# Optimizing Push/Pull Envelopes for Energy-Efficient Cloud-Sensor Systems

Yi Xu
CISE Department
University of Florida
Gainesville, FL32611, USA
yixu@cise.ufl.edu

Sumi Helal
CISE Department
University of Florida
Gainesville, FL32611, USA
helal@cise.ufl.edu

My T. Thai
CISE Department
University of Florida
Gainesville, FL32611, USA
mythai@cise.ufl.edu

Mark Schmalz
CISE Department
University of Florida
Gainesville, FL32611, USA
mssz@cise.ufl.edu

## ABSTRACT

Unlike traditional distributed systems, where the resources/needs of computation and communication dominate the performance equation, sensor-based systems (SBS) raise new metrics and requirements for sensors as well as for computing and communication. This includes sensing latency and energy consumption. In this paper, we present a performance model for SBS based on a three-tier architecture that uses edge devices to connect massive-scale networks of sensors to the cloud. In this architecture, which we call *Cloud, Edge, and Beneath* (CEB), initial processing of sensor data occurs in- and near-network, in order to achieve system sentience and energy efficiency. To optimize CEB performance, we propose the concept of *optimal push/pull envelope* (PPE). PPE dynamically and minimally adjusts the base push and pull rates for each sensor, according to the relative characteristics of sensor requests (demand side from the Cloud) and sensor data change (supply side from Beneath). We demonstrate the CEB architecture and its push/pull envelope optimization algorithm in an experimental evaluation that measures energy savings and sentience efficiency over a wide range of practical constraints. In addition, from the experiments we demonstrate that by combining PPE optimization algorithm with lazy sampling algorithm, we can achieve further energy saving.

## Categories and Subject Descriptors

C.2 COMPUTER-COMMUNICATION NETWORKS. C.2.1 Network Architecture and Design. C.2.4 Distributed Systems. C.4 PERFORMANCE OF SYSTEMS.

## General Terms

Algorithms, Experimentation, Measurement, Performance.

## Keywords

Pervasive computing, cloud computing, energy efficiency, sentience efficiency, push pull envelope, optimization, performance

## 1. INTRODUCTION

Mobile and pervasive computing have recently garnered significant attention because of their potential to enable novel and attractive solutions in areas such as environmental monitoring [7], transportation enterprises [16], and health care [8]. The development of mobile and pervasive computing has benefitted from many different views of the communication and computing universe such as layered design, client-server, distributed networks, cloud-based computing, and so forth. In particular, cloud computing provides on-demand provision of computational services, allowing end users to access applications and data from a cloud on demand, anywhere in the world. Applications are hosted in terms of the *Software as a Service* (SaaS) model, whereby cloud server(s) automatically scale to meet client demand. However, cloud-based computing paradigms also exhibit theoretical and implementational limitations and disadvantages. For example, cloud-based computing is limited by a lack of depth and expressiveness. This tends to promote problems when attempting to abstract sensor and application layers, in a physically rigorous manner that must be convenient for system designers. By way of illustration, we first consider a traditional model that combines pervasive computing with the cloud computing paradigm, then contrast this with the CEB paradigm.

### 1.1 The Two-Layered Model: Scalability Challenges in Cloud-Based Sensing

In typical practice, cloud computing providers deliver common business applications online that can be accessed from a Web service such as a browser, while the actual programs and data are stored on servers. Clouds often appear to users as single points of access for their computing needs, and cloud-based products are usually expected to meet customers' quality of service (QoS) requirements, typically including service level agreements (SLAs). As a result, the emergence of cloud-based computing has provided ample opportunity for rejuvenation of a maturing information technology (IT) industry, in terms of a layered paradigm (*user* > *application* > *cloud* > *supporting devices*) with unique QoS and accounting models and practices.

In customary models of cloud-based computing, applications reside in the cloud, and devices that service the cloud's provision of applications reside in some device layer. Additionally, the emergence of pervasive sensing seeks to exploit sensors from cloud-based applications. Since the focus of this paper is on sensor computing in active pervasive spaces, we assume that the device layer includes sensors. Unfortunately, this two-layered model, shown notionally in Figure 1.1, has the following drawbacks that make it impractical for pervasive sensor computing:

- The massiveness (scope and detail) of sensor hardware in pervasive spaces makes coherent sensor management and maintenance extremely difficult. As a result, system complexity does not scale linearly with an increasing number of sensors and associated hardware devices.
- Application programmers must know many implementational details of sensor and communication hardware, in order to successfully program the two-layer model (cloud and sensors)

to realize pervasive sensing. This implies the presence of system integration knowledge, personnel, and effort, which can be expensive, time-consuming, and error-prone.

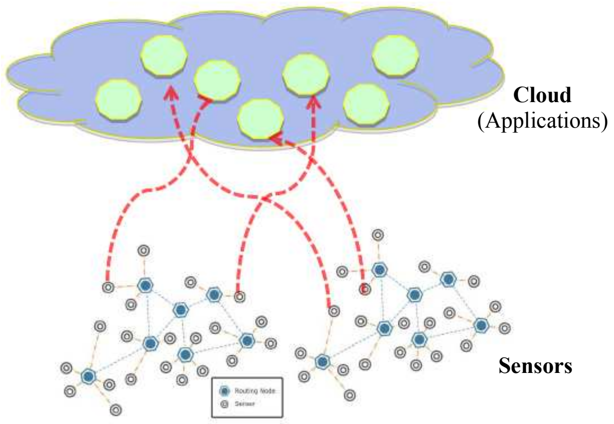


**Figure 1.1. Customary two-layer model of cloud-based sensor computing**

For purposes of discussion, let us suppose that this two-layer model is applied to cloud computing, where applications are in the cloud and sensors are beneath the cloud. Here, it is possible that an application directly connected to the device layer could make excessive demands on the devices, thereby overloading the devices. For example, sensors deployed in a busy highway to monitor traffic conditions can soon run out of power due to the fact that they could be overpowered when being accessed by large number of applications at the same time. This could cause excess energy consumption, with possible harm to the sensors or other objects in the device layer. Similarly, it is possible to slow down or "crash" the cloud, by having too many sensors for the cloud to process efficiently.

Further, to achieve efficiency and accuracy, we have found that the user should not be involved in dynamic interactions between the cloud and device layers. Thus, we have found an important additional flaw in the two-layer model:

- Importantly for wireless applications, the two-layer model raises significant concerns about energy consumption. In particular, applications in the cloud (by definition of cloud computing) need not know of each other's existence or functionality. Thus, when different applications request data from one sensor or group of sensors, these requests can be processed separately. This property supports redundancy of requests directed to a given sensor or subnet, which leads to redundant data acquisition that causes unnecessary energy consumption. Recall that cloud applications need not know of, or communicate among, each other. Thus, there exists no structural basis for optimizing the cloud's use of sensor hardware in terms of energy consumption, or other parameters such as field-of-view; optimization of spatial, temporal, or spectral coverage; latency hiding; and performance constraints on collaborative distributed sensing.

To opmization we propose a three- layered model (diagrammed in Figure 1.2) that has a novel layer called the *edge*, which resides between the cloud and sensor layers. We discuss the properties of this new model, as follows.

