Optimizing Sensor Data Acquisition for Energy-Efficient Smartphone-based Continuous Event Processing

Archan Misra*, Lipyeow Lim†
* School of Information Systems, Singapore Management University
† Information and Computer Sciences Department, University of Hawai'i at Mānoa
E-mail: archanm@smu.edu.sg, lipyeow@hawaii.edu

***Abstract*—Many pervasive applications, such as activity recognition or remote wellness monitoring, utilize a personal mobile device (aka smartphone) to perform continuous processing of data streams acquired from locally-connected, wearable, sensors. To ensure the continuous operation of such applications on a battery-limited mobile device, it is essential to dramatically reduce the energy overhead associated with the process of sensor data acquisition and processing. To achieve this goal, this paper introduces a technique of 'acquisition-cost' aware continuous query processing, as part of the Acquisition Cost-Aware Query Adaptation (ACQUA) framework. ACQUA replaces the current paradigm, where the data is typically streamed (pushed) from the sensors to the smartphone, with a pull-based asynchronous model, where the phone retrieves appropriate blocks of sensor data from individual sensors, only when the stream elements are judged to be relevant to the query being processed. We describe algorithms that dynamically optimize the sequence (for complex stream queries with conjunctive and disjunctive predicates) in which such sensor data streams are retrieved by the phone, based on a combination of the communication cost and selectivity properties of individual sensor streams. Simulation experiments indicate that this approach can result in 70% reduction in the energy overhead of continuous query processing, without affecting the fidelity of the processing logic.**

I. Introduction

An increasing variety of pervasive applications are being scripted around the use of a mobile computing device (typically called the 'smartphone') as a personalized gateway for aggregating and processing multiple streams of sensor-generated data. While smartphones already have several on-board sensors (e.g., GPS, accelerometer, compass and microphone), there are many situations where the smartphone aggregates data from a variety of other specific *external* medical (e.g., ECG, EMG, Sp02) or environmental (e.g., temperature, pollution) sensors, using a Personal Area Network (PAN) technology, such as Bluetooth™, IEEE 802.15.4 or even WiFi (IEEE 802.11). This computing paradigm is broadly referred to as a "3-tier" monitoring architecture, with the smartphone acting as a pervasive, mobile gateway that makes the information captured by the PAN-connected sensors available to the backend "cloud" logic. A major role of the smartphone in this paradigm is to perform embedded event processing on the sensor data streams to extract appropriate *individual context* in near-real time, for use in a variety of applications, ranging from automatic activity updates for social networking applications (e.g., [2]) to dynamic adaptation of reporting thresholds for adaptive remote health monitoring (e.g., [3]).

Unfortunately, the energy requirements on the smartphone (as well as the sensors) continue to bedevil the *continuous* operation of such context-extraction logic — it has been well documented that the continuous processing of even moderate-data rate streams (such as Sp02 or GPS) can cause commercial smartphone batteries to be depleted in as low as 4-5 hours (e.g., see [1]). Our work thus explores an approach to reduce the energy footprint of such continuous context-extraction activities, primarily by **reducing the volume of sensor data that is transmitted wirelessly over the PAN interface between a smartphone and its attached sensors, without compromising the fidelity of the event processing logic.** More specifically, we aim to replace the "push" model of sensor data transmission, where the sensors simply continuously transmit their samples to the smartphone, with a "phone-controlled dynamic pull" model, where the smartphone *selectively* pulls only appropriate subsets of the sensor data streams. This new model exploits the intelligent programming and data filtering capabilities becoming commonplace on many emerging wearable sensor platforms (e.g., the SHIMMER platform [4]), whose data storage and transmission behavior can be programmed 'over the air'–for example, we can not only implement a 'step counter' algorithm over the accelerometer data on a SHIMMER device, but also dynamically adjust the algorithm's 'amplitude threshold' by sending instructions over the Bluetooth or 802.15 radios.

We believe that significant improvements in the energy efficiency of continuous pervasive "context sensing" can be achieved by not only intelligently pushing data filtering logic to such programmable sensor platforms, but also by dramatically reducing the quantity of sensor data that is actually transmitted over the sensor-phone wireless PAN interface. In particular, we observe that the detection of a specific activity context on the smartphone typically involves the processing of a *complex query, consisting of multiple predicates and involving multiple independent sensors.* Accordingly, we focus on a 'dynamic pull' model, where the event processing engine on the smartphone dynamically modifies both the *order* and the *segments of*

data streams that it will request from each individual sensor. The heart of this paper describes such dynamic pull-based algorithms for a generic query model, where the complex query consists of a combination of disjunctive and conjunctive predicates over 'tumbling window'-based stream operators.

The innovations addressed in this paper are twofold:

- We develop the algorithms to implement an 'acquisition cost' aware event processing paradigm, where the event engine on the smartphone dynamically optimizes the order in which it retrieves data streams from individual phones. The algorithms ensure that, ideally, only a small fraction of the sensor data tuples, judged to be relevant to (partially executed) stream queries, are actually transferred from the sensors to the smartphone. We specifically describe algorithms to optimize the energy overheads of such data processing, taking into account both the wireless transfer cost and the query selectivity properties.
- We explicitly account for and exploit the significant transmission energy savings that result from the intermittent, scheduled use of the PAN link between individual sensors and the phone to transfer the sensor data, as opposed to the continuous transmission of generated sensor data streams. Qualitatively speaking, transmitting the sensor data in batches or 'bursts' allows the wireless radios to operate on a low duty cycle, and better amortizes the overheads of wireless packet transmission. The batched mode of data retrieval, however, introduces an interesting trade-off between the likely relevance of the sensor data and the energy cost of transmitting this data.

To support these innovations, we introduce a new continuous stream processing model called *ACQUA (Acquisition Cost-Aware Query Adaptation)*, which first learns the selectivity properties of different sensor streams and then utilizes such estimated selectivity values to modify the sequence in which the smartphone acquires data from the sensors. The principal focus of this paper is on the second component: namely, developing the algorithms for intelligently modifying the acquisition sequence for supporting a generic family of complex stream processing queries. Our goal is to effectively establish the magnitude of the energy savings that are likely to result from this algorithmic approach, and thereby encourage the adoption of the ACQUA framework in future smartphone systems.

The rest of the paper is organized as follows. Section II provides a brief survey of the related and prior work and establishes the documented trade-off between transmission energy efficiency and data 'batch' size for two representative PAN radio technologies. Section III captures the key objectives and issues that the ACQUA framework must consider and describes the component-level functional architecture of ACQUA. Section IV provides a formal enumeration of the event query model and the operator set that we consider. Section V details the ACQUA sequence-computation algorithms and event processing logic, while Section VI presents simulation-based studies to evaluate the expected performance benefits. Finally, Section VII concludes the paper with a discussion of open issues that we are working to address.

