FTR: Performance-Aware and Energy-Efficient Communication Protocol for Integrating Sensor Networks into the Internet

Sinung Suakanto[1], Suhono H. Supangkat[2], Suhardi[2] & Roberd Saragih[3]

[1]School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Jalan Ganesa 10, Bandung 40132, Indonesia
[2]Information Technology Research Group, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Jalan Ganesa 10, Bandung 40132, Indonesia
[3]Industrial and Financial Mathematics Research Group, Faculty of Mathematics & Natural Sciences, Institut Teknologi Bandung, Jalan Ganesa 10, Bandung 40132, Indonesia
Email: mr.sinung.suakanto@gmail.com

Abstract. Integrating sensor networks into the Internet brings many advantages. For example, users can monitor or control the state of the sensors remotely without visiting the field. Some researchers have proposed methods using a REST-based web service or HTTP to establish communication between sensors and server via the Internet. Unfortunately, as we know, HTTP is a best-effort service. In some cases this means that if the number of sensors increases the end-to-end Quality of Service will decrease. The end-to-end network delay increases, as well as the failure rate of data sending caused by HTTP timeouts. In this paper, we propose Finite Time Response (FTR) HTTP as a communication protocol suitable for integrating sensor networks into the Internet. We have defined a cross-layer approach that coordinates between the application layer and the physical layer to control not only performance but also energy efficiency. The HTTP request-response delay measured at the application layer is used as the decision factor at the physical layer to control the active and sleep periods. We also propose a forced-sleep period as a control mechanism to guarantee average performance for all nodes. The experimental results have shown that FTR has the ability to maintain better performance, indicated by a lower average response time and a lower average timeout experience. Optimization is still needed to gain better performance and better energy efficiency while also considering the average value of the update time.

Keywords: *communication protocol; energy efficiency; internet; quality of service; sensor network.*

1 Introduction

Sensor networks are defined as a collection of sensors that have their own functionality and can be coordinated to perform certain tasks, such as environmental [1] and disaster monitoring [2]. For certain tasks the state of the

Received February 26th, 2013, Revised August 5th, 2013, Accepted for publication August 29th, 2013.
Copyright © 2013 Published by ITB Journal Publisher, ISSN: 2337-5787, DOI: 10.5614/itbj.ict.res.appl.2013.7.1.5

sensors needs to be monitored or controlled remotely. In these cases, the sensor network must report its state or the states of the sensors to the data center or server. For this purpose the sensor network needs to be integrated into an external network such as the Internet. Sensor networks based on TCP/IP have been proposed to accommodate this.

Sensor networks based on TCP/IP have the advantage of being able to communicate directly with an existing infrastructure either of a wired IP network or of IP-based wireless technology such as GPRS [1]-[3]. Using TCP/IP partially or fully for sensor networks has many advantages over interconnecting through a separate IP device. The ubiquity of TCP/IP and the Internet is the reason they are commonly used as a basis for networking.

Many researchers have proposed communication protocols to integrate sensor networks into the Internet with HTTP or a web service using REST. Some have focused on REST protocols, such as Tiny-REST [3] and Constrained Application Protocol (CoAP) [4],[5]. REST architectures allow implementing IoT (Internet of Things) and Machine-to-Machine (M2M) applications to be developed on top of the web service, which can be shared and reused [5]. The states of the sensors become abstract resources identified by Unique Resource Identifiers (URIs), represented with arbitrary formats, and can be manipulated with HTTP methods such as GET, PUT, POST and DELETE [5]. As a consequence REST architecture for sensor networks drastically reduces the application development complexity [5].

The main problem that has to be addressed when integrating wireless sensors into the Internet is decreasing performance caused by the increased number of nodes. Integration into the Internet or a public network is challenging because you have to deal with a network that mostly is not under your own administrative control. Also, Internet protocol communication using HTTP running on TCP/IP is categorized as best-effort service. This means that all nodes compete to send data. In a previous work we have already shown that increasing the number of nodes or users sending data to the server (cloud server) can decrease the average performance of all users [6].

In this paper, we propose the Finite Time Response (FTR) protocol to overcome this problem. Our approach tries to integrate network sensors into a server via Internet communication using HTTP. Our approach also considers energy-efficiency, which is the biggest concern in sensor network deployment. We will compare FTR with conventional methods using REST, such as Tiny-REST [3]. We will also compare it with a simple approach using energy efficiency only (energy efficiency-aware).