## 1.2 The Three-Layered Model: *Cloud*, *Edge*, and *Beneath* (CEB)

In the three-layered (CEB) model of sensor computing, sensors are partitioned into groups, each of which is managed by a single edge. In practice, an edge could be a mobile device, a computer, or a collection of sensor devices or computers. Such edge devices could be controlled by an application designed to monitor multiple conditions or activities within a particular environment. For example, the edge could be controlled by a homeowner via a security application, and be used for managing sensors deployed in his home (e.g., temperature, humidity, smoke, fire, or intrusion sensors). Similarly, sensing devices could be employed in monitoring patient health at home, in order to supply assistive services. Thus, sensors can be conveniently grouped implementationally by *type* (e.g., scalar, staring-array, spectral), *functionality* (e.g., temperature, surveillance camera), or *location* (e.g., within a building or an office complex, or outdoors).

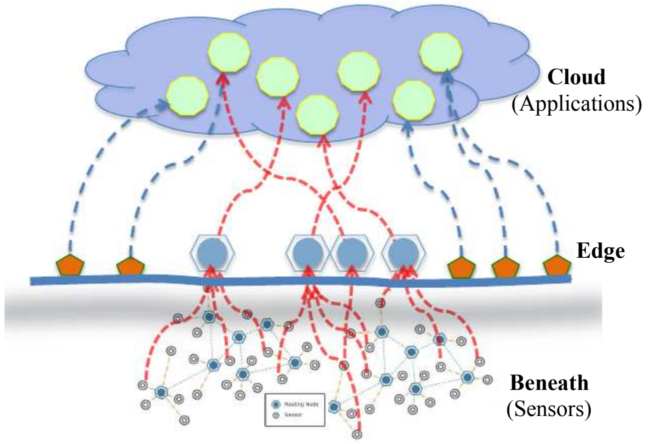


**Figure 1.2. Three-layer model of sensor computing with cloud, edge, and beneath**

In contrast with the traditional two-layered model illustrated in Figure 1.1, our three-layered CEB model realizes the following advantages and benefits:

- Sensor management and maintenance are performed by one or more edge(s), to which the sensors are connected. Thus, locality of control is preserved, since an edge only controls sensors located within its domain. As a result of such partitioning, scalability is significantly improved – as opposed to the two-layer model where sensors can be globally controlled by multiple competing or interacting applications (as discussed in Section 1.1).
- Edges can abstract unnecessary information about implementational or internal details of sensors, routers, and other sensing-related hardware located in the *Beneath* layer. This directly facilitates tractable models and programming of sensor configuration and control procedures, which we have previously shown to be efficient and robust in practice [17].
- Edges support staging and optimization, while allowing for many types of optimization procedures to be implemented. Edges also implement *localization* and decoupling, to achieve good software engineering practice through a loosely-coupled, layered architecture. In practice, edges can implement staging and power optimization by more efficiently retrieving sensor data on behalf of applications, for example, by *caching*. Via

data caching mechanisms, it is possible to reuse historical sensor data, thereby reducing energy consumption associated with repeated sensor data acquisition and sampling by multiple applications. In addition to energy reduction, caching could conceivably allow a reduction in the total number of sensors, as coverage could be partially managed by the caching capabilities of one or more edges.

Therefore, the introduction of the edge layer makes the three-layer architecture more tractable, better engineered, and more efficient. However, the minimization of energy consumption still remains a concern. In response to this situation, this paper describes the concept of an *optimal push-pull* (Section 2). We briefly explain a supporting event-driven programming model (Section 3), then present our optimization approaches (Section 4): In particular, our OPT-1 optimization algorithm is based on the relative characteristics of demand side (from the *Cloud* layer), while OPT-2 exploits the relative characteristics of data on the supply side (from *Beneath*). We also demonstrate experimentally determined performance gains of OPT-1, OPT-2, and their combination, in terms of practical applications.

## 2. Problem: Energy Efficiency of Cloud, Edge and Beneath

Recall that the two-layer model (Section 1.1) generates repeated data acquisition requests from one or more applications in the cloud to sensors in the device layer. This behavior can cause significant power consumption due to repeated sensor sampling. In the three layer model, we seek to minimize energy consumption by sentient control of sensors in the beneath (device) layer via optimization algorithms in the edge layer. Correspondingly, we have developed optimization algorithms that efficiently manage energy consumption between the cloud and edge layers.

In particular, our research shows that the updating of sensor readings can be realized via communication between cloud, edge, and beneath layers using information push and pull mechanisms. Push allows the data sink (e.g., the edge layer) to subscribe to a particular data source (e.g., a sensing device in the sensor layer), to received continuous readings at a constant rate. In contrast, information pull supports on-demand data query, such that the data sink can request, then acquire, sensor readings as one or more individual values. Advantages and disadvantages of these approaches are discussed, as follows.

### 2.1 The Push/Pull Envelope

When sensor data are needed at a constant rate, push requires less downlink traffic (cloud to edge, edge to sensor), since the application issues one subscription request only, then acquires the stream of values from the sensor. In contrast, pull incurs a round-trip penalty for each data query (e.g., involving steps such as request, transmit, receive, and acknowledge). However, push mode tends to be less economical when handling sporadic data requests, as an application-subscribed sensor in push mode samples and transmits a stream of data, whether or not all data values are processed by the application. In push mode, this physical transmission of the sensor value stream occurs continuously – even when the data are not needed by the application – leading to a substantial expenditure of energy. Since some sensor values are not processed, some of this energy is wasted. Thus, to achieve a design tradeoff between energy costs associated with push and pull modes, we need to know something about the application. For example, as shown in Figure 2.1 we have found that, when transitioning from MICA2 [18] to RCB [6], network communication decreases from 25 percent to 4.5 percent of total energy cost. Correspondingly sensor energy consumption due to sampling increases from 74 percent to 94 percent.

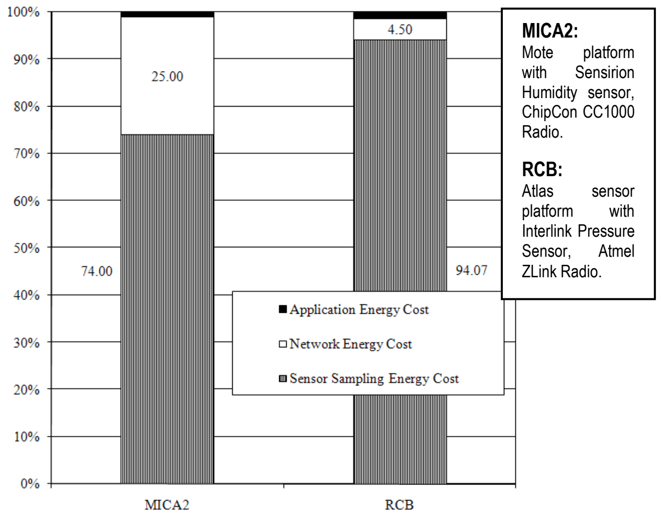


**Figure 2.1. Differences in energy consumption between MICA2 and RCB (lower power, same sampling technology) for wireless sensing applications.**

In response to this problem, we have developed the concept of a *push-pull envelope* (PPE), which is defined as follows.