II. Related and Prior Work

The use of complex event processing of sensor data streams on a smartphone for detecting context on a smartphone has been previously explored in system prototypes such as Harmoni [3] (which used such context to dynamically change the definition of anomalous medical states) and CenceMe [2] (which applied rich operators on audio and accelerometer sensor streams to identify pre-defined human activities). To further reduce the energy overheads, the MediAlly prototype [8] used such inferred context to dynamically activate the collection of data from other external sensors. In contrast, our ACQUA framework seeks to optimize the data transfer during the process of context determination itself.

The eventual solution for continuous query extraction on smartphone is likely to employ a combination of software and hardware innovations. Recently, the *LittleRock* [5] prototype has demonstrated how the use of a special low-energy coprocessor can result in a two order-of-magnitude decrease in the *computational energy* spent in embedded processing of on-board sensor data streams. Our ACQUA framework can be viewed as complementary to such hardware or system-level innovations, as we seek to additionally reduce the *communication energy overheads* involved in acquiring the data wirelessly from additional external sensors. To address the challenges of energy-efficient *continuous* event processing on smartphones, the *Jigsaw* continuous sensing engine [6] has recently developed a pipelined stream processing architecture that adaptively triggers different sensors at different sampling rates to meet the context accuracy required by different applications. Our ACQUA framework is also complementary to Jigsaw, in that it focuses on optimizing the retrieval of stream segments from PAN-connected external sensors, rather than on optimizing the sensing fidelity of on-board sensors. However, while Jigsaw considers only *extrinsic* sensor properties (such as its sampling rate), ACQUA uses both the retrieved *values* of the sensor data tuples, and the intermediate query evaluation results, to alter the data retrieval and processing pipeline.

The BBQ [11] approach is among the closest to ACQUA, in its use of a model of the selectivity characteristics of each sensor source to optimize the data acquisition overhead. In particular, BBQ builds a multi-dimensional Gaussian probability density function of the sensors' likely data values, and then uses conditional probabilities to determine, in iterative fashion, the next sensor whose value is most likely to resolve a given query. ACQUA differs from [11] principally in its focus on continuous stream queries (as opposed to snapshot queries) and in the explicit consideration of the impact of batched acquisition of individual streams both on the wireless acquisition cost and the selectivity properties.

Relationship between Batching and Transmission Energy. A key element of our contribution is the explicit capture of the effects of batched data acquisition on the communication energy overheads associated with real wireless PAN technologies. To study this phenomena in detail, we utilize prior work that accurately captures the key characteristics of two specific

wireless technologies–WiFi (IEEE 802.11) and Bluetooth. Although the transmission power and energy associated with data transfers, as well as the link bandwidth, will be unique for each specific technology, we believe that these two widely-used radio technologies represent two broad classes of PAN wireless technologies. In particular, IEEE 802.11 represents a high-power, high-data rate PAN technology, while Bluetooth represents a low-power, low-data rate alternative.

As we will shortly see, both of these technologies have two distinct modes–a low-power 'idle' mode (where the radio lies dormant and consumes significantly lower power) and an 'active' mode (where the radio is actually capable of engaging in packet transmission or reception activity). In general, let P_a be the power consumption in active mode, and P_i ($P_i \ll P_a$) be the power consumed in the 'idle' mode. Also, let B be the transmission bandwidth (bps) of the radio link, when active. We consider the case of a generic sensor, operating under a sampling frequency of f Hz with a sample size of S bits (resulting in a data generation rate, R, given by $R = f * S$). We consider the communication energy overhead as a function of N, the number of sensor samples that are batched by the sensor and then transmitted in a burst to the smartphone. Fig 2 summarizes the key parameters associated with batched transmission in 802.11 and Bluetooth, which we now discuss.

1) IEEE 802.11: Commercial IEEE 802.11 radios can operate in two states–a normal 'active' mode (when the radio interface receives or transmits packets) and a Power Save Mode (PSM), where the radio periodically wakes up to check if there any pending transmissions or receptions. The following are two key relevant properties associated with 802.11 hardware:

- Due to the switching characteristics of the radio hardware, there is typically a lower bound on the minimal idle time Th_{idle}, below which the radio cannot enter the PSM mode (typically, this is around 100 ms) [9].
- There is a fixed, *duration-independent* switching energy E_{switch} spent when a radio transitions from the PSM to the 'active' mode.

Accordingly, it follows that the total transmission time for the N samples, generated over a time interval of $\frac{N}{f}$, equals $\frac{N*S}{B}$ and the total energy E_t consumed over this time interval equals:

$$E_t = \begin{cases} P_i * (\frac{N}{f} - \frac{N*S}{B}) \\ \quad +P_a * \frac{N*S}{B} + E_{switch} & \text{if} \frac{N}{f} - \frac{N*S}{B} > Th_{idle} \\ P_a * \frac{N}{f} & \text{otherwise} \end{cases} \quad (1)$$

The second case corresponds to the situation where the 'idle' time for the 802.11 radio is not enough for it to switch into the low-power PSM state. On the other hand, if the idle time is large enough, the energy per sample progressively diminishes, as the fixed cost of switching to a low power state is amortized over a longer 'idle' time.

2) Bluetooth: Bluetooth radios typically operate in three states: transmit, receive or sleep, each of which has a different power consumption profile [10]. As we focus principally on the smartphone, which primarily receives data from an external sensor, we denote its active energy consumption P_a as the energy spent in actively receiving data. We consider the Bluetooth version 2.0+ EDR and assume, for analytical tractability, that a single sensor device attaches as a slave to the master located on the smartphone. While the low-power mode results in significantly low power consumption, note that there is a latency T_{switch} involved in switching from the non-associated low-power mode to the associated-active mode. Accordingly, any data transfer duration would consist of the total time spent in transfer $\frac{N*S}{B}$, plus the additional time T_{switch}. Accordingly, the total energy consumed in transmitting the sensor stream in batches of N samples is given by:

$$E_t = P_i*(\frac{N}{f} - \frac{N*S}{B} - T_{switch}) + P_a*(\frac{N*S}{B} + T_{switch}) \quad (2)$$

Fig. 1 plots the resulting energy overhead (energy per sample) for both IEEE 802.11 and Bluetooth (computed by using Equations 1 and 2), as a function of the batch size N, for a representative accelerometer sensor, with $S = 192$ bits/sample and $f = 100$ Hz. It is clear that the choice of the batch size N, for a given radio technology, has a significant effect on the energy efficiency of the data acquisition process, and is thus an important design parameter for the ACQUA framework. For the specific accelerometer sensor considered here, the energy overhead for 802.11-based transmissions becomes dramatically lower when the sensor tuples are transferred in batches of 500 msecs or greater; for the Bluetooth interface, a batch duration of 5 secs or higher is more efficient.

