled variables and performance measures used in the benchmark. Section 3 describes the experiments that constitute the benchmark, in terms of the tasks to be carried out and the features from Section 2. Section 4 describes the application of the benchmark and the lessons learned. Related work on benchmarking and performance evaluations for data intensive tasks over WSNs is provided in Section 5, and conclusions are presented in Section 6.

[1]The benchmark, including scripts that provide support for running the data processing techniques evaluated in this paper on the Avrora emulator [38], is available from `https://code.google.com/p/sensorbench/`. The scripts are designed to be easily extensible to accommodate other data processing techniques and/or simulators according to user requirements. An optional script is also provided to enable workloads to be run on the HTCondor High Throughput computing platform [37].

2. BENCHMARK PROPERTIES

2.1 Desiderata and Scope

The benchmark aims to enable users to gain an understanding of trade-offs, limitations and assumptions associated with particular data processing approaches in WSNs. The notion of a data processing technique encompasses a broad range of approaches to process data within WSNs, which fall under three broad categories:

- The *warehousing approach*, which involves shipping raw sensor readings outside of the WSN, for storage and subsequent analysis. In this case there is no in-network processing within the WSN. This approach is likely to be appropriate in cases such as when all the sensor readings need to be stored in a database outside the WSN, and there is sufficient energy to transmit all the data out of the WSN. It may also be desirable in cases when the data requires very computationally intensive processing and, as a result, sending the raw data values achieves higher performance than carrying out in-network data processing. MultihopOscilloscope, a sample data collection application which ships with TinyOS [21], is an example of a software artefact that implements this approach.

- The *bespoke, hand-crafted approach*, which involves the implementation of techniques designed to carry out a specific task within the WSN. This category involves in-network processing to carry out a fixed task within the WSN, such as aggregation or outlier detection. This approach is likely to be useful when it is known that the WSN will be used for a single purpose, and energy savings can be obtained by carrying out the data processing partially or wholly within the WSN. For example, the D3 algorithm [35] carries out outlier detection within the WSN.

- The *sensor network query processing (SNQP)* approach, capable of carrying out a broad range of tasks, designed to handle *ad hoc* user-specified queries. This category involves in-network processing and enables the WSN to be repurposed on-the-fly, and is likely to be useful when the task to be carried out by the WSN is likely to change. Examples in this category include TinyDB [24] and SNEE [7].

It should be possible to apply the benchmark to evaluate the performance of data processing techniques from these three categories. The warehousing approach should, intuitively, be the most resource-hungry approach as it involves shipping all the data out of the WSN, and can therefore be used as a baseline to evaluate the benefits obtained by the other approaches, which involve in-network processing.

Adaptivity. We note the emergence of data processing techniques in each of the three categories that are adaptive, i.e., they have the ability to respond to changes in the computing fabric or environment during data processing, e.g., [1,34]. Adaptation typically incurs a resource overhead, and it is done with the hope of obtaining a future saving as a result of having carried out the adaptation. The benchmark is agnostic as to whether a data processing technique is adaptive or not: any savings (or losses) will be reflected in the performance results obtained when applying the benchmark.

Target Class of Applications. The focus of this benchmark are applications with wireless nodes at fixed locations, which involve sensing data at regular intervals, and where energy is a scarce resource. Typical examples are, for instance, environmental monitoring applications [6, 27]. We note that, in most of the cases reported in the literature, such applications have nodes at fixed locations, and that results are transmitted back to a single gateway node (i.e., requests are not initiated by, and results are not sent to, arbitrary nodes in the network, in a peer-to-peer fashion). Thus, current real-world deployments, which are the focus of this benchmark, tend to be relatively simple. In the future, WSNs are likely to become more pervasive and increasingly complex and, as such, it is likely that WSN benchmarks will need to be extended to cover scenarios of increasing complexity.

We further note that this benchmark considers deployments energy in which is considered a scarce resource, as nodes rely on a non-replenishable source of energy, i.e., they are not connected to the mains power, or have the ability to periodically replenish themselves of energy, and therefore the preservation of energy tends to be of paramount concern. Thus, they are significantly different to applications that run on MANETs [15], which comprise mobile nodes which typically are replenished at regular intervals, and in which requests are typically initiated by arbitrary nodes in the network.

Platforms. Given the plethora of WSN hardware available [14], the benchmark aims to be agnostic with regards to the platform being used (i.e., the hardware/OS). As with any benchmark, it will be necessary to provide the hardware specifications alongside the results obtained. These should include information such as the specifications of the hardware (CPU, radio, RAM, flash memory) of each node in the WSN, the energy resources available (battery type and energy supply in Joules).

Use of Simulation. The benchmark is not designed to measure hardware performance but rather to contrast different styles of processing WSN data. As such, the benchmark is not suitable to differentiate between the performance of alternative hardware platforms but rather between different approaches on how to process the data. While it might at first seem desirable to run SensorBench on a WSN testbed with real hardware, e.g., Motelab [42], or indeed, a real-world deployment, it is unlikely to be practical to do so, because some of the workloads are designed to contrast the consequences of doing, or not, all the processing outside the network when one expects the lifetime of the deployment to be in order of months or years. Running the benchmark at real time would imply waiting very long times for the measurements to be final. While testbeds provide more accurate performance measurements than simulator/emulators (such as, e.g., Avrora [38] or PowerTossim [33]), the latter provide reasonable approximations of performance in practice (e.g, as demonstrated by the comparisons of an improved version of Avrora with the SANDbed testbed [12]). By using emulation, we can carry out systematic experiments that cover a broader region of the WSN application design space in an efficient manner, whilst still obtaining sufficiently accurate performance data compared to the use of a testbed.

2.2 Variables

Table 1 enumerates several variables pertinent to WSN environmental monitoring and agricultural applications, based on descriptions of deployments in the literature. For each variable, we enumerate the range of possible values, with reference to deployments or application scenarios described in the literature. In the experiments described in Section 3, each variable may be fixed, or be allowed to vary, depending on the relationships or trade-offs being explored. We therefore specify a default value used for each variable for experiments when it is fixed.

For a particular deployment, the values assigned to these variables result from temporal or spatial characteristics of the phenomenon being measured. For example, in the case of *acquisition interval*, this is likely to depend on the rate of change of the phenomenon being measured. While monitoring a volcanic eruption may require sensor readings to be taken at a very high frequency, in the case of monitoring the movement of ice in a glacier it is likely to be adequate to take readings far less frequently. Resource constraints are another factor likely to influence acquisition interval as, e.g., scarcity of energy may result in measurements being taken less frequently than would be desired, to increase the lifetime of the WSN. As the default for most experiments, we assume that readings are taken every 32s, which allows nodes to sleep for a considerable proportion of the time.