**Definition 1:** An *optimal push-pull envelope* is a multidimensional sequence that determines (a) which sensors are active, and (b) which sensors should be in pull or push mode (along with push filters). The push-pull envelope minimizes work and total energy consumption in the sensor layer, as well as in the cloud, thus maximizing *sentience efficiency*.

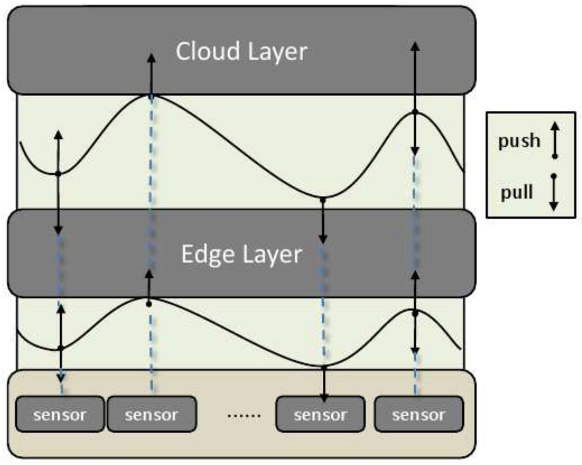


**Figure 2.2. Push-pull envelope between cloud and edge, and between edge and sensors**

In practice, the PPE can be thought of as a hybrid approach for achieving near-optimal energy consumption, as illustrated notionally in Figure 2.2. Architecturally, a PPE can exist between the cloud and edge layers, as well as between the edge and sensor layers. In a slightly lower-level view, a PPE can be thought of as an optimal configuration of push/pull modes for each sensor over the lifetime of service execution in the cloud. In this perspective, the PPE effectively describes an optimal execution of applications whose combined energy consumption (due to push and pull operations) is minimized. This does not imply a static view – the PPE can vary spatiotemporally. Therefore, in this paper, we seek to dynamically compute the optimal PPE between adjacent layers of the three-layer model that optimizes system energy efficiency.

Before presenting our algorithm for finding the optimal PPE, we overview an event-driven programming model that supports the programming paradigm employed in our system development of applications in the cloud.

## 3. Event-Driven Programming Model

An event-driven programming model is a programming paradigm whose control and data flows are determined by events. In pervasive sensor computing, an event is always associated with sensor outputs. In practice, applications developed on event-driven programming paradigms usually consist of two phases: (1) event detection, and (2) event handling.

In particular, in our model, the basic element of an application is a *rule*, which conforms to the *Event, Condition, and Action* (ECA) structure. By specifying events, conditions and actions, program-mers formulate rules that represent constraints on permissible application behavior over a range of situations. To ensure the adherence to these rules, the application constantly checks sensor data and evaluates the prespecified rules. When a rule evaluates to *true* in response to one or more events, corresponding actions (e.g. actuating a device or invoking a service) are executed, to respond to the event. Details about the design and implementation of ECA are found in our early work [15].

We have extended the event formulation structure to support composite events [15]. For example, an event associated with a single sensor only is called a *basic event*, while an event defined in terms of one or more basic events is called a *composite event*. Composite events usually require data from multiple sensors. For example, assume that a basic event $e_1$ is associated with a thermometer and another basic event $e_2$ is associated with a humidity sensor. Then a composite event could be defined based on $e_1$ and $e_2$, with the purpose of monitoring a physical event such as current weather condition.

In the next section, we present our optimization approaches to realize energy efficiency of systems that are build based on this event-driven programming model.

## 4. Cloud-Sensor Energy Optimization

Our research has shown that an optimal PPE can be achieved via two optimizations (called OPT-1 and OPT-2) that collaboratively determine the data delivery strategy among the three layers of our CEB sensor computing model. For each sensor, OPT-1 attempts to compute an optimal configuration of the push/pull envelope between the cloud and edge layers. Via monitoring data requests from the cloud, the edge learns the history of relative characteristics of an application's demands for each sensor. The edge then utilizes this reference knowledge to derive the optimal push/pull strategy for subsequent data deliveries that are expected to minimize energy cost. Additionally, this PPE is dynamically optimized at runtime to adapt to changing sensor sampling requests from the cloud. For each sensor, the second algorithm (OPT-2) optimizes the configuration of the push/pull envelope between the edge and beneath layers. For example, given long-term learning of the data output by a particular sensor, assume that the sensor data changes at a relatively constant rate that is less than the required data push rate calculated by OPT-1. Here, the sensor can choose a reduced data push rate, which further reduces energy consumption.

### 4.1 Cloud–Edge PPE Optimization

This section describes a dynamic algorithm that, for a single sensor, finds an optimal constant base push rate $f^*$, given the relative characteristics of sensor sampling requests from the cloud. As we have observed, requests from the cloud usually change with time. However once the arrival rate of a request surpasses a prespecified level for a given time interval, we can use base push with a constant rate to satisfy a portion of the data requests. This statement holds because push is less costly than pull if (a) the pushed data is known to be adopted by its sink (i.e., applications in the cloud), and (b) we can employ supplemental pulls to meet the unsatisfied requests. Taking advantage of this optimization opportunity, the edge adjusts the rate of base pushes to effect an overall (base + supplemental) rate change in order to find the optimal $f^*$ that minimizes total energy cost. It is important to note that the edge continuously senses and analyzes sensor sampling requests from the cloud, to capture the relative characteristics of past requests that will support computation of $f^*$.

As the computation of $f^*$ is based on a history of sensor values, our optimization approach must be able to dynamically adapt to changes in requests issued by the cloud during a specific time interval. Therefore, the edge chooses an evaluation window within which sensor sampling requests are analyzed. Our algorithm implements a simple but self-adaptive approach that selects the length $W_e$ of this evaluation window. The following terms pertain to our discussion, in terms of a single sensor $s$:

| | |
|---|---|
| $R\ (t_i)$ | Total number of data acquisition and sampling requests for sensor $s$ received by the edge from the cloud, over time $t_i$ |
| $D$ | Sliding window at each end of which $R\ (t_i)$ is recorded. The length of $D$ is $d$, which implies that $d = t_i - t_{i-1}$. |
| $R'\ (t_i)$ | Average arrival rate of data acquisition and sampling requests from sensor $s$ received by the edge from the cloud, within the sliding window $D\ [t_i - d, t_i]$, as follows: |

$$R'(t_i) = \frac{R(t_i) - R(t_{i-1})}{d} \tag{1}$$

| | |
|---|---|
| $W_e$ | Evaluation window within which multiple sensor requests from the cloud are monitored and analyzed by the edge, to provide information for computing the next $f^*$. |
| $L$ | Length of evaluation window defined as the number of sliding windows within We , i.e., $L = \lvert We \rvert / d$. |

In summary, by analyzing the history of sampling requests from the cloud for a single senor $s$ within the evaluation window $W_e$, the edge finds the optimal base push rate $f^*$ that is expected to maximize energy savings for subsequent data queries. Therefore, the following three problems need to be solved: (1) determine the objective function and an algorithm to solve it; (2) determine when to re-evaluate $f^*$, and (3) specify the evaluation window $W_e$.