III. The ACQUA Functional Architecture

We now consider the key properties that the ACQUA framework must consider, and then describe how the various functional components of ACQUA address these key design objectives. We first start by illustrating the basic principle of how the acquisition energy per sensor sample (i.e., a stream data tuple) and the selectivity characteristics of such tuples affect the ACQUA optimization framework.

Consider a hypothetical activity/wellness tracking application that seeks to detect an episode where an individual "walks for 10 minutes, while being exposed to an ambient temperature (95^{th} percentile over the 10 minute window) of greater than 80°F, while exhibiting an AVERAGE heart rate (over a 5 minute window) of > 80 beats/min". Assume that this application uses an external wrist-worn device, equipped with accelerometer (sensor S_1, sampling at 100 samples/sec), heart rate (S_2, sampling at 1 sample/sec) and temperature (S_3, sampling at 1 sample/sec) sensors. We'll address many of the precise semantic aspects of this query later (e.g., do we use tumbling vs. moving windows for averaging?) — for now, note that is essentially a *conjunctive* query, where the context requires the simultaneous satisfaction of three separate predicates, related to accelerometer, HR and temperature data streams.

Assume that the probability of the accelerometer readings

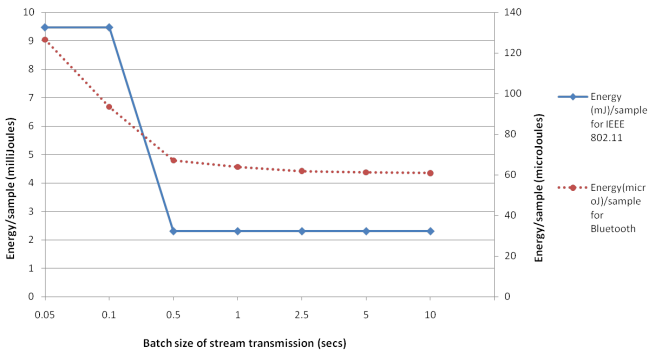


Fig. 1. The Impact of Batched Transmissions on the Transmission Energy Overhead per Sample. The figure plots the energy/sample for both 802.11 and Bluetooth interfaces, as a function of the batch duration, for a typical accelerometer sensor.

	IEEE 802.11	Bluetooth 2.0+EDR
P_a	947 mW	60mW
P_i	231 mW	5 mW
B	54 Mbps	1 Mbps
E_{switch}	14 μJoule	–
Th_{idle}	100 ms	–
T_{switch}	–	6 msec

Fig. 2. The Energy Overheads for IEEE 802.11g & Bluetooth Radios

Sensor Type	Bits/ sensor channel	Channels/ device	Typical sampling frequency (Hz)
GPS	1408	1	1 Hz
SpO2	3000	1	3 Hz
ECG (cardiac)	12	6	256 Hz
Accelero-meter	64	3	100 Hz
Temperature	20	1	256 Hz

Fig. 3. Representative Data Rates for Some Common Sensors

indicating that the user was walking for 10 mins, denoted by $P(S_1)$, equals 0.95; likewise, the probability of '95^{th} percentile of temp being greater than 80°F', $P(S_2)$, equals 0.05 and the probability of 'AVG(HR)' being greater than 80, $P(S_3)$, equals 0.2. Furthermore, given the potentially different sample sizes and transmission rates for each sensor, let us assume that the acquisition energy costs, denoted by $E(S_i)$ are as follows: $E(S_1) = 0.2$ nJ/sample; $E(S_2) = 0.02$ nJ/sample and $E(S_3) = 0.01$ nJ/sample.

We then observe that the choice of the best acquisition sequence should take into account both the acquisition energy cost and the selectivity properties. More specifically, we should ideally retrieve the data chunk from the sensor that should have a low acquisition cost and also a high likelihood of helping to terminate the predicate evaluation. For the conjunctive query, we note that a single 'FALSE' predicate implies that the complex predicate is FALSE and that the subsequent steps of predicate evaluation can be aborted. Accordingly, in our formulation, we first compute the '*normalized acquisition cost*' (**NAC**) as a ratio of the acquisition cost normalized by the 'predicate being FALSE' probability. Accordingly, we get $NAC(S_1)$=$100 * 0.02/0.05$, $NAC(S_2)$=$5 * 0.02/0.95$=0.105 and $NAC(S_3)$=$10 * 0.01/0.8$=0.125. Based on these computations, it would follow that the best sequence of acquiring the sensor data streams for evaluating the conjunctive query above would be $\{S_2, S_3, S_1\}$.

Consider instead the *disjunctive* query counterpart that seeks to detect an episode where an individual was either walking for 10 mins OR exposed to an ambient temperature (95^{th} percentile over the 10 minute window) of greater than 80°F, OR exhibited an AVERAGE heart rate (over a 5 minute window) of > 80 beats/min". For such a disjunctive query, the processing can terminate as soon as there is a single 'TRUE' predicate. Accordingly, in this case, the NAC should be computed as a ratio of the acquisition cost normalized by the 'predicate being TRUE' probability. Plugging in the values from before, we have $NAC(S_1)$=$100 * 0.02/0.95$=2.11, $NAC(S_2)$=$5 * 0.02/0.05$=2 and $NAC(S_3)$=$10 * 0.01/0.2$=0.5. Accordingly, the best sequence of acquiring the sensor data streams is now $\{S_3, S_2, S_1\}$.

The above examples illustrate how the ACQUA framework needs to take into account both the stream's query selectivity properties as well as the different wireless communication costs (acquisition costs) associated with the different sensor streams. We next describe some of the additional real-life artifacts that the ACQUA framework must consider.

A. *Functional Requirements from the ACQUA framework*

- **Accommodate Heterogeneity in Sensor Data Rates, Packet Sizes and Radio Characteristics:** Sensor data streams exhibit significant heterogeneity in terms of their sensor data rates (the number of sensor samples/sec), the data sizes (the bytes/sample) as well as the communication energy costs associated with their radio interfaces. As an illustration, Fig. 3 lists the data rates and sample sizes associated with a number of well-known medical and non-medical sensor streams. The communication energy costs will depend not just on the sensor type, but also on the specific wireless radio implementation on the embedded sensor device platforms. The ACQUA framework must thus be capable of incorporating different sensor data rates and wireless transmission characteristics in the query optimization framework.
- **Adapt to Dynamic Changes in Query Selectivity Properties:** To apply ACQUA, it is extremely important to have correct estimates for the query selectivity properties of different data streams. However, we need to keep in mind that these selectivity properties are not only individualized, but also vary dramatically over time due to changes in an individual's activity. For example, the likelihood of HR samples exceeding 80 might be very low when a person is engaged in sedentary office activity, but will be very high when the person is walking or working out in the gym. Accordingly, the ACQUA framework must be capable of

using context to accurately predict (albeit statistically) the selectivity characteristics of different sensor streams.