Other variables, such as the *Network Size* and *Node Layout*, depend on the characteristics of the sensing field in question, the points in space for which sensor readings are desired, and the location of the base station. For example, the points of interest, where sensor measurements are needed, will dictate the location of *source nodes* where measurements are taken. If these are not within radio range of each other, *relay nodes* may be needed to bridge radio communications between nodes. As such, the *Network Size* may be influenced by factors including the area of the phenomenon to be observed, the node radio range, the degree of redundancy desired, and available budget. In the experiments, we assume network sizes of 9, 25 and 100, which lend themselves well to a square grid topology. As a default size, we assume a network of 25 nodes, which allows a medium size grid to be defined.

With respect to *Node Layout*, a *linear* topology has few paths to choose from, from the source nodes to the gateway node. As an extreme example, the Crowden Brook deployment [26] has a linear topology in which communications between nodes take place in a pre-determined sequence, partly a consequence of the fact that sensors are placed along the transect of a hill slope. In contrast, with a *grid* topology, there may be several different paths available to route the data. An example of such a deployment occurs in the Sniper Localization deployment application [20], with 60 nodes over a 80m $\times$ 80m $\times$ 6m area, given that acoustic sensor readings need to be taken in 3-dimensional space in order to pinpoint the location of a sniper. In the real world, due to physical obstacles and the nature of natural phenomena, it is often not possible to have nodes laid out as a perfect grid spaced out evenly. We therefore consider a *arbitrary* topology, which arguably more closely reflects conditions in the real world, as the default layout assumed in the experiments.

Node Density considers how close nodes are to each other in space. Factors that may affect node density include the spatial resolution at which sensor readings are required, the radio communication range, the need for redundancy in the network (e.g., so that in the event of node failure, traffic may be routed via another route), as well as budget avail-

Variable	Range of values from real-world deployments	Default
Acquisition Interval Amount of time between sensor readings.	*Almost continuous*, e.g., Volcán Reventador deployment [41] during an eruption, in which a reading is taken every 10 ms; *Moderate*, e.g., Great Duck Island [25, 36] in which readings are taken every 5–60 min; *Very infrequent*, e.g., Glacier monitoring, every 4 hours [28].	32s

Network Size
Number of nodes in the WSN deployment.
	Small network (2–10 nodes), e.g., Crowden Brook [26] comprises only 4 nodes);
	Medium-sized network (11–30 nodes), e.g., the Volcán Reventador deployment [41] has 16 nodes;
	Large network (30+ nodes) e.g., the Great Duck Island deployment [25, 36] had 100+ nodes.
		25

Node Layout
Spatial distribution of nodes throughout the WSN.
	Linear, e.g, Crowden Brook deployment [26];
	Grid, e.g., Sniper Localization deployment application [20];
	Abitrary, e.g., Great Duck Island [25, 36].
		arbitrary

Node Density Measure of how close nodes are to one another in the WSN.	*Sparse* topology, e.g., Crowden Brook [26]; *Dense* topology, e.g., with Sniper Localization deployment [20].	3
Proportion of Sources Percentage of nodes in WSN which have sensors, and can therefore act as data sources.	This information is hard to glean from descriptions of existing WSN deployments. However, it is reasonable to assume that to keep costs down the number of relay nodes will be minimized.	80 %
Radio Packet Loss Rate Percentage of radio packets which are not received successfully by their intended recipient.	Analyses by Szewczyk *et al.* [36] found that measurements in the Great Duck Island network revealed an average 30% packet loss over a single hop.	0 %

Table 1: Variables used in SensorBench.

able. In a *dense* network topology, nodes are relatively close together, so a routing algorithm may have the option of communicating via shorter or longer radio hops. In a *sparse* topology, nodes are relatively far apart, so there is less leeway when choosing a node's neighbours. We introduce the notion of node *density*, defined as the reciprocal of the distance between nodes in terms of the maximum radio range supported by the nodes. For example, the CC100 radio for the Mica2 sensor node platform has a default range R=60 m, so if *density* = 1, every node in the network has at least one neighbour 60 m away from it, and if *density* = 30, every node has at least one neighbour 2 m away from it. In the experiments, we assume a default *density* of 3, as it gives some leeway with routing approaches, but does not provide an overly high degree of redundancy.

The *Proportion of Sources* depends on the number of *relay nodes* required in the WSN to ensure radio connectivity between the nodes of the network. It is assumed that the number of relay nodes in a network will be kept to a minimum, due to cost implications, so 80% is taken as the default value for this variable.

Finally, the *Radio Packet Loss Rate* is a property that depends on the radio propagation characteristics of the sensing field, and the type of radio used. For the experiments, we assume a 0% packet loss as default in order not to distort other factors which may be responsible for a reduction in tuple cardinality, such as load shedding (e.g., as in [31]) or sampling (e.g., as in [17]) which may be carried out by the data processing technique.

2.3 Performance Metrics

In WSN data processing, the resource constraints inherent in the computing fabric lead to stark trade-offs. For example, when monitoring a volcanic eruption, a shorter lifetime of the WSN is acceptable in order to take measurements more frequently and receive results without delay. In other types of applications, maximizing lifetime of the WSN may be the goal, and it may be necessary to minimize radio activity and accept a significant delay in receipt of results. Traditionally, the focus of WSN data processing applications has been to minimize energy consumption. However, WSN applications cover a broad spectrum, and have varied and diverse non-functional requirements [8]. As such, SensorBench supports various performance metrics, and those that will be of interest to a specific user will depend on the characteristics of the application that the data processing technique being evaluated is to be used for. The following performance metrics are considered in SensorBench:

Lifetime is defined as the amount of time it takes for a WSN to be unable to carry out the data processing task due to energy depletion. This variable is influenced by the energy consumption of individual nodes, routing of tuples which may lead to hotspots if certain nodes in the WSN are overloaded, and the robustness of the system in question. Lifetime is measured by recording the time taken for node failure to affect the tuples being delivered.

Total energy consumption is defined as the sum of the energy consumed by all the nodes in the sensor network, over a specific period of time. In contrast to lifetime, which depends on the energy consumption on a per-node basis, total energy consumption increases as the number of nodes in the WSN increases.

```
surface(node_id, time, light, temp, humidity)
burrow(node_id, time, light, temp, humidity)
sensors(node_id, time, light, temp, humidity)
```

Figure 1: Abridged schema for the Great Duck Island Application

Delivery fraction is defined as the percentage of tuples that are delivered to the gateway, of the total that should be delivered to the gateway. For tasks which are lossy in nature (e.g., outlier detection), this is the percentage of the maximum possible number of result tuples. This variable is affected by radio packet loss, as well as load-shedding or sampling policies which are part of the data processing technique.

Delivery delay is defined as the time elapsed between an event occurring in the environment, and the event being reported in the results. This variable is influenced by factors such as the routing strategy used, and whether results are transmitted immediately over the radio, or buffered and sent in batches so that less energy is used with overheads such as turning the radio on. This is measured by recording the average time elapsed between a sensor reading being acquired, and the corresponding result being reported by the system.