To solve Problem 1, we construct the objective function by using the energy saving rate of our mixed push-pull scheme (base push rate at $f^*$), to replace the pure pull scheme that is one of the customary approaches in current sensor data acquisition practice. The energy cost of a sensor node is modeled by considering both transmission and sampling [6] as major contributing factors for overall energy consumption. Accordingly, we define two energy cost coefficients: $\alpha$, the energy consumption factor for one-time transmission (either sending or receiving a packet); and $\beta$, the energy cost for one sensor sampling operation. Therefore, the cost of a pull operation is $2\alpha + \beta$ (receiving query + sending data + sampling), and a push operation costs $\alpha + \beta$ (neglecting the one-time subscription). Within evaluation window $W_e$, the energy cost of our mixed push-pull strategy (base push rate at $f^*$) is

$$C_{PPE} = (\alpha+\beta)\sum_{t_i\in T}(f^*\times d) + (2\alpha+\beta)\sum_{t_i\in c_1}(R'(t)-f^*)\times d \tag{2}$$
$$= (\alpha+\beta)\times\sum_{t_i\in(c_1\cup c_2)}(f^*\times d) + (2\alpha+\beta)\times\sum_{t_i\in c_1}(R'(t)-f^*)\times d$$

where $T$ represents all of the end points of sliding window $D$ within $W_e$; $c_1$ denotes the times $t_i$ at which $R'(t_i) > f^*$ (i.e. supplemental pull needed at sliding window $[t_{i-1}, t_i]$); and $c_2$ indicates the times $t_i$ at which $R'(t_i) \le f^*$. Observe that $T = c_1 \cup c_2$.

Additionally, the energy cost by using pure pull scheme to satisfy all of the data requests is given by

$$C_{ppull} = (2\alpha+\beta)\sum_{t_i\in T}(R'(t_i)\times d) = (2\alpha+\beta)\sum_{t_i\in(c_1\cup c_2)}(R'(t_i)\times d) \tag{3}$$

Therefore, the objective function (energy saving rate) is defined as:

$$\max R_{save}(f^*, W_e) = \frac{C_{ppull} - C_{PPE}}{C_{ppull}} \tag{4}$$
$$= \frac{\alpha\times f^*\times L - (2\alpha+\beta)\sum_{t_i\in c_2}(f^* - R'(t_i))}{(2\alpha+\beta)\sum_{t_i\in T}R'(t_i)}$$

where $R_{save}$ is determined by $f^*$ and the evaluation window $W_e$. The objective function is thus given by

$$\max R_{save}(f^*, W_e) = \frac{\alpha\times f^*\times L - (2\alpha+\beta)\times\frac{\sum_{t_i\in c_2}(f^*-R'(t_i))}{L}\times L}{(2\alpha+\beta)\times\frac{\sum_{t_i\in T}R'(t_i)}{L}\times L}$$
$$= \frac{\alpha\times f^* - (2\alpha+\beta)\times\frac{\sum_{t_i\in c_2}(f^*-R'(t_i))}{L}}{(2\alpha+\beta)\times\frac{\sum_{t_i\in T}R'(t_i)}{L}} \tag{5}$$

We define the average of $R'(t)$ within evaluation window $W_e$ as

$$P_r = \frac{\sum_{t_i\in T}R'(t_i)}{L} \tag{6}$$

which denotes the average arrival rate of sampling requests for $s$ received by the edge from the cloud within evaluation window $W_e$. We also define the average rate at which superfluous push operations happen within evaluation window $W_e$, as follows:

$$SPR(f^*) = \frac{\sum_{t_i\in c_2}(f^* - R'(t_i))}{L} \tag{7}$$

It is clear that the value of $SPR(f^*)$ changes with variations in $f^*$.

Therefore, we can reformulate the objective function as

$$\max R_{save}(f^*, W_e) = \frac{\alpha\times f^* - (2\alpha+\beta)\,SPR(f^*)}{(2\alpha+\beta)\,P_r} \tag{8}$$
$$= \frac{\alpha}{2\alpha+\beta}\times\frac{f^*}{P_r} - \frac{SPR(f^*)}{P_r}$$

In addition, the optimization algorithm must be able to adapt to the change of requests from the cloud (Problem 2). In our approach, after $f^*$ is determined for subsequent data queries, the edge will continue to monitor factors that may affect PPE performance. For example, two factors in our performance evaluation algorithm are $P_r$ and $SPR(f^*)$ (Equations 6 and 7). For either factor, once the difference between its current value and the value used to evaluate current $f^*$ exceeds a designated threshold $\varphi$, a new round of $f^*$ evaluation will be started. Simultaneously, the current PPE will be terminated as it might be inaccurate for processing current sampling requests.

We solve Problem 3 by initially assuming that the evaluation window $W_e$ starts at the time when the current $f^*$ is calculated, and ends at the time when $f^*$ is re-evaluated. This simple approach is effective, since the length of $W_e$ will adapt to the changing value of $R'(t_i)$. For example, if $R'(t_i)$ changes rapidly, then the performance of the current PPE will likely decrease rapidly. Therefore, the length of $W_e$ for the next evaluation of $f^*$ will be relatively shorter than it is when $R'(t_i)$ changes slowly. So the length of the evaluation window reflects how rapidly the evaluation of $f^*$ adapts to changing sensor requests from the cloud. The shorter the evaluation window, the faster the evaluation of $f^*$ adapts to the change of requests. For this reason, $W_e$ should be shorter when $R'(t_i)$ changes rapidly. Conversely, $W_e$ should be longer if $R'(t_i)$ changes slowly. Our experimental research results show that this approach supports the accurate tracking of changes in sensor request patterns that are typical of sensor demands by cloud applications analyzed thus far.

Given the preceding development, we summarize the proposed optimization algorithm (OPT-1), as follows.

**OPT-1: Optimizing PPE between Cloud and Edge**

```
Input: threshold $\varphi$; initial length of evaluation window $L$
* $t_c$: current system time
Algorithm:
1. Set evaluation window $W_e = [t_c - L \cdot d,\ t_c]$;
2. while true do
3.     Calculate optimal $f^* = FindFstar(R(t_c), W_e)$;
4.     Record $SPR' = SPR(f^*)$; $P_r' = P_r$;
5.     Set base push rate $= f^*$ and start mixed push pull mode;
6.     Set $t_1 = t_c$;
7.     $PerformCheck(f^*, SPR', P_r')$ //quit when $f^*$ needs re-evaluation
8.     Set $t_2 = t_c$; $W_e = [t_1, t_2]$;
9. end while
```

The supporting algorithm *FindFstar*, which optimizes $f^*$, is specified as follows.