We have developed a new communication protocol mechanism that is aware of both energy and performance. Our approach is a cross-layer approach because it has a mechanism to coordinate between the application layer and the physical layer.

This paper is structured as follows. Section 2 describes some previous work related to research about the integration of sensor networks into the Internet. Section 3 presents the Finite Time Response (FTR) protocol and a simple verification test. Section 4 describes our experimental setup for validation of our method. Section 5 describes the simulation results and a discussion of these results. Section 6 contains our conclusions.

2 Related Work

Research related to the integration of sensor networks into the Internet has been done using different approaches [1]-[5]. Some authors have proposed to integrate sensor networks into cloud computing [7]-[10]. The purposes for deploying sensor networks are various, such as environmental monitoring [1], disaster monitoring [2],[11], patient monitoring [9], and supply-chain management [9]. Some researches related to sensor networks focus on energy-efficiency [12],[13]. They propose active and sleep periods at idle state. The duration of the active and sleep periods can be adjusted adaptively based on external conditions [13]. Some have also considered reliability or traffic conditions [14]-[16]. Almost all of them modified the system on the MAC layer and the physical layer based on the OSI layer [12],[13]. There is also one cross-layer approach, which uses coordination between the application, the network and the MAC layer [15].

Transport protocol has been proposed as a real-time communication protocol. Transport protocol in IP-based sensor networks is commonly employed using the two main protocols in the TCP/IP stack: best-effort UDP and reliable byte-stream protocol TCP [14]. The main problem of TCP is that it exhibits poor performance in wireless environments, both in terms of throughput and energy efficiency [14]. Dunkels, *et al*. have proposed TCP Segment Caching to improve TCP performance significantly [14]. UDP can be used for sensor data that do not use unicast reliable byte-stream transmission [14]. TCP should be used for administrative tasks that require reliability and compatibility with existing application protocols, for example HTTP or REST.

REST has been employed as a communication protocol between wireless sensor networks and the Internet [3],[4]. This has some advantages because it applies uniform interfaces, self-descriptive data, stateless communication, cache components, and code-on-demand constraints on a simple client server architecture [3]. Its architectural style is suitable for distributed network applications and emphasizes scalability, generality of interfaces, independent/self-organized deployment of components and intermediary components [3]. Thomas, *et al*. have employed Tiny-REST as a simple communication protocol between sensor objects and the Internet using the HTTP methods GET and POST [3]. Tiny-REST operates at the application layer but it can also be applied at the sensor-based on TCP/IP.

3 The FTR Protocol

In this section we describe the FTR protocol as a novel protocol for integrating sensor networks into the Internet.

3.1 Basic Foundation

There are three basic mechanisms for sending data from the sensors to the server (data center): *push-based*, *pull-based* and *hybrid* [17]. In this research we have focused on a push-based system where the sensors must periodically report their state to the server. The sample of usage of this kind of system can be used for environment monitoring or disaster monitoring, which need periodical data reading and reporting from the sensors [11]. The user needs to receive periodic data collections concerning environmental conditions such as air pollution and temperature in order to have information about the state of the environment at all times. In other cases the user needs periodical information such as the river water level to predict or anticipate floods [11].

Our push-based system with periodical data sending is depicted in Figure 1. We define the duty-cycle period (ΔT) as the interval time for periodically sending data from node to server. In practical implementation the duty-cycle period can be adjusted depending on the need. If the state of the sensors needs to be monitored more precisely a shorter duty-cycle period can be used.

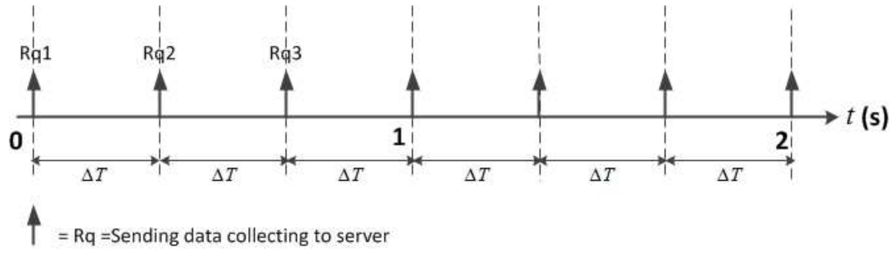

Figure 1 Push-based system with periodic data sending.