Output rate is defined as the amount of data, in bytes per second, produced by the system. Measuring output rate enables observation of how the system behaves under increasing load. This variable depends on several factors including the acquisition interval, the task being run, and the processing capacity of the WSN.

3. BENCHMARK DESIGN

The representative data processing tasks used in SensorBench are described in terms of the Great Duck Island deployment [25, 36], a classical application in which a WSN is used for habitat monitoring, briefly described in Section 3.1. The specification of the tasks is presented in Section 3.2. Finally, in Section 3.3, we set out a series of experiments that involve running these data processing tasks while changing the variables defined in Section 2.2, and measuring the performance metrics described in Section 2.3.

3.1 Case Study

The Great Duck Island [25, 36] WSN deployment has become a well-known example of an environmental monitoring application. It was employed to monitor the habitat of the Leach's Storm Petrel, an endangered seabird that nests in underground burrows. The aim was to understand the nesting patterns of the seabirds in the burrows, and the variations in climatic conditions over time. It was particularly important for the batteries to last 9-12 months, and for the WSN to be as unobtrusive as possible. The deployment comprised two types of nodes: those placed underground, within the burrows that the birds use for nesting, and those placed above ground, to monitor local microclimatic conditions. Figure 1 presents a simplified schema of the application in question, used in the task definitions in the next section. The `sensors` stream is the union of the `surface` and `burrow` streams.

3.2 Tasks

Table 2 presents a collection of representative data processing tasks over a WSN, described in terms of the Great Duck Island application. For each task, we provide a description and, if applicable, a specification using the SNEEql query language [5], an expressive language for querying WSNs based on CQL [3], used for classical data streams. We note that not all the data processing tasks described were actually carried out in the Great Duck Island WSN deployment. Rather, the intention is to cover a set of types of task that occur frequently throughout WSN environmental monitoring applications.

3.3 Workloads

Building on the variables relating to WSNs described, and the performance metrics deemed relevant in this context, we designed the experiments presented in Tables 3 and 4 with the aim of identifying a collection of tests that can be used with different systems to explore their performance when carrying out several representative workloads. The experiments cover a comprehensive design space: each experiment aims to explore the impacts on the five performance metrics of varying one of the six variables at a time, while keeping the other variables fixed, i.e., as usual in analytical experiments, we apply a *ceteris paribus* assumption.

Exp. 1: Impact of varying network size. This experiment aims to observe the scalability of a system. As the network size increases, there are more source nodes, so more data is acquired. Task Select results in the amount of data being transmitted increasing in proportion to the amount of data generated. Therefore, increasing the network size can be expected to lead to increased energy consumption (and hence a shorter lifetime), and also a delay in data that is acquired reaching the gateway node. Tasks Aggr and OD may lead to a reduction in data transmitted resulting from in-network processing. Indeed, in the case of Aggr, increasing the network size should not increase the size of the final result. It can therefore be expected that for these tasks, lifetime and delivery delay should be less adversely affected.

Exp. 2: Impact of varying node layout. This experiment explores the relationship between linearity and the performance metrics. If there are multiple paths available to send tuples to the gateway, it should be possible to spread the workload more evenly and thus achieve a longer lifetime. Having fewer paths may mean that hotspots develop as nodes, particularly those closer to the base station, may become overloaded. If a node is overloaded it may lead to a shorter network lifetime, and congestion leading to a delay in receiving data thus impacting delivery delays.

Exp. 3: Impact of varying node density. In cases where networks are dense, i.e., there are several nodes within the range of each other, systems may be able to choose between having shorter hops, and potentially saving energy, or having longer hops, which may expend more energy but results arrive at their destination faster. Note that in this experiment, only 20 % of the nodes are sources, in order to give more leeway with routing radio traffic.

Exp. 4: Impact of varying acquisition interval. This experiment is essentially a stress test on the system. Acquiring data frequently should lead to rapid energy depletion, whereas doing so less often should allow the hardware to sleep for longer and therefore consume less energy. With respect to output rate, if data is acquired too frequently for a

Task	Description	SNEEql Query
Select	*Get raw sensor readings.* Report raw data readings from the nodes in the WSN.	`RSTREAM SELECT *` `FROM sensors[NOW];`
Aggr	*Get aggregated sensor readings.* Report the average temperature reading for the current time over the nodes in the WSN.	`RSTREAM SELECT AVG(temp)` `FROM sensors[NOW];`
Join	*Correlate data from different regions of the WSN.* Report the `node_id` and `temp` readings of the burrows that have a higher temperature than a surface sensor.	`RSTREAM SELECT b.node_id, b.temp` `FROM burrow[NOW] b, surface[NOW] s` `WHERE b.temp > s.temp;`
Join2	*Correlate present data with past data from different regions of the WSN.* Report the `node_id` and the `temp` readings of the burrows that have a greater temperature than a surface sensor one minute ago.	`RSTREAM SELECT b.node_id, b.temp` `FROM burrow[NOW] b, surface[NOW-1 MINUTE] s` `WHERE b.temp > s.temp;`
LR	*Linear Regression.* Compute the coefficients α and β for a linear function $y = \alpha x + \beta$ that describes the relationship between two sensed variables, where x =`light` and y =`temp`, over the `sensors` stream.	
OD	*Outlier Detection.* Report `temp` readings that deviate from those of other `temp` readings collected at the same time on the `sensors` stream.	

Table 2: Representative data processing tasks used in SensorBench workloads.

Variable	*Exp. 1*	*Exp. 2*	*Exp. 3*	*Exp. 4*
Tasks	{Select, Aggr, LR, OD}	{Select, Aggr, LR, OD}	{Select, Join, LR, OD}	{Select, LR, OD}
Acquisition int. (s)	32	32	32	{1,2,4,8,16,32,64,128}
Network size	{9, 25, 100}	25	25	25
Node Layout	arbitrary	{linear, grid, arbitrary}	arbitrary	arbitrary
Node Density	3	3	{1, 2, 3, 8}	3
Prop. of sources (%)	80	80	20	80
Radio loss rate (%)	0	0	0	0

Table 3: Experiments 1–4: Varying network size, node layout, node density, and acquisition interval.

system to cope, it may return a compile time error (as is the case with SNEE) or perform load shedding of excess tuples (as is the case with TinyDB).

Exp. 5: Impact of increasing proportion of source nodes. This experiment is similar to the previous one in that it is also a stress test on the system. However, in this case the acquisition interval is kept constant, and the workload is increased by increasing the proportion of source nodes in the sensor network.

Exp. 6: Impact of varying radio loss rate. This experiment aims to observe how a system reacts to varying radio propagation conditions. If the loss rate is high, systems that employ techniques such as sending acknowledgements and retransmitting packets are expected to consume more energy, but also achieve a higher output rate than systems such as SNEE that only send data once and do not check whether it has been received. Systems such as SNEE may consume less energy if the radio loss rate is high but are likely to have a significantly worse output rate.