**Method** *FindFstar*: **Find the Optimal *f** Using Binary Search**

**Input:** $R(t_i)$, $We$
**Output:** the base push rate $f^*$.

```
1.  Calculate $R'(t_i)$, for each $t_i$ within $We$.
2.  Find min $R'(t_i)$ and max $R'(t_i)$, for $t_i$ within $We$ .
3.  if $R_{save}$( min $R'(t_i)$, $We$ ) > $R_{save}$( min $R'(t_i)+\Delta f$, $We$) then
4.      return min $R'(t_i)$ as the optimal $f^*$;
5.  end if
6.  if $R_{save}$( max $R'(t_i)$, $We$) > $R_{save}$( max $R'(t_i)-\Delta f$, $We$ ) then
7.      return max $R'(t_i)$ as the optimal $f^*$;
8.  end if
9.  $f_L$ = min $R'(t_i)$, $f_R$ = max $R'(t_i)$ ;
10. $f' = (f_L + f_R)/2$
11. while $R_{save}(f' - \Delta f, We) > R_{save}(f', We)$ or
12.     $R_{save}(f' + \Delta f, We) > R_{save}(f', We)$ do
13.     if $R_{save}(f' - \Delta f, We) > R_{save}(f', We)$ then
14.         $f_R = f'$
15.     else $f_L = f'$
16.     end if
17.     $f' = (f_L + f_R)/2$
18. end while
19. return $f'$ as the optimal $f^*$
```

**Method** *PerformCheck:* **Monitoring Performance of *f***

**Input:** $f^*$, $SPR'$, $P_r'$
* $W_c$: time window within which runtime $SPR(f^*)$ and $P_r$ are calculated. It has fixed length, i.e., $|W_c|$.
* $\gamma_1$: threshold used for monitoring $SPR$ beyond which $f^*$ re-evaluation will be triggered.
* $\gamma_2$ : threshold used for monitoring $P_r$ beyond which $f^*$ re-evaluation will be triggered.

```
1.  ReEva=false; //Initialization
2.  while !ReEva do
3.      Calculate $SPR(f^*)$ and $P_r$ within current $W_c$
4.      if $|SPR(f^*) - SPR'| / SPR' > \gamma_1$ or
5.          $|P_r - P_r'| / P_r' > \gamma_2$ then
6.          ReEva=true;
7.      end if
8.      Wait $|W_c|$ time for the next time window $W_c$
9.  end
```

The following proof sketch demonstrates the rationality and correctness of the function *FindFstar*. In order to apply a binary search algorithm to our process of searching for an optimal value of $f^*$, the objective function max $R_{save}(f^*, W_e)$ must satisfy the following conditions:

i. The optimal $f^* \in$ [min $R'(t_i)$, max $R'(t_i)$] for $t_i \in W_e$.

ii. There is only one optimal $f^*$ for *max* $R_{save}$ $(f^*, W_e)$, s.t., min $R'(t_i) \le f \le$ max $R'(t_i)^*$.

iii. For min $R'(t_i) \le f < f^*$, $R_{save}(f, W_e)$ increases as $f$ increases; for $f^* < f \le$ min $R'(t_i)$, $R_{save}(f, W_e)$ decreases as $f$ increases.

Condition (i) is trivial. The proof of conditions (ii) and (iii) are as follows.

First, by calculating the derivative of (8) we get

$$\frac{d\,R_{save}(f^*)}{d\,f^*} = \frac{1}{P}.(\frac{\alpha}{2\alpha+\beta} - \frac{d\,SPR(f^*)}{d\,f^*}) \tag{9}$$

If we can prove (9) is a monotone decreasing function of $f^*$, then (ii) and (iii) will be guaranteed in that $R_{save}(f^*)$ can changes only in the following three ways as $f^*$ increases from min $R'(t_i)$ to max $R'(t_i)$: a) always positive, b) always negative, c) first positive and then negative.

First of all, assume that functions $f = R_{save}(t)$ maps $t_i \in W_e$ to $f_i \in$ [min $R'(t_i)$, max $R'(t_i)$] for all $i$ = 1..$n$. Sort $f_i$ in non-decreasing order, i.e.

$$\min R'(t_i) = f_0 \le f_1 \le \ldots \le f_n = \max R'(t_i) \tag{10}$$

Partition the interval [min $R'(t_i)$, max $R'(t_i)$] into disjoint union of half-open intervals $[f_i, f_{i+1})$ where $i$ = 0..n-1. For $f' \in [f_i, f_{i+1})$

$$SPR(f') = \frac{1}{L}\sum_{t=0}^{i}(f' - f_t) = \frac{1}{L}(i \times f' - S_i) \tag{11}$$

Where, $S_i = \sum_{t=0}^{i} f_t$

Hence,

$$\frac{d\,SPR(f')}{d\,f'} = \frac{i}{L} \tag{12}$$

For $f' \in (f_i, f_{i+1})$. Note that the derivative is not defined when $f' = f_i$, $i$ = 1..$n$. However, the side limits (lim − and lim +) of $SPR(f')$ agrees at those special points, $f' = f_i$, $i$ = 1..$n$-1. In other words, $SPR(f')$ is continuous within [min $R'(t_i)$, max $R'(t_i)$] and it is differentiable within [min $R'(t_i)$, max $R'(t_i)$] except at a finite set of points.

Assume that $f_a \in (f_{i_a}, f_{i_a+1})$ and $f_b \in (f_{i_b}, f_{i_b+1})$.

If $f_a < f_b$ , then $i_a \le i_b$ . Therefore, $SRP(f_a)' \le SRP(f_b)'$

So we can see that the derivative of $SPR(f^*)$ is a monotone increasing function of $f^*$ and hence the derivative of $R_{save}(f^*)$ is a monotone decreasing function of $f^*$ according to (9). Finally, we prove that condition (ii) and (iii) are satisfied.

## 4.2 Edge-Sensor Sampling Optimization

After the edge calculates the optimal base push rate $f^*$ for a particular sensor $s$ by running OPT-1, it will send $f^*$ to $s$ to set its data sampling rate. Fortunately, after receiving $f^*$, the sensor also has the chance to determine if it can choose an even lower sampling rate at which data is pushed to the cloud via the edge. The key concept is, if the data from $s$ changes at a rate much lower than the required sampling rate $f^*$, then a lower push rate might sufficiently reflect the change of sensor data. We call this mode *lazy sensor sampling*. For our specific scenario where applications follow an event-driven paradigm, the base push rate of a particular sensor $s$ should reflect the rate of change of the values of basic events associated with $s$. The goal of our lazy sampling algorithm (OPT-2) is to find the rate $f^{**}$ at which $s$ samples and pushes its data to the edge, using the lowest sampling rate to meet the cloud's demands for sensor data.