The communication between the nodes as client and the server is depicted in Figure 2. We define the response time (Δr_i) as the time delay measured from the sender starting to send a message to the server (request) until the sender receiving a reply (response) from the server. Here, we know that the response time consists of the delay from client to server: the processing delay at the server and the response delay from the server to the client related to the requested message.

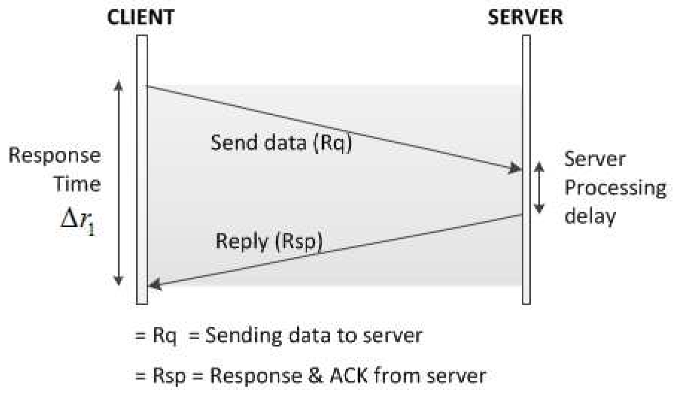
CLIENT
SERVER
Response Time Δr_1
Send data (Rq)
Server Processing delay
Reply (Rsp)
= Rq = Sending data to server
= Rsp = Response & ACK from server

Figure 2 Communications from client to server.

In HTTP communication the delay can vary depending on circumstances such as network traffic congestion, server performance, and many more. In some cases the client can have a good response time (short response time), in other cases the client can have a bad response time (long response time). In some cases the client has to wait too long for a response from the server, in which case the conversation will be terminated. This is called an HTTP timeout. Timeout is defined by the maximum time the client has to wait until he receives a response. If the defined period expires, the last conversation will be suspended or stopped.

3.2 Energy-Efficient Method

In this section we propose a simple method to maintain energy efficiency. We propose using an active period and a sleep period. The basis of this method was introduced with the S-MAC protocol [12]. The S-MAC protocol proposes a low duty cycle (short active period) to decrease power consumption especially during the node's idle state [12]. However, we propose a different mechanism to tell each node when to enter the active state or the sleep state.

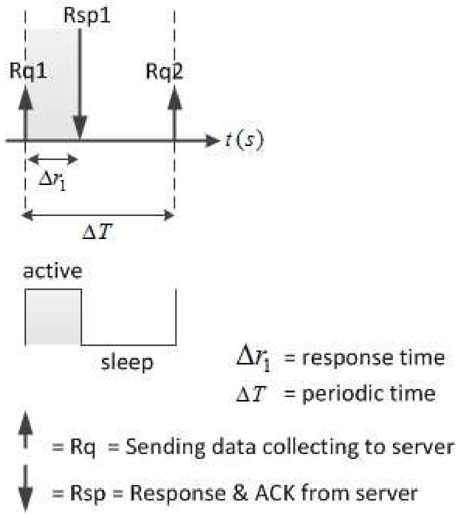
Rsp1
Rq1
Rq2
$t(s)$
Δr_1
ΔT
active
sleep
Δr_1 = response time
ΔT = periodic time
= Rq = Sending data collecting to server
= Rsp = Response & ACK from server

Figure 3 Basic mechanisms for controlling active and sleep period.

The basic mechanism for the sleep period and the active period is based on the length of the current response time, as depicted in Figure 3. The response time may vary depending on many things, as mentioned before. The period of waiting for data response is defined as active period and the rest as sleep period. Based on this approach, we can formulate a simple model to calculate the energy efficiency, as follows:

$$E = \frac{t_a}{t_a + t_s} = \frac{t}{\Delta T} \tag{1}$$

E	=	Energy efficiency (%)
t_a	=	Active period (in seconds)
t_s	=	Sleep period (in seconds)
ΔT	=	Duty cycle period / (in seconds)

From this formulation we can see that a low percentage value means better energy efficiency and, conversely, that a high percentage value means worse energy efficiency. Of course, 100% is the maximal value and 0% is the minimal value. We don’t recommend a value of 0% for the system because this means that the nodes do not send any data to the server all of the time. In the next section we will discuss what the optimal value is for energy efficiency.

From a hardware perspective, when the node enters the active state it will use this period to read data from the sensor and use radio communication to transmit and wait for a response. Conversely, in the sleep state the node can enter standby mode and turn off radio communication and/or turn off data reading from the sensors. During a sleep period we expect the node to have reduced power consumption.