Exp. 7: Impact of varying task. This experiment aims to assess how energy efficient each system is with regard to each type of task. This is an indicator of the degree of in-network processing supported by a particular system for a particular task, e.g., those systems that are able to support in-network aggregations should be able to do the more energy efficiently than those that do not.

4. BENCHMARK IMPLEMENTATION

In Section 4.1, we briefly describe the systems that we used to test the benchmark. In Section 4.2, we describe the experimental context, and the steps followed to implement the benchmark, with reference to the scripts which are publicly available to enable others to do the same. We present the results of our performance evaluation in Section 4.3, and give examples of the lessons that can be learnt by applying the benchmark in Section 4.4.

4.1 System Descriptions

To exemplify the use of SensorBench, we ran it with four systems. As a baseline, we used *MultihopOscilloscope* (MHOSC), an application that comes bundled with TinyOS [21]. It ships all the raw data out of the network and is essentially an example of the warehousing approach. As an example of an in-network query processor, we used *SNEE (for Sensor NEtwork Engine)* [7], developed at the University of Manchester. It employs a rich, expressive query language that supports in-network window-based aggregation and joins[2]. Declarative SNEEql queries are compiled into query execution plans using a query compilation stack based on the ones used in traditional distributed query processors. The query execution plan is translated into nesC [9] source code that, when compiled, generates binaries for execution on WSN nodes. We also evaluate two types of hand-crafted, in-network data analysis algorithms developed at the University of Athens. The *Outlier Detection* algorithm (OD), based on the D3 algorithm [35], constructs a model at each

[2]Note that *delivery delay* is a variable as well as a metric in SNEE, as in SNEE the desired delivery time is a tunable property of the system.

Variable	*Exp. 5*	*Exp. 6*	*Exp. 7*
Tasks	{Select, OD, LR}	{Select, Aggr, LR, OD}	{Select, Aggr, Join, Join2, LR, OD}
Acquisition int. (s)	32	32	{1, 2, 4, 8, 16, 32, 64, 128}
Network size	25	25	25
Node Layout	arbitrary	arbitrary	arbitrary
Node Density	3	3	3
Prop. of sources (%)	{20, 40, 60, 80, 100}	80	80
Radio loss rate (%)	0	{0, 20, 40, 60, 80}	0

Table 4: Experiments 5–7: Varying proportion of sources, radio packet loss rate, and task.

source node and sends tuples that are deemed to be outliers to the gateway. The *Linear Regression* algorithm (LR), based on [39], carries out a partial computation of the model at the source nodes, and results are combined to produce a final result at the gateway.

4.2 Running the Benchmark

We ran the benchmark for the four systems, described in the previous section, for the workloads described in Section 3.3, using the Avrora emulator [38][3]. We chose Avrora to illustrate the application of SensorBench as it emulates instructions with CPU-cycle accuracy, unlike PowerTossim [33] (another popular WSN simulator) which simulates events at a more coarse-grained level. The sensor node hardware we emulated was the MICAz mote (CPU: 8-bit 8 MHz Atmega128L; RAM: 4 kB; Program Memory: 128 kB, Radio: MPR2400) with an energy stock of 31320 J (2 Lithium AA batteries). The executables that ran on WSN nodes were compiled using TinyOS 2.1.1.

We have developed a collection of scripts, available at the SensorBench website[4], to run the benchmark on the four systems mentioned. These scripts provide various types of functionality, including preparation of jobs to be run on the Avrora emulator, running workloads (optionally using the HTCondor [37] parallel computing platform if available), and parsing the output from the simulations to obtain the performance metrics of interest. We note that these scripts are intended to be extensible, so as to enable further systems to be evaluated, or a different simulator to be used, with minimal adaptations.

We generated 10 network topologies for each combination of the ⟨*Network Size, Node Layout, Node Density, Proportion of Sources*⟩ parameters in Tables 3 and 4 using a script which generates network topologies. For arbitrary topologies, the nodes are placed randomly across the sensing field. By taking the average over 10 instances for each scenario, results are less likely to be distorted by individual instances which may exhibit atypical characteristics. To ensure a fair comparison, these topologies should be used in future runs of the benchmark. For all tasks except Outlier Detection, we use the Avrora random value generator to produce the input sensor readings. For Outlier Detection, we provide synthesized datafiles with approximately 10% outliers[5].

We did not run each experiment for the same amount of time, because over a fixed time interval, the amount of data collected (and hence, the amount of work done by the data processing task) varies depending on the acquisition interval. Rather, each experiment was run for ten complete *data collection cycles*, thereby ensuring that the results were comparable. The duration of a data collection cycle is equal to the product of the acquisition interval and buffering factor β (defined as the number of tuples that are buffered before they are transmitted over the radio). For MHOSC, β is fixed at 5; for SNEE, β can vary depending on the parameters set by the user, but in these experiments was set to 1 to minimize delivery delay; for LR and OD, $\beta = 1$.

We measured energy by parsing the output of the Avrora energy monitor, which gives per-node and per-component breakdowns of energy consumption by emulation at CPU cycle accuracy. As none of the techniques evaluated are adaptive, it is reasonable to assume that the energy consumption would remain fairly constant throughout the deployment duration, and the total network energy values were scaled for a period of 6 months. The lifetime calculation was based on the same assumption. Firstly, the energy consumption per unit time per node was computed. Based on the energy resources of the node, a per-node lifetime was calculated based on the current energy consumption. The lifetime of the WSN was assumed to be equal to that of the shortest node lifetime in the WSN. The output rate, delivery fraction and delivery delay were measured by parsing the Avrora log files to obtain the times that tuples are acquired at the source nodes, and delivered to the gateway.

4.3 Performance Comparison

Figure 2 presents a selection of the results to illustrate how the benchmark can be used to derive the metrics introduced in Section 2.3 and to evaluate and compare different strategies for implementing the data processing tasks. Each point in the graphs corresponds to the average measurement of ten runs using different topologies for each scenario.

Limitations of the Evaluation. Due to inherent characteristics of the platforms being evaluated, and also of the simulator used, it was not always possible to carry out the experiments according to the specifications given in Section 3.3. For *Exp. 1*, SNEE had to be run using an acquisition interval of 128s, as it does not support an acquisition interval of 32s for 100-node networks. This is due to the query compiler using time cost estimation models to determine that it would not have enough time to deliver all the tuples for a shorter acquisition interval. Throughout the experiments, only SNEE could be configured to vary the proportion of source nodes. The other platforms therefore have this variable fixed at 100%. Plots for *Exp. 5* are not shown as this experiment only provides a full set of results for SNEE. Finally, *Exp. 6* could not be executed using Avrora as it does not support setting the radio packet loss as a parameter.

[3]We used Avrora version 1.7.113 with modifications to the sensor data parsing component.