- **Take into Account other Objectives Besides Energy Minimization:** Operating with a heterogeneous set of sensors implies that energy minimization, while important, might not be the only objective of interest to a user of ACQUA. For example, it is possible that one of the N sensors might have very little battery capacity–in such a case, to extend the overall operational lifetime of the context detection activity, it might be more prudent to preferentially retrieve and process data from an alternative sensor, even though the selectivity characteristics of the alternative sensor may not be the highest.
- **Support Multiple Queries and Heterogeneous Time Window Semantics:** Sensor-based context extraction is becoming an intrinsic feature of a variety of smartphone applications that may be executing concurrently. Different applications may specify distinct predicates over a shared set of sensor streams–for example, the accelerometer sensor may be used to both evaluate step-counts in a wellness monitoring application and to understand the user's current mode of transport in a separate social networking application. The query predicates would differ not just in their predicate logic, but also in the time windows over which the stream query semantics are expressed. Accordingly, ACQUA must support a unified *application-independent* query representation framework and sequential data acquisition capability that is able to optimize the evaluation sequence across *all concurrently executing stream queries.*

B. The ACQUA Architecture

Fig. 4 shows the ACQUA functional architecture for supporting the energy-efficient, dynamically varying, sequential data retrieval from different sensor streams. The figure describes the logically distinct components of the ACQUA architecture — as such, a specific implementation may implement multiple functional components separately or as a single sub-system (e.g., the $SelectivityTracker$ component may be implemented either on the smartphone itself or on a backend 'cloud' component).

The heart of the ACQUA framework are the *Stream Selectivity Tracker (SST)* and the *Adaptive Stream Retrieval Subsystem (ASRS)* components. The SST is responsible for computing and establishing the selectivity properties of different sensor streams–in effect, computing the likely probability distribution of the values of each individual stream elements. To compute these values, ACQUA requires the SST to interface with the embedded Stream Processing Engine (SPE) to obtain the empirical observations of how the stream elements (individually or time-windows) satisfy different query predicates. The ASRS component is responsible for dynamically computing the sequence in which different (batches of) stream elements are retrieved by the smartphone from the locally-connected sensor. Note that the Stream Processing Engine (SPE) and the Sensor Data Adapter are pre-existing and non-ACQUA specific components needed to perform the basic functionality of a) performing the appropriate query execution on the incoming data streams and b) interfacing with the sensor to retrieve the appropriate sensor samples. The Query Specification Subsystem is another ACQUA component that is responsible for receiving the various query specifications (associated with multiple applications) and for compiling them into a common Predicate Evaluation Graph. This graph is the data structure used by the ASRS algorithms to determine the preferred sequence in which data is pulled from individual sensor streams–the formal model for this graph will be presented shortly (in Section IV). To algorithmically determine the best evaluation sequence, the ASRS also requires the knowledge of the energy per sample profile associated with different sensor devices and radios–it receives these specifications from the corresponding Sensor Data Adapter. As mentioned before, the ACQUA framework requires some degree of embedded data processing and storage capability on each individual sensor. In particular, the sensor-resident ACQUA components include the Data Cache, which acts as a temporary local repository for the stream tuples that may or may not be eventually pulled by the smartphone, and the Selective Data Transmitter, which is responsible for receiving requests for specific subsets of the stream tuples and for transmitting (in batches) these requested subsets.

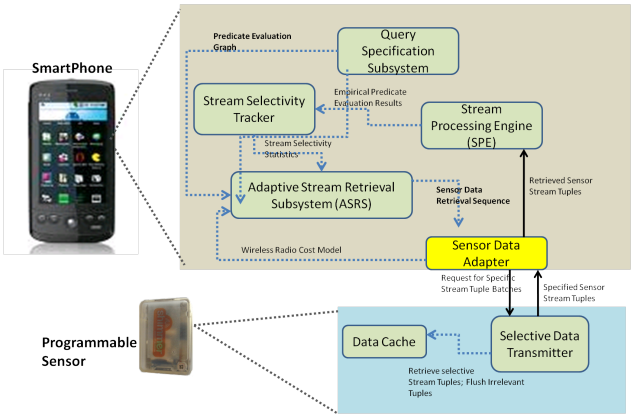


Fig. 4. Functional Component-Level Architecture of the ACQUA Framework.

A fully functional ACQUA-based stream processing framework will require the implementation and integration of all these subsystems. The focus of this paper is, however, principally on the ASRS algorithms that determine the optimal data acquisition sequence, and on understanding the likely benefits of such selective retrieval of sensor data. Accordingly, for the rest of this paper, we will focus on the study of the ASRS algorithms, and implicitly assume the a-priori availability of the *a)* stream selectivity statistics, *b)* the predicate evaluation graph and *c)* the wireless radio cost models.

IV. The Stream-Oriented Query Model

We now focus on the basic ASRS algorithms for acquisition-cost-aware query processing. We first need to mathematically rigorously define the types of queries that we consider.

Query Specification and Representation. For this paper, we consider complex stream queries that are expressed as arbitrary conjunction or disjunction predicates over a set of stream-oriented SQL aggregate (e.g., MAX or AVG) or user-defined (e.g., determining the Fourier coefficients) functions, defined over a time-window of each individual sensor stream. Mathematically, an individual query specification Q can be formally expressed as:

$$\begin{aligned} \text{Q} &::= Predicate \mid (\text{ Q } \textbf{AND } \text{ Q }) \mid (\text{ Q } \textbf{OR } \text{ Q }) \\ Predicate &::= AggFunc\ (\ SExp,\ w\ )\ CmpOp\ Const \mid \\ &\quad\ \textbf{NOT }\ Predicate \\ SExp &::= StreamName \mid \\ &\quad\ StreamExp\ ArithmeticOp\ Numeric \end{aligned}$$

where $AggFunction$ can be any SQL aggregate function or user defined function applied over a time window $(t-w, t)$ of stream values (t being the current time), $CmpOp$ can be any comparison operator such as $\{>, <, =\}$, $Const$ denote a constant of any appropriate type, $StreamName$ uniquely identifies a stream, $ArithmeticOp$ denotes an arithmetic operator $\{+, -, \div, \times\}$, and $Numeric$ denote a numeric constant.