For a single sensor $s$, the following terms pertain:

| | |
|---|---|
| $V(t_i)$ | Number of changes (until time $t_i$) in the values of basic events that are associated with $s$. |
| $D_l$ | Sliding window at each end of which $V(t_i)$ is recorded. The length of $D_l$ is $d_l = t_i - t_{i-1}$. |
| $V'(t_i)$ | Average change rate of the value of basic events that are associated with $s$ within the sliding window $D_l$ $[t_i - d_l, t_i]$. Thus, $V'(t_i)$ is given by |

$$V'(t_i) = \frac{V(t_i) - V(t_{i-1})}{d_l} \tag{13}$$

Similarly, we obtain the acceleration in the basic event change rate associated with the *s*, which we denote as $V''(t_i)$, as follows:

$$V''(t_i) = \frac{V'(t_i) - V'(t_{i-1})}{d_l} \tag{14}$$

Let us now describe constraints on lazy sampling. An activation condition of lazy sampling guarantees that the sensor data changes at a relatively constant rate, such that our lazy sampling algorithm will not lose track. This condition can be expressed intuitively in terms of a threshold derived from $V''(t_i)$, as follows:

$$\sum_{i=c-k}^{c} |V''(t_i)| < \tau \tag{15}$$

where $t_c$ is the current time, $\tau$ is the threshold, and *k* is obtained experimentally.

If the above condition is satisfied, then our OPT-2 algorithm will compute an optimal or near-optimal rate $f^{**}$ at which *s* will sample and push its data to the edge. This near-optimal sampling rate is determined by summing the predicted $V'^{*}(t)$ at the next time slice, toleranced by a safe margin, to compensate for any abrupt change of sensor data. The estimation of $V'^{*}(t)$ at the next time slice should be based on the sensor data history, with provision for temporal tolerancing, as mentioned previously. We have found that the following method for determining $V'^{*}(t)$ is practicable:

$$\begin{aligned} V'^{*}(t) &= \gamma \times V'(t_c) + \gamma \times (\gamma - 1) \times V'(t_{c-1}) + \gamma \times (\gamma - 1)^2 \times V'(t_{c-2}) \\ &+ \ldots + \varepsilon \sum_{i=c-k}^{c} |V''(t_i)| = \gamma \times \sum_{i=0}^{k} (\gamma - 1)^i \times V'(t_{c-i}) + \varepsilon \sum_{i=c-k}^{c} |V''(t_i)| \end{aligned} \tag{16}$$

where $\gamma$ influences how rapidly $V'^{*}(t)$ adapts to changes of $V'(t_i)$, and $\varepsilon \sum_{i=c-k}^{c} |V''(t_i)|$ denotes the temporal tolerance (margin of safety).

We thus summarize our lazy sampling algorithm, as follows.

**OPT-2: Lazy Sampling Algorithm**

```
Input: $f^*$ calculated by edge through OPT-1, $V(t_i)$
Algorithm:
1. lazyMode=false; //Initialization
2. while true do
3.   if Condition (15) is satisfied and ! lazyMode then
4.     set $f^{**}$= FindLazyRate ($f^*$, $V(t_i)$);
5.     lazyMode=true;
6.   end if
7.   if Condition (15) is not satisfied and lazyMode then
8.     set $f^{**}=f^*$;
9.     lazyMode=false;
10.  end if
11.  wait for $k$ time.
12. end while
```

**Method *FindLazyRate*: Find the sensor sampling rate *f*****

```
Input: $f^*$, $V(t_i)$
Output: the base push rate $f^{**}$.
1. Calculate $V'^{*}(t)$
2. if $V'^{*}(t) < f^*$ then
3.     return $f^{**}= V'^{*}(t)$ // lazy sampling mode starts
4. else return $f^{**}=f^*$;
5. end if
```

Importantly, to enable our optimization, caching as a supporting technology must be implemented in our three layer model.

### 4.3 Supporting Technology: Caching

Some events are tolerant to time fidelity in that their respective applications do not require frequent evaluation at all times. For example, when monitoring room temperature, an air conditioning system does not need to get temperature reading at a very high frequency, since the changing rate of room temperature is usually slow. In addition, an air conditioning system may need to monitor room temperature only occasionally during the spring and fall. Considering these scenarios, data of a particular sensor can be considered fresh within a certain "lifetime" after it is sampled. Taking advantage of this fact, we can employ data caching mechanisms so that historical sensor data could be reused to satisfy different requests for a common sensor data, thereby reducing energy consumption. In our model, caching are employed in all three layers (cloud, edge, and sensors) to realize energy saving.

Caching data in the cloud [19] allows different applications in the cloud (possibly located at different cloud servers) to reuse sensor data without requesting sensor data from the lower layers. Once a request is issued by an application, the cache is checked for "freshness". If it is, data will be used directly. Therefore, only those requests that cannot be satisfied by cached sensor data will be forwarded to the edges that will fetch data on behalf of applications. Similarly, caching is implemented at the edge and even sensors to further reduce energy consumption by enabling historical data reuse whenever possible.

## 5. Experimental Evaluation

### 5.1 Experimental Setup

We evaluate the performance of CEB under OPT-1, OPT-2 and their combination, by comparing their performance with that of CEB under a pure pull mechanism. We use an emulation approach where the requests from the cloud and data from the beneath are simulated by generators and in which edges and sensor nodes are simulated by software emulators. As input to our optimization algorithms, the arrival of sensor sampling requests and the generation of sensor data are simulated in the following way.

Sensor sampling request generator emulates the arrivals of requests from applications in the cloud asking for data from sensors. We simulate total number of *K* applications and total number of *S* sensors. In our experiment, we choose {*K*=50, *S*=50; *K*=100, *S*=50; *K*=500, *S*=100}. Sensors are randomly assigned to each application following a normal distribution ( $N$ ($\mu$=2, $\sigma^2$=4)). Each application requires data from an arbitrary set of sensors out of the sensors assigned to it for execution. The time between two active phases of an application follows an exponential distribution with average arrival rate λ=1. For the generation of sensor data, we use the normal distribution. However, parameters of $\mu$ and $\sigma^2$ are varied during the course of each experiment. High $\sigma^2$ emulates the phase when the data of a sensor is changing rapidly while lower $\sigma^2$ emulates the phase when sensor data is changing slowly.

We emulated the effect of three sets of sensor platforms [Table 1] by normalizing their energy consumption coefficients. We conduct experiments on each set of coefficients to quantify the platform's receptiveness to our optimizations. Transmission energy cost includes two parts, cost of transmission between edge and sensor and cost of transmission between cloud and edge. We assume that for all three sensor platforms transmission cost between edge and cloud is unchanged.