[4]`https://code.google.com/p/sensorbench/`

[5]Topology and sensor datafiles are available at: `http://dx.doi.org/10.6084/m9.figshare.934307`

Exp. 1 investigated the impact of network size on the performance metrics. Figure 2a reports the percentage of tuples delivered for the SNEE Select and Aggr tasks, MHOSC, OD and LR. As network size increases, the percentage of tuples delivered by SNEE remains constant at 100, which is a consequence of its approach of generating a pre-determined routing tree and agenda, which essentially ensures (through a strict, time-slotted approach) that there is enough time for every sensing, processing and radio communication activity to take place. On the other hand, the MHOSC percentage of tuples delivered is approximately 90%, for smaller network sizes, and decreases to 80% for large networks. This is a consequence of the MHOSC approach of generating a routing tree dynamically at run-time and using an underlying protocol that aims to provide best-effort tuple delivery, relying on acknowledgements to confirm that each packet has been received, and retransmitting a certain number of times, if necessary. OD has a low tuple delivery fraction as only a small proportion of tuples are identified as outliers. This is consistent with the behaviour one would expect the outlier detection task to exhibit. However, as network size increases, the delivery fraction decreases, as a result of radio packet collisions, leading to tuple loss, despite the use of radio packet acknowledgements to mitigate this.

Delivery delay plots for the same runs are shown in Figure 2b. MHOSC takes considerably longer to deliver data to the gateway node. This is a consequence of having a fixed buffering factor of 5, i.e., it collects 5 sensor readings before transmitting them to the WSN gateway. Conversely, when SNEE optimizes for delivery delay, it sets its buffering factor to 1, and delivers data in a considerably shorter time, at least for the network sizes shown on the graph. OD and LR both transmit data up the routing tree as soon as it is sensed. They do not have a time-slotted approach like SNEE, meaning that the delivery delay is shorter, although packets are lost due to collisions. Unlike MHOSC, OD and LR, whose delivery delay increases slightly as network size increases, SNEE's delivery delay increases significantly. This is because the SNEE compiler allocates more time for its strict time-slotted approach, as increasing network size means that more data needs to be sensed, processed and transmitted through the WSN. Therefore, as network size increases, the SNEE time-slotted approach is less scalable, in terms of delivery delay, than asynchronous approaches in which radio collisions may occur.

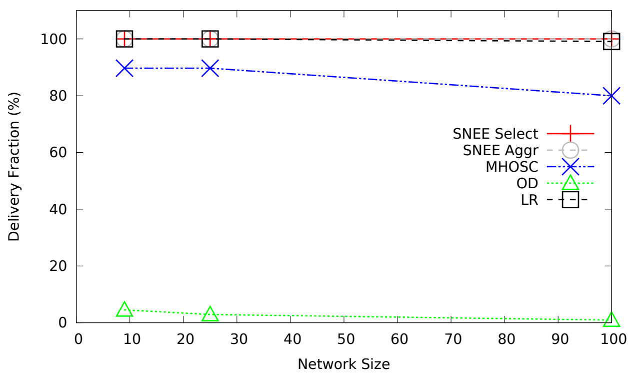
Exp. 2 investigates the effect of node layout type. The results for lifetime are reported in Figure 2c. Overall, it can be seen that SNEE leads to considerably longer lifetimes than either MHOSC, LR or OD. This is because SNEE puts the CPU into a low-power processing mode, whereas MHOSC, OD and LR do not, meaning that the CPU remains in idle mode during periods of inactivity. Although MHOSC and LR transmit considerably more data than OD, their lifetimes are virtually the same, as the CPU idle costs dominate in this case. For MHOSC and OD, there is also no visible difference in lifetime for the different node layouts. In contrast, SNEE does lead to significant differences in lifetime for the different network layouts and tasks. Running an Aggr task over a linear network yields an average 15.4% improvement in lifetime over the grid and arbitrary topologies. This is because the incremental aggregation approach used by SNEE ensures that at most one tuple is transmitted and received by each node in a linear topology, which is free of hotspots (i.e., no node in the routing tree receives tuples from multiple child nodes). The arbitrary topology has the shortest lifetime for both SNEE tasks, as unlike the linear and grid topologies, which consist of nodes spaced at regular intervals, having the nodes distributed in a arbitrary fashion can lead to the emergence of more severe hotspots in the routing tree.

Figure 2d contrasts the delivery delay obtained by varying node layouts for the OD and LR techniques. Overall, LR has significantly longer delivery delay than OD. This is due to two factors: Firstly, LR transmits considerably more data than OD. It carries out aggregations at each source node, which result in several values being sent to the gateway. In contrast, OD only sends values which are deemed to be outliers when these are detected. Secondly, LR coordinates radio transmissions by using control messages between the nodes. This results in transmissions being scheduled so that they take place in the same order as a depth-first traversal of the routing tree. This means that the delivery delay increases with the number of edges in the routing tree. Furthermore, radio collisions are avoided, and a high tuple delivery fraction is achieved (as is evidenced in Figure 2a). In contrast, OD transmits outlier readings to the gateway as soon as these are detected. This leads to a considerably shorter delivery delay. Acknowledgements are used to resend packets in event of radio collisions. As a consequence of the shape of the routing tree (and in particular, the number of edges it has), a linear topology leads to a longer delivery delay for both OD and LR. A grid topology leads to the shortest delivery times in both cases.

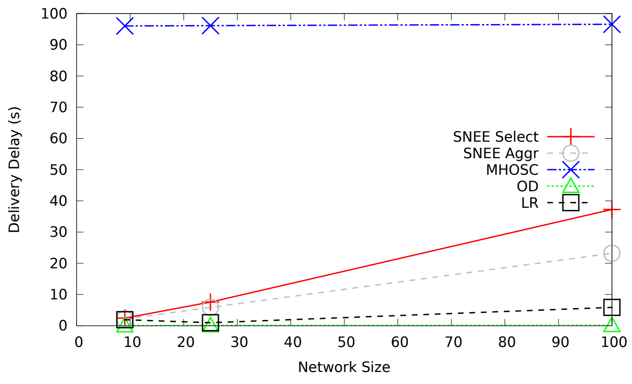
Exp. 3 investigates the effect of varying the node density for a fixed network size of 25 nodes and a fixed acquisition interval of 32s. Results for the output rate metric are reported in Figure 2e. For SNEE, the tuple rate is constant, as its time-slotted approach ensures that there is sufficient time to deliver all tuples to the gateway. In contrast, the MHOSC output rate decreases by 18.2% as the network density increases from 3 to 8. This can be explained by the fact that having the nodes closer together leads to more radio collisions and a higher rate of packet loss. The LR and OD output rate is not affected by varying node density. LR produces a single result tuple for each acquisition time, which is the result of several aggregations. If some of the input tuples for the aggregations are lost, the cardinality of the result is not affected. With the OD task, packets are retransmitted using acknowledgements, thus mininizing tuple loss.