In some cases, an active period based on the waiting period for response can create a new problem. During a busy period, the node not receiving a response until the next duty cycle will cause an HTTP timeout. This can create a very long active period and decrease energy efficiency drastically. To anticipate this problem we propose the Finite Time Response (FTR) mechanism, which will be discussed in the next section.

3.3 QoS-Aware Method

As mentioned in the previous section, our basic energy-efficient method has problems dealing with busy periods. To overcome this issue, we propose a mechanism to control average performance: the Finite Time Response (FTR) method. Basically, FTR means forcing the node to sleep when it detects poor performance indicated by a long response time for the last session. To indicate that the last session has poor performance, we use a finite parameter defined by the maximum response time allowed ($MaxTR$). Our method introduces a mechanism that limits the response time allowed by the system. We can formulate this as follows:

$$P_{i+1} = \begin{cases} 1 & if\,(\Delta r_i < MaxTR) \\ 0 & if\,(\Delta r_i \geq MaxTR) \end{cases} \tag{2}$$

P_{i+1}	=	Next mode at next duty cycle period (*i+1*) It has only 2 values: 0 or 1. • 0 indicates that during the next duty cycle period the node must enter sleep mode *(forced sleep*). This means that the node isn't allowed to send data for some period of time. • 1 indicates that during the next duty cycle period the node is allowed to transmit data to the server.
Δr_i	=	Response time during duty cycle period-*i* (seconds)
MaxTR	=	Parameter of maximum response time allowed (seconds)

The duration of forced sleep is defined as:

$$T^{FS} = K.\Delta T \tag{3}$$

T^{FS}	=	Duration of forced-sleep period (%)
K	=	Constant multiplier to indicate duration of forced sleep
ΔT	=	Duration for single duty cycle time (in seconds)

In the previous mechanism (energy-efficient method) there are only two node states: active and sleep. FTR proposes a new state, called *forced sleep*. Forced sleep is a condition similar to the sleep state but it is driven by a decision from equation (2). For K=1 it is full sleep during one single duty cycle period, if we set K=2 it is full sleep for twice one duty cycle, and so on. Our three states and their transitions are depicted in Figure 4. The sleep state is condition after the node receives a response from the server. The duration of the sleep period is until the next duty cycle period only.

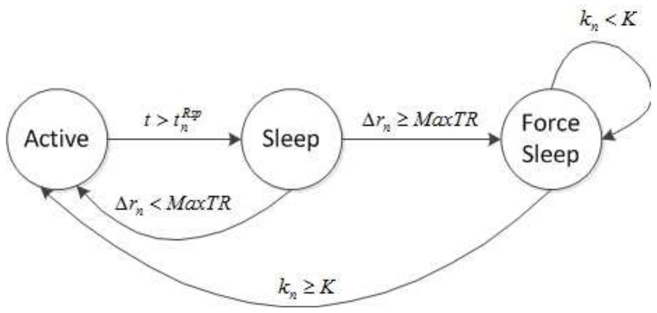

Figure 4 Finite-state model for QoS-aware method.

This method can be described with an algorithm that is implemented at each node, as shown in Figure 5.

Decision Making Algorithm at Next Duty Cycle

```
   procedure decision_next_step (last_response_time)
1. if (last_response_time >= maxTr) then
2.         force_sleep
3. else
4.         read_and_send_data
5. end if
```

Figure 5 Algorithm for next step decision using Finite Time Response.

Our approach is a cross-layer approach because it uses coordination between the application layer and the physical layer at the OSI layer, as shown in Figure 6. At the application layer we can measure or retrieve the response time (last performance) and make a decision to control the transmission of the next data. On the basis of the last performance, the node will decide to send data or not at the next duty cycle. This method is called *controlled message passing*, which means that the node can control message passing at the next duty cycle. Events occurring at the application layer will stimulate coordination with the duty cycle manager at the physical layer to control active, sleep or forced-sleep state. Events occurring at the application layer include data transmission events and response receiving events.