All such queries are compiled into a uniform *Query Tree* representation, such that the root of the query tree represents the entire set of concurrent query predicates, an internal node is associated with a boolean conjunction or disjunction operator. and a leaf node is associated with a predicate. This query tree provides us the unifying application-independent query representation framework; hence, the specific ASRS algorithms are defined in terms of such a query tree. Fig. 5 illustrates the query trees for the following three example queries:

Q1: AVG$(A,5)$<70 AND (MAX$(B,4)$>100 OR C<3),
Q2: (AVG$(A,5)$<70 AND MAX$(B,4)$>100) OR
(C<3 AND Speed$(D,2) < 1.0$),
Q3: (AVG$(A,5)$<70 AND MAX$(B,4)$>100) OR
(C<3 AND MIN$(B,7) < 80$),

Query Q2 illustrates a query involving a user defined function for computing the average speed over the last two seconds of the values of stream D. Stream D could be displacement samples in the x, y, z axes from an accelerometer. Query Q3 illustrates a query where stream B appears in two predicates with different window sizes.

Evaluation Period ω. In this paper, we consider queries defined over *tumbling windows* of the individual sensor data streams. Formally, this implies the notion of a 'time shift' value $\omega(Q)$ associated with a query Q, such that the query is evaluated repeatedly at the time instants $t = (\omega, 2\omega, 3\omega, \ldots)$. Note that the time-shift value ω is distinct from the time windows associated with the individual predicates and operators of the query Q. For example, a specific query may be be defined to perform an $AVG(5)$ operation (i.e., an average of the last 5 seconds worth of sensor data with $\omega = 7$; in this case, the query would be evaluated at $t = 0$ over the stream tuples belonging to the time window $(-5, 0)$, and then again at time $t = 7$ over the time window $(2, 7)$. Fig. 6 illustrates this relationship between the query *time shift* value, the predicate time windows and the individual stream tuple-generation rates–the first three figures illustrate different cases for the query Q1 (Fig. 5(a)), while the fourth figure illustrates the evaluation for query Q3 (Fig. 5(c)) with $\omega = 5$.

An important consequence of this evaluation mode is the fact that the stream tuples needed for the evaluation of the query at a particular time instant may or may not be distinct from the tuples needed at a subsequent time instant. Consider query Q3 for the evaluation schedule in Fig. 6(d). Suppose each stream is associated with a buffer and the buffers are initially empty. At time $t = 7$, suppose the evaluation of Q3 requires the retrieval of the data elements $A : (2, 7], B : (3, 7], C : (6, 7]$. Now, at time $t = 12$, suppose MAX$(B, 4) > 100$ is evaluated first and is false; we need proceed to evaluate the right subtree of the top OR node (in Fig. 5(c)). Note now that the evaluation of MIN$(B, 7) < 80$ requires the stream tuples $B : (5, 12)$. However, a subset of this required set of tuples has already been previously acquired–only the samples $B : (7, 12]$ need to be acquired. *This example illustrates that, even with tumbling window queries, the acquisition cost for a particular sensor stream may be different at different evaluation instants, depending upon the data tuples that may have been acquired during prior event processing.*

V. The ASRS Sequential Retrieval Algorithm

Having formally defined the semantics of our query, we now proceed to define the algorithm for computing the preferred data retrieval and evaluation sequence. For purposes of simplifying the exposition, we make the following two assumptions in this section: *a)* Each unique sensor stream in the query tree is associated with a single 'tumbling window' value, even though the same window of the sensor data can appear in multiple nodes of the query tree and be associated with multiple predicates, and *b)* The unique sensor-specific 'tumbling window' value defines the basic batch size in which the smartphone's Sensor Data Adapter retrieves data from each sensor.

Algorithm 1 defines the high-level logic of query evaluation. Intuitively, following the approach discussed in Section III, the algorithm first computes the lowest expected cost of evaluating different portions of the query sub-trees, and thereby determines (using the recursive Algorithm 2 CALCACQUISITIONCOST) the optimal sequence for retrieving the data from the different sensor streams. Subsequently, the actual query is evaluated using the recursive Algorithm 3 EVALUATEQUERY, which essentially follows the specified sequence to evaluate the sub-trees. As mentioned previously, the actual retrieval of data tuples for a given stream needs to consider the relevant tuples that have already been retrieved (at prior instants or while processing other parts of the query tree): Line 3 in Algorithm 3 achieves this by adjusting the window of data tuples that are actually retrieved from the corresponding sensor.

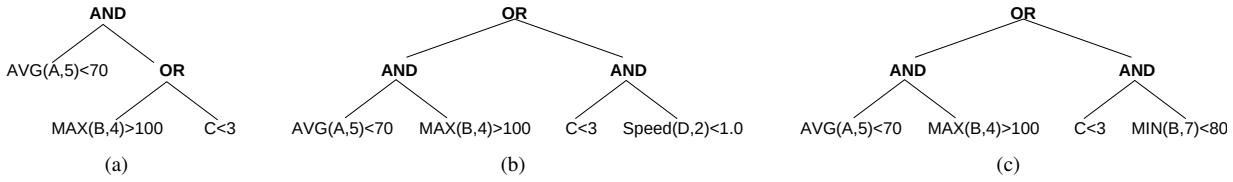
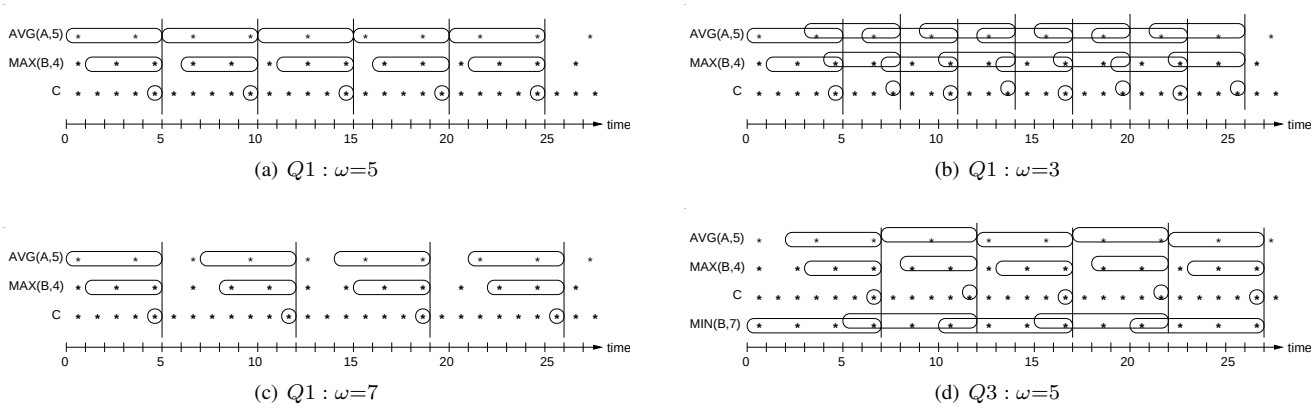


Fig. 5. Query trees for three example queries.