Exp. 4 explores the relationship between acquisition interval and the performance metrics measured. The results for delivery delay are reported in Figure 2f. As was apparent in Figure 2b, MHOSC takes considerably longer to send results than SNEE as it buffers 5 data readings before transmitting them to the gateway. As the acquisition rate increases, the time taken to deliver the data increases linearly. In contrast, SNEE, OD and LR send result tuples immediately, and therefore the acquisition interval does not affect the delivery delay. The missing points reveal limitations of the platforms: SNEE does not support an acquisition interval smaller than 16s for a 25-node network, and MHOSC does not support an acquisition interval greater than 32s regardless of network size.

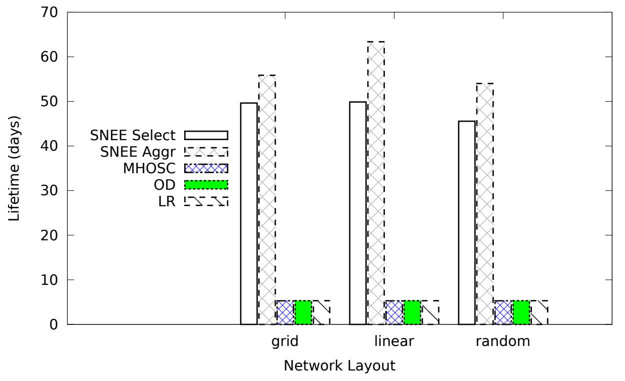
Figure 2g shows the results for total network energy used from *Exp. 4*. The stark difference between SNEE and the other two systems is because SNEE moves to a low-power



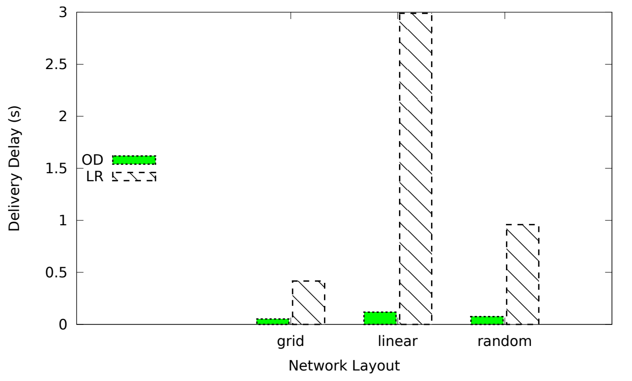
(a) Exp. 1: Network size vs. Tuples Delivered (%)



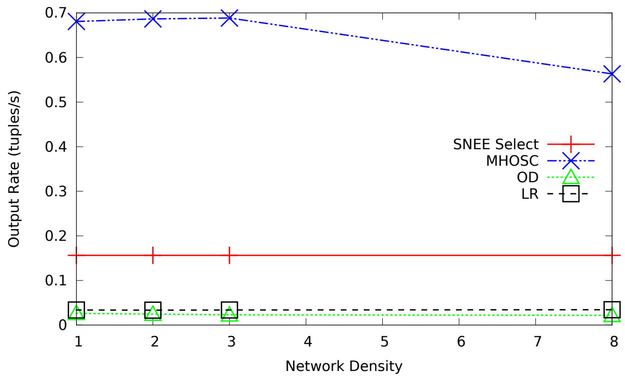
(b) Exp. 1: Network size vs. Delivery Delay (%)



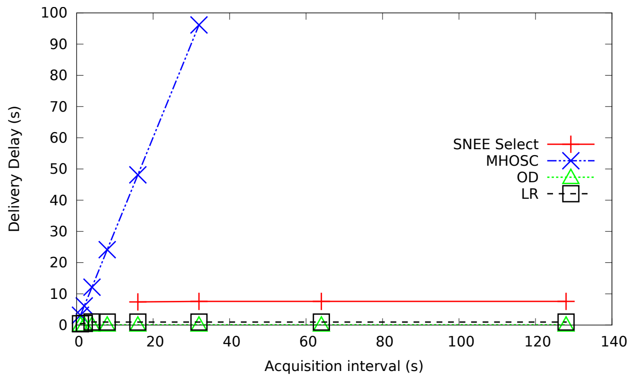
(c) Exp. 2: Node layout vs. Lifetime (days)



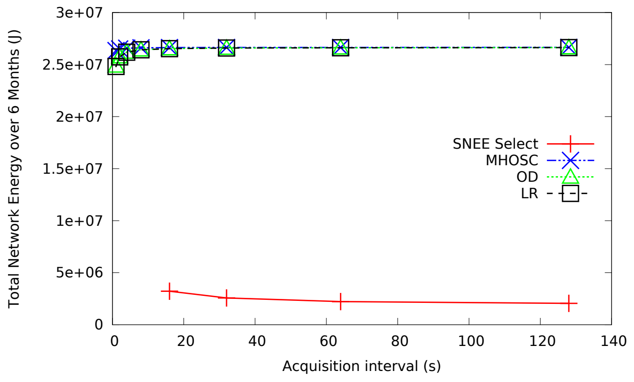
(d) Exp. 2: Node layout vs. Delivery Delay for OD and LR.



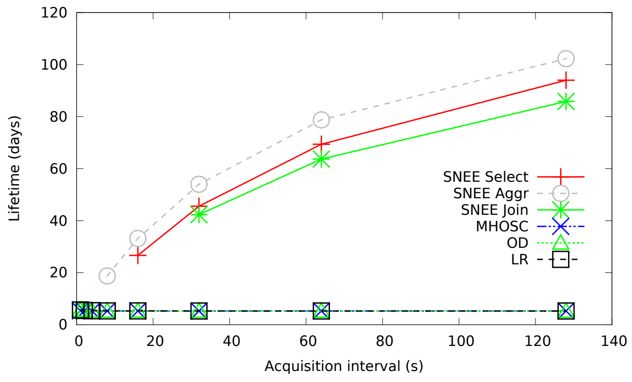
(e) Exp. 3: Node density vs. Output Rate (tuples/s)



(f) Exp. 4: Acquisition Interval vs. Delivery delay (s)



(g) Exp. 4: Acquisition Interval vs. Total 6-month Energy (J)



(h) Exp. 7: Acquisition Interval vs. Lifetime (days)

Figure 2: Experiment Results

mode during periods of inactivity, and the effect of this approach is also visible in Figures 2c and 2h. The total network energy only decreases slightly for SNEE, as it is dominated by the number of nodes in the network.

Exp. 7 varies the acquisition interval for different tasks and platforms. Results are reported in Figure 2h. For MHOSC and OD, the results are consistent with those previously reported. The SNEE Aggr task achieves the highest lifetime, as it causes the least number of tuples to be transmitted. The SNEE Join has a shorter lifetime than the SNEE Select as the join is executed in-network, and has a higher selectivity than that anticipated at compile time.