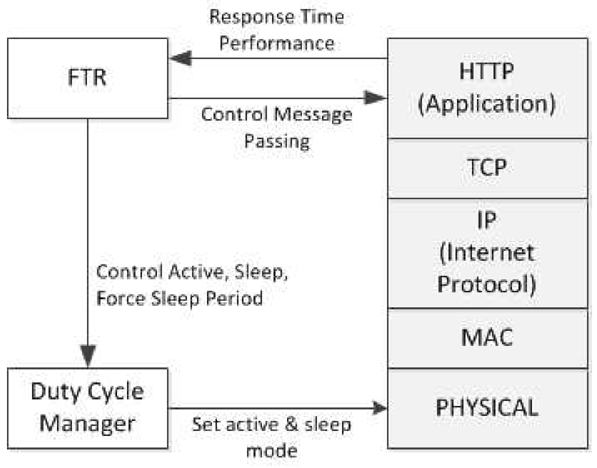

Figure 6 Cross-layer approach between application layer and physical layer.

Note that the average response time at each node is defined as follows:

$$R_m = \frac{\sum_{i=1}^{N} \Delta r_i}{N} \tag{4}$$

R_m	=	Average response time for node-*m* (in seconds)
Δr_i	=	Actual response time at duty cycle-*I* (in seconds)
N	=	Number of duty cycle periods reflecting frequency of data sending to server

Furthermore, we may formulate the average energy efficiency at each node as follows:

$$E_m = \frac{\sum_{n=1}^{N} \frac{T_n^A}{T_n^A + T_n^S}}{N} = \frac{\sum_{n=1}^{N} \frac{T_n^A}{\Delta T}}{N} \tag{5}$$

E	=	Average energy efficiency for node-*m* (%)
T_n^A	=	Duration of active period during duty cycle-*n* (in seconds)
T_n^S	=	Duration of sleep period (including forced sleep) during duty cycle-*n* (in seconds)
ΔT	=	Duration for single duty cycle time (in seconds)
N	=	Number of duty cycle periods reflecting frequency of data sending to server

3.4 Verification of System Performance

Before carrying out our experiment we executed a verification to check how our new method performs. This verification will be described briefly in this section. We describe the system's performance at a single node for several duty cycles. For verification we focused on how our proposed method succeeds in resolving the energy-efficiency problem. For traffic conditions we distinguished two classes of traffic:

1. *Busy traffic*: Indicated by a long response time but not resulting in HTTP timeout
2. *Very-busy traffic*: Indicated by a long response time and even leading to HTTP timeout.

First, we present how the system works under busy traffic conditions for six duty cycle samples, as depicted in Figure 7. We compare three scenarios: (i) Energy-Efficient Only Method, (iia) FTR with K=1, and (iib) FTR with K=2.

From Figure 7, we can summarize the active and sleep periods (including forced sleep) as depicted at Figure 8. We can also calculate the average energy efficiency, as summarized in Table 1.

Table 1 shows that our proposed method can maintain energy efficiency better than the Energy Efficient Method Only. We believe that increasing parameter K results in better energy efficiency. Therefore, if we want good energy efficiency we can set a very high value for K. However, the system must actually make a trade-off with other aspects, such as average response time and average update time.

Now, we define ΔT^U as the average update time. This is reflected in the speed of data updating from the sensor to the server. If the average update time is a low value it means that the resolution for data capturing is very high and has a very high precision. If it is set at a high value it means that the data resolution has a low rate and doesn't have a high precision. Actually, users can set the tolerance of the average update time based on their own needs.