**Table 1. Normalized energy coefficients for transmission and sampling on three different sensor platforms. S1: MICA2 sensor platform with Sensirion Humidity sensor, ChipCon CC1000 Radio [18], S2: Atlas sensor platoform with Interlink Pressure Sensor, Atmel ZLink Radio [6] and S3: MicroLEAP with ECG sensor, class-2 Bluetooth 2.0 radio [20]. $\alpha_2$ is derived from [21] (assume average 10 hops from edge to cloud).**

| Sensor platform | $\beta$: sampling | $\alpha_1$ transmission between edge and sensor | $\alpha_2$ transmission between cloud and edge |
|---|---|---|---|
| S1 | 0.21 | 0.36 | 0.43 |
| S2 | 0.88 | 0.03 | 0.09 |
| S3 | 0.26 | 0.33 | 0.41 |

## 5.2 Evaluation Metrics

The main performance metric, which is measured throughout all experiments, is the energy saving rate of our mixed push-pull scheme to replace the pure pull scheme i.e. $R_{save}$. The actual energy saving rate in all experiments is given by (19).

$$C_{ppull} = (2\alpha_1 + 2\alpha_2 + \beta) \times m_{pull} \quad (17)$$

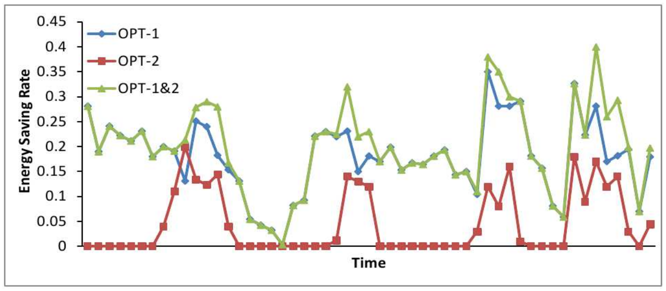
$$C_{ppe} = 2\alpha_2 \times n_{pull1} + \alpha_2 \times n_{push1} + (2\alpha_1 + \beta) \times n_{pull2} + (\alpha_1 + \beta) \times n_{push2} \quad (18)$$

$$R_{save} = 1 - \frac{C_{ppe}}{C_{ppull}} \quad (19)$$

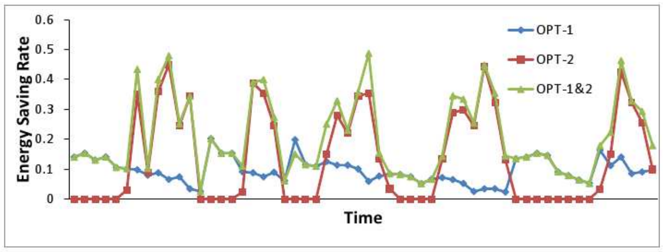
(17) indicates the total energy cost by using pure pull scheme in which $m_{pull}$ means the total number of pulls (cloud pulls data from sensors) in pure pull scheme. (18) represents the total energy cost by using our mixed push pull scheme. $n_{pull1}$ and $n_{push1}$ means the total number of pulls and pushes respectively between edges and the cloud in our mixed push pull scheme; while $n_{pull2}$ and $n_{push2}$ means respectively the total number of pulls and pushes between edges and sensors in our mixed push pull scheme.

## 5.3 Performance Comparison Results

We compare the performance of OPT-1, OPT-2 and OPT-1 and OPT-2 combined, for sensor platforms S1 and S2. The result is recorded every $T$ time (5 sec) as shown in Figure 5.1. ($K$=500, $S$=100).

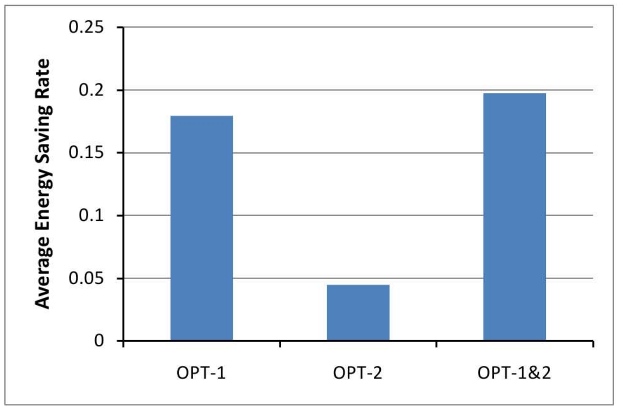


(a) Result for sensor platform S1

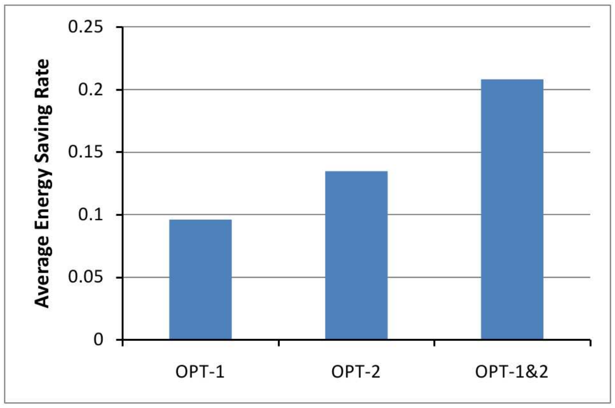


(b) Result for sensor platform S2

**Figure 5.1. Energy saving rate for three optimization techniques on sensor platform S1 and S2**



(a) Results for sensor platform S1



(b) Results for sensor platform S2

**Figure 5.2. Average energy saving rate for three algorithms on sensor platform S1 and S2 (over 255 seconds)**

From Figure 5.1. and 5.2., we can see that for sensor platform S1 whose transmission energy cost accounts for most of the energy consumption, OPT-1 shows better performance than that for sensor platform S2 where sensor sampling accounts for most of the energy consumption. This can be explained by the motivation of OPT-1, in which by replacing as many pulls with pushes the number of transmission can be reduced, leading to energy saving. On the contrary, OPT-2 shows better performance in S2 than in S1. The reason is that OPT-2 reduces the number of push as well as the number of data sampling when data of a given sensor does not change rapidly during a particular period. In Figure 5.2, we show that the combined application of OPT-1 and OPT-2 gives a superior performance than by utilizing OPT-1 or OPT-2 alone.

In addition, we measure the effects of optimization on three sets of sensor platforms and compare the results in Figure 5.3. From this result, we see that S1 adopts more pushes than S2 while it adopts less pull than S3. The reason is that in S2 supplemental pull has fewer penalties than in S1 and hence pull will be preferred as its on-demand feature.

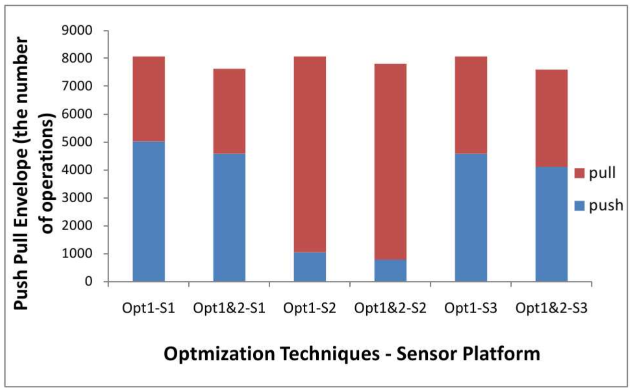


**Figure 5.3. The push-pull envelope on different sensor platforms (S1-S3)**

## 6. Related Work

Integrating sensor networks with the emerging data center cloud model of computing is becoming a popular paradigm of choice in system development for various application domains including health care [1], warehouse monitoring [4] and environmental monitoring [3]. In order to realize this integration, in [4] and [5] similar architectures are proposed to utilize virtualization as supporting technology for integrating physical sensors as services into the cloud. These architectures, however, lack an intermediate layer between the cloud and various physical devices, which raises serious scalability issues. In order to enable scalable collaborative sensor-centric applications in the cloud, [2] and [1] provided their own frameworks which respectively use Pub/Sub broker and exchange service as intermediate layer between the cloud and device layers, to preserve locality of control. However, these approaches disregard energy consumption, which could significantly affect system performance. In our work presented herein, we introduce, justify, and take advantage of a three-layer model (cloud, edge, and beneath) to enhance scalability as well as increase energy efficiency via algorithms for optimizing the push/pull envelope.