4.4 Lessons Learnt

Running the benchmark has confirmed the following with respect to the different approaches employed by the systems:

- In-network processing is generally beneficial.
- Systems which do not go into a low power state during periods of inactivity have very short lifetimes, compared to those that do. The extent to which the idle CPU activity dominates the energy cost, which makes the difference between the lifetimes of MHOSC, OD and LR negligible, is perhaps more surprising.
- Adopting a strict time-slotted approach for carrying out activities such as radio transmissions can ensure that all tuples are delivered (in the absence of noise). However, as workload increases (e.g., with a larger network size), such an approach can lead to a significant increase in delivery delay. Asynchronous approaches tend to result in shorter delivery delays (but have a lower percentage of tuples delivered).
- Generally speaking, grid topologies are amenable to routing trees in which workloads are more evenly distributed throughout the nodes in the WSN, meaning less hotspots in the network and longer lifetimes. Unless an aggregation task is being performed, linear topologies result in the most severe hotspots, resulting in the shortest lifetimes and longest delivery delays.

5. RELATED WORK

This section discusses work that is related to SensorBench, focusing on *benchmark properties*, *benchmarks in related areas* and *performance evaluations of data processing techniques in WSNs*.

In terms of *benchmark properties*, The Benchmark Handbook identifies *relevance*, *portability*, *scalability* and *simplicity* as key criteria for a domain-specific benchmark [10]. A benchmark is considered to be *relevant* if it measures peak performance and price/performance when performing typical operations. In SensorBench, we explore peak performance in terms of the delivery delays and lifetimes, and provide a range of operations that can be found in common applications. A benchmark is considered to be *portable* if it is easy to implement on different systems. In SensorBench, we have developed scripts to support benchmark execution on the widely used Avrora emulator, and have shown the benchmark in use with several different and independently developed systems. A benchmark is considered to be *scalable* if it applies to small and large systems. In SensorBench, the benchmark is agnostic to the specific properties of the individual nodes, and experiments include network size as a variable. A benchmark is considered to be *simple* if it is understandable. In SensorBench we have used a collection of common tasks, most of which are easily expressed by a stream query language, and we explore variables and report properties that have featured in previous performance evaluations, as discussed below.

In terms of *benchmarks in related areas*, relevant topics include *stream data management* and *WSNs*. In relational stream data management, the most prominent benchmark is Linear Road [4], which defines data analysis tasks and data generators representing the monitoring of traffic on a toll road. SensorBench also defines a scenario, data generators and analyses, but is complementary in exploring different variables and reporting different result types, reflecting the focus on WSNs. Several other benchmarks have focused on WSNs. These benchmarks complement SensorBench, by evaluating different features, such as devices [13], processors [30], cryptographic algorithms [19] and communications [40]. In turn, sensor network performance analyses are often supported by simulators, such as PowerTOSSIM [33] or Avrora [38]. Such simulators complement and support benchmarks for sensor data management, by enabling controlled experiments to be carried out conveniently and at manageable cost.

In terms of *performance evaluations of data processing techniques in WSNs*, we relate the scope and nature of representative experimental evaluations from the literature with the properties and tasks described in Sections 2 and 3. Table 5 applies the terminology used in SensorBench to summarize differences in coverage between SensorBench and other experimental evaluations. The purpose of the table is to identify where features from SensorBench occur in other evaluations, and to identify where SensorBench omits features that occur elsewhere, so that design decisions on scope can be explored systematically.

In terms of *Variables*, SensorBench covers quite a broad range. This is because it seems appropriate for a benchmark to capture many different features of the environment in which a deployment may occur, even if quite a few of these features are constants in any one deployment. One interesting feature from several experiments is the reporting of how energy consumption (for example) varies over *time*. This feature is not included in SensorBench, because between them the variables and metrics tend to summarize behaviour, rather than drill down to capture rapidly changing features.

In terms of *Performance Metrics*, SensorBench uses a relatively broad range of metrics. This is perhaps not surprising, as some of the individual evaluations focus on particular aspects of a technique that may be studied satisfactorily using only a few metrics. We note that different metrics may be strongly correlated; for example, the *Node Load* in [29] is related to *Network Lifetime*, as the most highly loaded node is likely to determine the useable lifetime of the network. Interestingly, in TinyDB, maintenance overheads are investigated but they are not part of SensorBench because identifying and isolating overheads in consistent ways across multiple benchmarked systems seems likely to violate the principle of *simplicity* in benchmark design.

In terms of *Task Types*, SensorBench tends to subsume those of other proposals, although some proposals take finer grained control of query parameters, such as selectivity. Here

Proposal	Variables	Performance Metrics	Task Types
SensorBench	Acquisition Rate, Node Layout, Node Density, Network Size, Proportion of Sources, Packet Loss Rate	Network Lifetime, Energy Consumption, Delivery Fraction, Delivery Delay, Output Rate,	Select, Aggregate, Join, Regression Outlier Detection
TinyDB [24]	Acquisition Rate, Selectivity, Time	Energy Consumption, Delivery Fraction, Output Rate, Maintenance Overhead	Select, Aggregate
AnduIN [18]	Time, Window Size	Energy Consumption, Computation Time	Select, Aggregate, Join, Outlier Detection
MicroPulse [2]	Node Density, Time	Energy Consumption	Select
SNEE [7]	Acquisition Rate, Delivery Time, Node Layout	Network Lifetime, Energy Consumption, Memory Usage	Select, Aggregate, Join
Aspen [29]	Selectivity, Window Size Time, Node Layout	Network Traffic, Node Load	Join
Bisque [23]	Network Size, Selectivity	Node Energy Consumption, Delivery Delay, Delivery Fraction	Select, Aggregate

Table 5: High level features of evaluations of sensor data management systems.

again there is a trade-off between having large numbers of tasks and keeping the benchmark simple. We have chosen to have a significant number of task types and less variety within these task types, on the basis that the task types are most likely to provide higher-level insights.

The closest piece of work to SensorBench is Bisque [23], a benchmark for sensor query processing, the technical report of which applies the benchmark to compare several TinyDB variants. Bisque is a useful contribution, but SensorBench covers more ground in terms of: (i) the variables explored, since, for example we consider more varied and realistic networks than the uniform grids supported by Bisque; (ii) the performance metrics, since we consider more aspects of energy usage; and (iii) query type, since our scope includes joins and several analysis tasks. We also make SensorBench available for use with the widely used Avrora [38] emulator. Overall, SensorBench has been designed to reflect the developments in sensor network data management that have taken place since Bisque was proposed, by providing a richer context for experimentation.

6. CONCLUSIONS

We have described SensorBench, a benchmark that supports the analysis of the performance of data intensive tasks over WSNs. The emphasis is on tasks, variables and measurements that are relevant to environmental monitoring applications, and the properties of the benchmark have been validated both: (i) in relation to published descriptions of WSN deployments; and (ii) in relation to existing evaluations of data intensive systems for WSNs. The benchmark subsumes most relevant existing empirical analyses in terms of its scope, while including modest numbers of distinct tasks, with a view to ensuring that the benchmark is practical to run. To further support its practicality, scripts have been developed to ease its application using a popular simulator.