(a) $Q1 : \omega$=5 (b) $Q1 : \omega$=3

(c) $Q1 : \omega$=7 (d) $Q3 : \omega$=5

Fig. 6. Relationship between query evaluation period, predicate time windows, and stream rates. An asterisk denotes a sampled tuple of a particular stream at a particular time. The rounded rectangles denote the sample or the time window of samples required in the evaluation at each evaluation event.

Algorithm 1 PROCESSQUERY(q, P, ω)

Input: Query tree q, probability P of each subquery evaluating to true/false, evaluation period ω
Output: Alert Stream

```
1: loop
2:    t ← current time
3:    CALCACQUISITIONCOST(q, t, P, C)
4:    if EVALUATEQUERY(q, t, P, C) = true then
5:       output alert tuple
6:    sleep ω seconds
```

Algorithm 2 CALCACQUISITIONCOST(q, t, P, C)

Input: Query tree q, current time t, probability function P, data acquisition cost function $C(\cdot)$
Output: Updates cost function $C(\cdot)$

```
1: if q is a predicate node then
2:    let s be the stream that q operates on, w be the window size
      of q, t_s be the latest time the buffer for s was updated.
3:    C(q) ← Calculate cost for acquiring the samples in time
      interval (max(t − w, t_s), t] for stream s using Eqn 1 or Eqn 2.
4: else
5:    CALCACQUISITIONCOST(q.left, t, P, C)
6:    CALCACQUISITIONCOST(q.right, t, P, C)
7:    if q.op = AND then
8:       C(q) ← Eqn. 3
9:    else
10:      C(q) ← Eqn. 4
```

Algorithm 2 CALCACQUISITIONCOST computes the acquisition cost of evaluating a query subtree according to the evaluation sequence determined by NAC. At a query tree node with an AND operator, we recursively calculate the data acquisition cost of the left and right subtrees (henceforth denoted by L and R respectively). The NAC for L and R are then computed using the probability of L and R evaluating to true or false. The order of evaluation (LR or RL) is completely determined by the NAC and we can now calculate the acquisition cost for the current node as,

$$C(q) = \begin{cases} P(q.left) \times [C(q.left) + C(q.right)] & \\ \quad +P(\neg q.left) \times C(q.left) & \text{if LR} \\ P(q.right) \times [C(q.left) + C(q.right)] & \\ \quad +P(\neg q.right) \times C(q.right) & \text{if RL} \end{cases} \tag{3}$$

A similar analysis can be applied for a query node with an OR operator, resulting in a acquisition cost of,

$$C(q) = \begin{cases} P(\neg q.left) \times [C(q.left) + C(q.right)] & \\ \quad +P(q.left) \times C(q.left) & \text{if LR} \\ P(\neg q.right) \times [C(q.left) + C(q.right)] & \\ \quad +P(q.right) \times C(q.right) & \text{if RL} \end{cases} \tag{4}$$

We note that the calculation is dependent on the probability function $P(\cdot)$ for each node in the query tree evaluating to true or false. These probabilities can be obtained statically from historical executions or more dynamically by keeping counters for the truth value of the evaluation of each query tree node.

The recursion ends when a predicate node is reached. A predicate node is associated with a stream and the cost of evaluating a predicate node is calculated using Eqn 1 or Eqn 2 depending on the transmission type (802.11 or Bluetooth), assuming the current state of the buffer associated with the stream. No actual data acquisition occurs in this computation. The result of CALCACQUISITIONCOST is that the acquisition cost function $C(\cdot)$ is now updated for each node in the query tree based on the current snapshot of the buffers for all the dependent streams. We are now ready to evaluate the query

Algorithm 3 EVALUATEQUERY(q, t, P, C)

Input: Query tree q, current time t, probability function P, data acquisition cost function $C(\cdot)$
Output: Truth value of q

1: **if** q is a predicate node **then**
2: let s be the stream that q operates on, w be the window size of q, t_s be the latest time the buffer for s was updated.
3: Acquire the samples in time interval $(\max(t-w, t_s), t]$ for stream s.
4: Update $C(\cdot)$ if s is used in multiple predicates
5: $truthval \leftarrow$ evaluate predicate q
6: return $truthval$
7: **else**
8: **if** $q.op =$ AND **then**
9: $leftshortcircuits \leftarrow P(\neg q.left)$
10: $rightshortcircuits \leftarrow P(\neg q.right)$
11: $shortcircuitval \leftarrow false$
12: **else**
13: $leftshortcircuits \leftarrow P(q.left)$
14: $rightshortcircuits \leftarrow P(q.right)$
15: $shortcircuitval \leftarrow true$
16: $evalorder \leftarrow (q.left, q.right)$
17: **if** $\frac{C(q.left)}{leftshortcircuits} > \frac{C(q.right)}{rightshortcircuits}$ **then**
18: $evalorder \leftarrow (q.right, q.left)$
19: **for all** $q' \in evalorder$ **do**
20: $truthval \leftarrow$ EVALUATEQUERY(q', t, P, C)
21: **if** $truthval = shortcircuitval$ **then**
22: return $truthval$
23: return $\neg shortcircuitval$

using the updated cost function $C(\cdot)$ and the probabilities function $P(\cdot)$ for each node in the query tree.

The actual acquisition of sensor data and the evaluation of the predicates occur in the recursive Algorithm 3 EVALUATEQUERY. The base case occurs at the predicate (leaf) nodes of the query tree. The required data tuples are retrieved from the dependent stream if they are not already in the stream buffer (Line 3). In queries where a particular stream is involved in multiple predicates, a data acquisition may change the acquisition cost of another predicate on the same stream. Line 4 updates the cost function $C(\cdot)$ for the query tree nodes affected by the stream buffer update. Finally, the predicate is evaluated and the truth value returned.

For the recursive case, EVALUATEQUERY computes the NAC of the query node's left and right subtrees using $C(\cdot)$ and $P(\cdot)$. The subtree with the lower NAC is recursively evaluated first. If the truth value of the evaluation results in a short circuit, the other subtree need not be evaluated and hence no data is acquired for the that subtree.

VI. PERFORMANCE EVALUATION AND RESULTS

We now describe the result of simulation-based studies to quantify the performance gains (in terms of the reduction in energy overheads) of our proposed ACQUA algorithms. Our studies are conducted using a Perl-based simulator which accepts as input both a query tree and probability distributions on the values of individual data streams. Synthetic traces of sensor-generated data tuples were then generated to reflect the probability distributions and fed into the simulator, which then applied the algorithms of Section V to compute the sequence of data that would be actually retrieved by an ACQUA-based implementation. Results are presented by averaging over 5 one-hour long traces and also include the 95% confidence intervals.