Additionally, in wireless sensor networks, special data acquisition techniques have been developed for event detection supporting real-time application execution. A typical scheme is polling [11], in which a data sink sequentially polls its underlying sensors for new data. In contrast, a bottom-up sensor-driven model [12] has also been proposed, assuming that sensors are capable of pushing data to applications when an event occurs. To improve the efficiency of data delivery and enable data sharing, messaging paradigms such as publish/subscribe [9] and push-pull [10] have been widely adopted in sensor data acquisition. Optimization techniques to balance push and pull have been extensively discussed in [10][13][14], which focus on network topology and routing algorithms. Furthermore, a new model discussed in [15] utilizes the mixed push/pull strategy and takes advantage of the optimization opportunity provided by the event structure and its time-frequency relaxations. However, none of the above approaches fit perfectly into a cloud-based sensor computing paradigm. This is especially true when one considers the massiveness of sensors and applications that tend to be invisible to each other. In order to overcome this problem, we have developed a push pull strategy based on the relative characteristics of sensor requests (demand side from the cloud) and sensor data (supply side from beneath). These two features can be easily captured in our model, and support our claims of effectiveness and significance in promoting our approach's energy efficiency.

## 7. Conclusion

We adopt the Cloud, Edge and Beneath (CEB) architecture to sensor data access by Cloud applications. CEB more naturally represents actual sensor system deployments in which an edge device (e.g., gateways) manages a locale of sensors. CEB allows for many optimization opportunities. In this paper, we present two optimizations. OPT-1 optimizes the proper mix of base push and supplemental pull between the cloud and the edge. OPT-2 optimizes the sampling and delivery of sensor data between the sensors and the edge, based on sensor domain value characteristics. We present both algorithms along with an experimental evaluation of their individual and combined impact. The results shows that by combining both algorithms, CEB can always achieve better performance than adopting any of the algorithms individually.

## 8. REFERENCES

[1] C. Rolim, F. Koch, C. Westphall, J. Werner, A. Fracalossi, G. Salvador, "A Cloud Computing Solution for Patient Data Collection in Health Care Institutions," in Proc. of ETELEMED. IEEE, 2010, pp. 95–99

[2] M. Hassan, E. Huh. "A Framework of Sensor-Cloud Integration: Opportunities and Challenges". International Conference on Ubiquitous Information Management and Communication.

[3] X.H. Le, S. Lee, T. Phan, V. La, A. Khattak, M. Han, H. Dang, M. Hassan, M. Kim, K. Koo, Y.K. Lee, E.N. Huh, "Secured WSN-integrated cloud computing for u-Life care", in: Seventh Annual IEEE Consumer Communication and Networking Conference (CCNC), January 9–12, Las Vegas, 2010, pp. 1–2.

[4] K Lee and D. Hughes. "System architecture directions for tangible cloud computing." In International Workshop on Information Security and Applications (IWISA 2010), in Qinhuangdao, China, October 22- 25, 2010.

[5] M. Yuriyama and T. Kushida, "Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing", The 13th International Conference on Network-Based Information Systems (NBiS-2010), 2010

[6] R. Bose and A. Helal, "Sensor-aware Adaptive Push-Pull Query Processing in Wireless Sensor Networks," Submitted to the 6th International Conference on Intelligent

Environments - IE'10, Kuala Lumpur, Malaysia, July 19-22, 2010.

[7] Y. Kawahara, M. Minami, H. Morikawa, T. Aoyama: "Design and Implementation of a Sensor Network Node for Ubiquitous Computing Environment", In Proc. Of VTC2003-Fall 2003.

[8] A. Helal, D. Cook, M. Schmalz, "Smart Home-based Health Platform for Behavioral Monitoring and Alteration of Diabetes Patients," Journal of Diabetes Science and Technology, Volume 3, Number 1, January 2009. Pp 141-148.

[9] N. Rosa, C.Ferraz, J. Kelner, E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira. "Mires: a publish/subscribe middleware for sensor networks". Personal and Ubiquitous Computing , Volume 10 Issue 1, December 2005. Springer-Verlag.

[10] X. Liu, Q. Huang, Y. Zhang. "Balancing Push and Pull for Efficient Information Discovery in Large-Scale Sensor Networks". IEEE Transactions on Mobile Computing, Volume 6, Issue 3, March 2007 Page(s):241 – 251.

[11] Z. Zhang, M. Ma, Y. Yang. "Energy Efficient Multi-Hop Polling in Clusters of Two-Layered Heterogeneous Sensor Networks". IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), April 2005.

[12] S. Reilly and M. Haahr. "Extending the Event-based programming model to support Sensor-Driven Ubiquitous Computing Applications". Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications (Percom'09).

[13] Z. Tao, Z. Gong, Z. OuYang, J. Xu. "Two New Push-Pull Balanced Data Dissemination Algorithms for Any-Type Queries in Large-Scale Wireless Sensor Networks". International Symposium on Parallel Architectures, Algorithms, and Networks, 2008. 7-9 May 2008 pp: 111 – 117.

[14] S. A. Hashish, A. Karmouch. "Topology-based on-board data dissemination approach for sensor network." Proceedings of the 5th ACM international workshop on Mobility management and wireless access. Chania, Crete Island, Greece, 2007. pp. 33 – 41.

[15] C. Chen, Y. Xu, K. Li, and A. Helal, "Reactive Programming Optimizations in Pervasive Computing," in proceedings of 10th Annual International Symposium on Applications and the Internet, Seoul, Korea.

[16] M. Khanafer, M. Cuennoun, H. T. Mouftah, "WSN Architectures for Intelligent Transportation Systems" Proceedings of 3rd New Technologies, Mobility and Security (NTMS), Cairo, Egypt

[17] J. King, R. Bose, H. Yang, S. Pickles and A. Helal, "Atlas – A Service-Oriented Sensor Platform, " Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006). Tampa, Florida, November 2006.

[18] S. Lai, J. Cao, Y. Zheng, "PSWare: a Publish/Subscribe Middleware Supporting Composite Event in Wireless Sensor Network". IEEE International Conference on Pervasive Computing and Communications, 2009. pp.1-6.

[19] Y. Cardenas, J. M. Pierson, L. Brunie: "Uniform Distributed Cache Service for Grid Computing." In: Proceedings of the International Workshop on Database and Expert Systems Applications, pp. 351–355 (2005)

[20] L.K. Au, W.H. Wu, M.A. Batalin, D.H. McIntire and W.J. Kaiser, "MicroLEAP: Energy-aware Wireless Sensor Platform for Biomedical Sensing Applications". IEEE BIOCAS2007. November 2007. pp.158-162.

[21] V. Sivaraman, A. Vishwanath, Z. Zhao, C. Russell, "Profiling Per-Packet and Per-Byte Energy Consumption in the NetFPGA Gigabit Router," IEEE INFOCOM 2011 Workshop on Green Communications and Networking