The benchmark has been applied to several different systems and algorithms that support access to and/or analysis of data from WSNs. This application has given some clear lessons. For example, there is little point in developing intelligent distributed algorithms that minimize network traffic if these algorithms are not deployed in ways that enable sensor nodes to maximize the time spent in power-saving modes. Furthermore, the application of the benchmark has cast light on the complex relationship between approaches to data transmission that seek to avoid clashes and those that seek to recover from them. As such, the application of the benchmark for comparing a variety of different applications and platforms has demonstrated its suitability for casting light on design decisions and trade-offs that can make the difference between successful and unsuccessful WSN deployments.

7. REFERENCES

[1] A. Alfadhly, U. Baroudi, and M. Younis. An effective approach for tolerating simultaneous failures in wireless sensor and actor networks. In *MiSeNet*, pages 21–26, New York, NY, USA, 2012. ACM.

[2] P. Andreou, D. Zeinalipour-Yazti, A. Pamboris, P. K. Chrysanthis, and G. Samaras. Optimized query routing trees for wireless sensor networks. *Inf. Syst.*, 36(2):267–291, 2011.

[3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.

[4] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. In *VLDB*, pages 480–491, 2004.

[5] C. Y. A. Brenninkmeijer, I. Galpin, A. A. A. Fernandes, and N. W. Paton. A semantics for a query language over sensors, streams and relations. In *BNCOD*, pages 87–99, 2008.

[6] P. Corke, T. Wark, R. Jurdak, W. Hu, P. Valencia, and D. Moore. Environmental wireless sensor networks. *Proc. of the IEEE*, 98(11):1903–1917, 2010.

[7] I. Galpin, C. Y. A. Brenninkmeijer, A. J. G. Gray, F. Jabeen, A. A. A. Fernandes, and N. W. Paton. SNEE: a query processor for wireless sensor networks. *Distrib. Parallel Dat.*, 29(1-2):31–85, Nov. 2011.

[8] I. Galpin, A. A. A. Fernandes, and N. W. Paton. QoS-aware optimization of sensor network queries. *VLDB J.*, 22(4):495–517, 2013.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *SIGPLAN Not.*, 38(5):1–11, May 2003.

[10] J. Gray. Database and transaction processing performance handbook. In *The Benchmark Handbook.* Morgan Kaurmann, 1993.

[11] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN*, pages 1–10. IEEE, 2004.

[12] C. Haas, J. Wilke, and V. Stöhr. Realistic simulation of energy consumption in wireless sensor networks. In *EWSN*, pages 82–97, 2012.

[13] M. Hempstead, M. Welsh, and D. Brooks. TinyBench: The case for a standardized benchmark suite for TinyOS based wireless sensor network devices. In *LCN*, pages 585–586, 2004.

[14] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. The platforms enabling wireless sensor networks. *Commun. ACM*, 47(6):41–46, 2004.

[15] J. Hoebeke, I. Moerman, B. Dhoedt, and P. Demeester. An overview of mobile ad hoc networks: Applications and challenges. In *European Telecommun. Congr.*, pages 60–66, November 2004.

[16] P. Jackman, A. J. G. Gray, A. Brass, R. Stevens, M. Shi, D. Scuffell, S. Hammersley, and B. Grieve. Processing Online Crop Disease Warning Information via Sensor Networks using ISA Ontologies. *Agriculture Engineering International: CIGR Journal*, 15(3):243–251, 2013.

[17] A. Jain and E. Y. Chang. Adaptive sampling for sensor networks. In *DMSN*, pages 10–16, New York, NY, USA, 2004. ACM.

[18] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K.-U. Sattler. Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in AnduIN. *Distrib. Parallel Data.*, 29(1-2), 2011.

[19] Y. W. Law, J. Doumen, and P. Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *TOSN*, 2(1):65–93, 2006.

[20] Á. Lédeczi, A. Nádas, P. Völgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dóra, K. Molnár, M. Maróti, and G. Simon. Countersniper system for urban warfare. *TOSN*, 1(2):153–177, 2005.

[21] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005.

[22] M. Liu, S. R. Mihaylov, Z. Bao, M. Jacob, Z. G. Ives, B. T. Loo, and S. Guha. SmartCIS: integrating digital and physical environments. *SIGMOD Rec.*, 39(1):48–53, Sept. 2010.

[23] Q. Luo, H. Wu, W. Xue, and B. He. Benchmarking in-network sensor query processing. Technical Report HKUST-CS05-09, Department of Computer Science, HKUST, 2005.

[24] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[25] A. M. Mainwaring, D. E. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA*, pages 88–97, 2002.

[26] I. W. Marshall, M. C. Price, H. Li, N. Boyd, and S. Boult. Multi-sensor cross correlation for alarm generation in a deployed sensor network. In *EuroSSC*, pages 286–299, 2007.

[27] K. Martinez, J. K. Hart, and R. Ong. Environmental sensor networks. *Computer*, 37(8):50–56, 2004.

[28] K. Martinez, R. Ong, and J. Hart. Glacsweb: a sensor network for hostile environments. *IEEE Sensor and Ad Hoc Communications and Networks*, 2004.

[29] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha. Dynamic join optimization in multi-hop wireless sensor networks. *PVLDB*, 3(1):1279–1290, 2010.

[30] L. Nazhandali, M. Minuth, and T. Austin. SenseBench: toward an accurate evaluation of sensor network processors. In *Proc. Int. Workload Characterization Symp.*, pages 197–203. IEEE, 2005.

[31] L. Peng and K. S. Candan. Data-quality guided load shedding for expensive in-network data processing. In *ICDE*, pages 1325–1328, 2007.

[32] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. What does model-driven data acquisition really achieve in wireless sensor networks? In *PerCom*, pages 85–94, 2012.

[33] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys*, pages 188–200, 2004.

[34] A. B. Stokes, A. A. A. Fernandes, and N. W. Paton. Resilient sensor network query processing using logical overlays. In *MobiDE*, pages 45–52, 2012.

[35] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198, 2006.

[36] R. Szewczyk, A. M. Mainwaring, J. Polastre, J. Anderson, and D. E. Culler. An analysis of a large scale habitat monitoring application. In *SenSys*, pages 214–226, 2004.

[37] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[38] B. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN*, pages 477–482, 2005.

[39] G. Valkanas, A. Kotsifakos, D. Gunopulos, I. Galpin, A. J. G. Gray, A. A. A. Fernandes, and N. W. Paton. Deploying in-network data analysis techniques in sensor networks. In *MDM*, pages 341–344, 2011.

[40] K. Veress and M. Maroti. Linkbench: Benchmark and metric framework for wireless sensor networks. In *IPSN*, pages 171–172, 2011.

[41] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *OSDI*, pages 381–396, 2006.

[42] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *IPSN*, page 68. IEEE Press, 2005.