To quantify our performance gains, we compared three different evaluation algorithms:

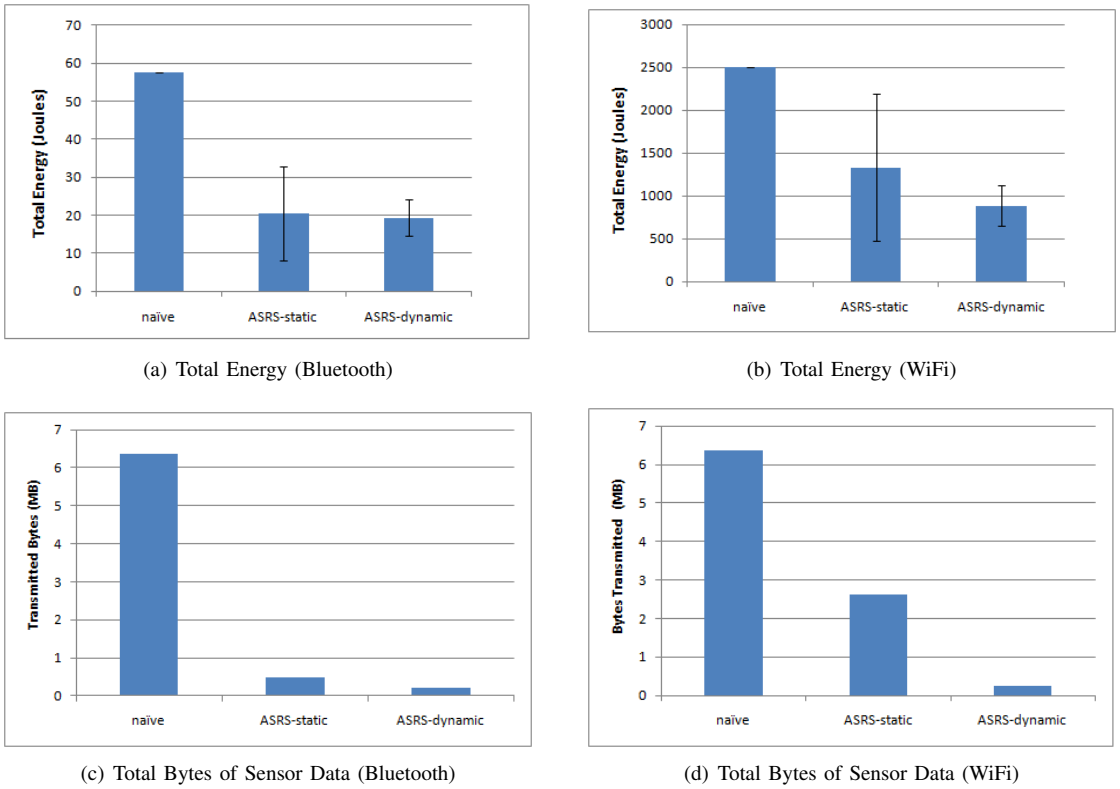
1) **Naive:** The naive retrieval algorithm requires each sensor to simply upload (in batched mode) its generated stream tuples to the SPE. Accordingly, while this algorithm utilizes batched transmission to reduce the energy overheads associated with the use of the PAN wireless interface, it does not exploit the selectivity properties to reduce the amount of sensor data that is actually needed by the SPE.
2) **ASRS-dynamic:** The ASRS-dynamic algorithm corresponds to the procedure described in Section V and requires the dynamic modification of the acquisition cost functions after each data retrieval and evaluation, to account for both the stream tuples already present in the smartphone buffer and the already-resolved ('short-circuited') query subtrees.
3) **ASRS-static:** While this algorithm's logic is broadly similar to ASRS-dynamic, it computes an optimal sequence only once (at the beginning of the simulation) based on the selectivity characteristics and the communication costs, and then applies the $EvaluateQuery()$ procedure to evaluate the query tree at successive 'time shift' instants. Accordingly, it does not perform the dynamic update of NAC values, based on the dynamically evolving state of the query processing state.

While $Naive$ helps to quantify the performance gains expected from our sequential acquisition strategy, ASRS-static helps us to isolate and understand the performance gains that arise from the dynamic consideration of the evolving query state.

Given our focus on understanding the expected benefits of our proposed ASRS algorithms, we focus on a single, relatively-simple but representative query:

> Generate an alert if (($AVG(SP02, 5sec)$<98%) AND
> (($SPREAD(Accel, 10sec)$<$2g$) AND
> ($AVG(HR, 10sec)$<75))) OR
> (($AVG(SPO2, 5sec)$<95) AND
> (($SPREAD(Accel, 10sec)$>$4g$) AND
> ($AVG(HR, 10sec)$>100))).

Intuitively, this query generates alerts either if the user's Sp02 values drop below 98% while the user is resting, or if the Sp02 values drop below 95% while the individual is engaged in vigorous activity (e.g., running). The accelerometer and Sp02 sampling rates and data sizes are adapted from Fig. 3, while the heart rate sensor has a sampling frequency of 0.5 Hz and a sample size of 32 bits. We experimented with both 802.11 and Bluetooth-based wireless transmission models. The underlying data traces are generated using the normal distribution $N(\mu, \sigma)$ (with appropriate truncation to avoid underflow below 0 or



(a) Total Energy (Bluetooth)

(b) Total Energy (WiFi)

(c) Total Bytes of Sensor Data (Bluetooth)

(d) Total Bytes of Sensor Data (WiFi)

Fig. 7. Comparative Energy (Joules) and Data (bytes) Overheads, with $\omega = 10secs$.

overflow above 100%) on each of these sensors as follows: Sp02 as $N(96, 4)$, HR as $N(80, 40)$ and Accel as $N(0, 10)$.

A. Evaluation with a Fixed Time-Shift Value for Each Stream

Figures 7(a) and 7(b) plot the total data acquisition energy (in Joules, over the 1 hour evaluation duration) for each of the three strategies, for the case of Bluetooth and 802.11-based PAN technologies respectively. These results correspond to a query with a time-shift value of $\omega = 10secs$. It is easy to see that our approach of sequential retrieval and evaluation of individual sensor streams, while taking into account their respective acquisition costs and selectivity characteristics, results in significant energy savings, compared to the naive approach where the data is pushed (albeit in batches) from each sensor. In particular, for 802.11 based transmissions, ASRS-static and ASRS-dynamic result in $\sim 50\%$ and $\sim 70\%$ reduction in energy overheads compared to the Naive scheme. For Bluetooth-based data transfers, the energy reductions are equally dramatic, with ASRS-static and ASRS-dynamic both achieving $\sim 70\%$ savings in energy overheads.

The results also demonstrate the benefits of ASRS-dynamic: by taking the dynamic state of a query and the contents of the data buffer into account, this approach is able to further reduce the energy overhead, compared to the static counterpart. The gains are, however, not as dramatic for the Bluetooth interface (even though ASRS-dynamic has significantly lower variance than ASRS-static)–this is most likely due to the non-negligible T_{switch} overhead in Bluetooth, which implies that Bluetooth does not provide as great an advantage for very short-sized data transfers compared to larger batch sizes. The figures thus reveal that the relative performance of the algorithms depend significantly on the fine-grained features of the PAN radio technology, implying that the ACQUA algorithms need to be carefully tailored to the characteristics of the specific PAN technology adopted.

Figures 7(c) and 7(d) similarly plot the total data overhead (in bytes). While the ASRS algorithms clearly require an order-of-magnitude less communication than the Naive counterpart, it is interesting to note that the energy savings are not directly proportional to the communication overheads. For example, with Bluetooth, ASRS-dynamic requires about 50% fewer bytes of sensor data than ASRS-static, but has only a $\sim 10\%$ lower energy overhead.

B. Evaluation under Varying Time-Shift Values

We also experimented with different values of the 'time shift' window ω, i.e., by altering the frequency with which our 'tumbling window' query is evaluated. Figure 8 shows the energy overheads for the three algorithms for three different values of $\omega = \{5sec, 10sec, 20sec\}$, for the case of IEEE 802.11-based sensor data transfers; $\omega = 3$ implies an overlap of time windows of successive evaluation instants. (Results for Bluetooth-based transfers are qualitatively similar and omitted due to space constraints.) It is interesting to observe that

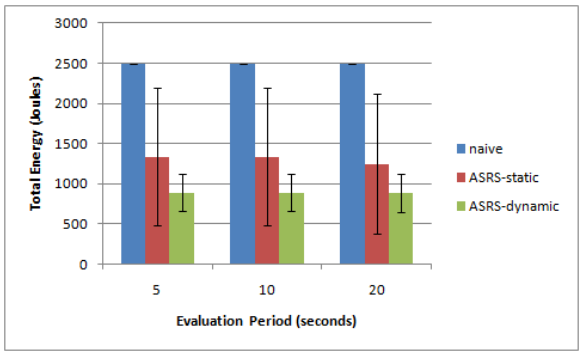


Fig. 8. Comparative Energy Overheads, under varying ω values, for 802.11-based Data Transfers

the relative gains are fairly independent of ω. In particular, when $\omega = 20$, there is no overlap between the evaluation window and the 'time shift' values; accordingly, the evaluation at a subsequent instant always starts with an empty buffer of data tuples. Nonetheless, the ASRS-dynamic algorithm is able to outperform the static variant, by better adapting its data acquisition sequence to take account of the intermediate query evaluations state (i.e., by eliminating data acquisition for those sub-trees that have already been 'short-circuited').

VII. Conclusion and Future Work

In this paper, we have motivated the ACQUA framework for energy-efficient continuous evaluation of complex queries over sensor-generated data on a smartphone. The key to the ACQUA framework is the sequential retrieval of subsets of data tuples from each individual stream, with the preferred sequence being determined by considering both the query selectivity properties of the individual data stream and the sensor-specific energy overheads incurred by the sensor in transmitting the data over a PAN wireless network to the smartphone. We described two algorithms that both consider in detail the transmission costs arising from batched transmission of sensor data tuples– while ASRS-static determines an optimal retrieval sequence once when the query is submitted for execution, ASRS-dynamic re-evaluates the optimal retrieval sequence at each evaluation instant, taking into consideration the state of both the stream buffers and the partially evaluated query. Our results on synthetic traces indicate that the ACQUA approach can result in $\sim 70\%$ reduction in the energy overheads of continuous query processing, *without any degradation in the fidelity of the processing logic.*

We conclude by emphasizing that the overall ACQUA effort is very much work in progress and encompasses two orthogonal threads. The algorithms presented in this paper *assume* the availability of the selectivity statistics for each sensor stream. At a systems level, we are working to implement the ACQUA components on an Android-based smartphone platform, with special focus on online-learning algorithms to estimate the selectivity statistics from the history of sensor-generated data. Subsequently, user studies with real-life sensor traces will be used to quantify the performance gains of the ACQUA framework using real-life, instead of currently-used synthetically generated, sensor traces. On an algorithmic level, we are working to define the algorithms to include additional query semantics, such as the support of *sliding window* queries and on techniques to further improve the dynamic computation of the cost functions, given the characteristics of the data tuples already available in the system buffer (for example, while the probability of $AVG(S5, 10) > 40$ may be generically 0.8, the probability at a specific instant should be different if 8 out of 10 samples in that evaluation window are already buffered and are all observed to be less than 10).

References

[1] S. Gaonkar, J. Li, R. Roy Choudhury, L. Cox and A. Schmidt, Micro-Blog: Sharing and Querying Content through Mobile Phones and Social Participation, Proceedings of ACM Mobisys'08, June 2008.
[2] E. Miluzzo, Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application., Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys '08), November 2008.
[3] I. Mohomed, A. Misra, M. Ebling and W. Jerome, Context-Aware and Personalized Event Filtering for Low-Overhead Continuous Remote Health Monitoring, IEEE WoWMoM 2008, June 2008.
[4] The SHIMMER sensor platform http://shimmer-research.com.
[5] B. Priyantha, D. Lymberopoulos and J. Liu, Enabling energy efficient continuous sensing on mobile phones with LittleRock, Proceedings of IPSN, April 2010.
[6] H. Lu, J. Yang, Z. Lu, N. Lane, T. Choudhury and A. Campbell, The Jigsaw Continuous Sensing Engine for Mobile phone Applications, Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys '10), November 2010.
[7] J. Liu and L. Zhong, Micro Power Management of Active 802.11 Interfaces, Proceedings of ACM Mobisys'08, June 2008.
[8] A. Roychoudhury, B. Falchuk and A.Misra, MediAlly: A Provenance-Aware Remote Health Monitoring Middleware, 8th IEEE International Conference on Pervasive Computing and Communications (PerCom), March2010.
[9] F. Dogar, P. Steenkiste and D. Papagiannaki, Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices, Proceedings of ACM Mobisys'10, June 2010.
[10] K. Jang, T. Lee, H.Kang and J. Park, Efficient Power Management Policy in Bluetooth, IEICE Transactions on Communication, August 2001.
[11] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, Model driven data acquisition in sensor networks, in Proceedings of VLDB. Margan Kaufmann Publishers Inc., 2004, pp. 144155.