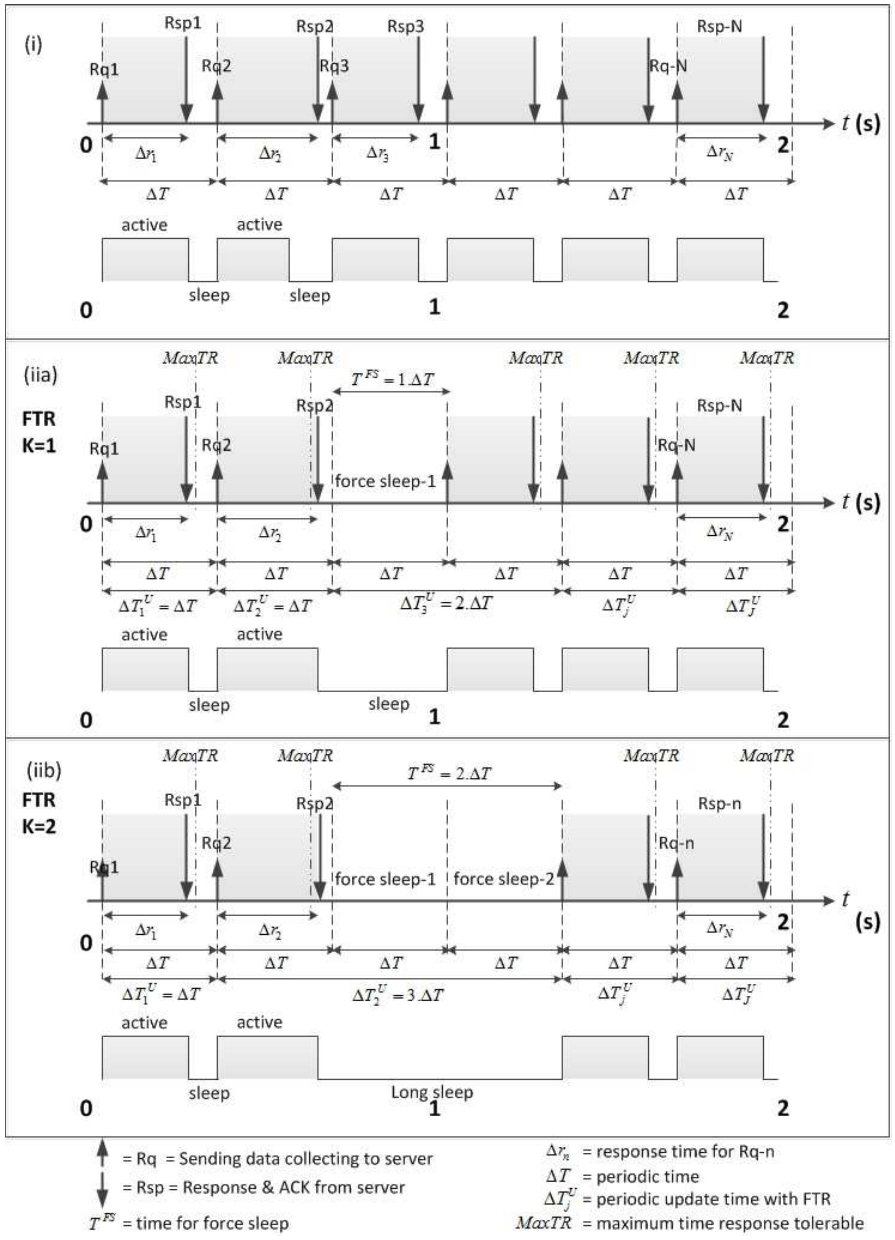

Figure 7 Comparing system performance for busy traffic conditions in three different scenarios: (i) Energy-Efficient Method Only, (iia) FTR with K=1, and (iib) FTR with K=2.

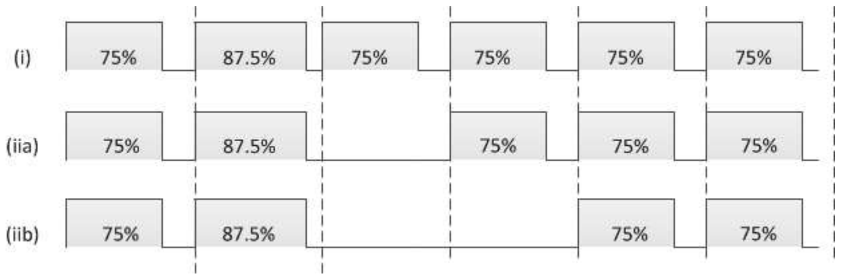

Figure 8 Summaries of active and sleep periods for busy traffic conditions.

Table 1 Result of comparing three scenarios for busy traffic conditions.

No	**Case**	**Average Energy Efficiency (%)**
(i)	Energy-Efficient Only Method	77,1
(iia)	FTR with K=1	64,6
(iib)	FTR with K=2	52,1

From Figure 1 we can calculate the average update time as follows:

For K=1: $$\Delta T^{U} = \frac{\Delta T + 2.\Delta T + \Delta T + \Delta T + \Delta T}{5} = 1{,}2\Delta T$$

For K=2: $$\Delta T^{U} = \frac{\Delta T + 3.\Delta T + \Delta T + \Delta T}{4} = 1{,}5\Delta T$$

For Energy-Efficient Method Only: $\Delta T^{U} = \Delta T$

Furthermore, we can compare the three scenarios in terms of average energy efficiency and average update time, as summarized in Table 2. We can see from Table 2 that when is K increased, the average energy efficiency improves. But we can also see that if K is increased, the average update time also increases. However, sometimes a higher value of the average update time is not acceptable for the user. In such cases we must carefully select an appropriate value for parameter K in order to get the optimal value between average energy efficiency and average update time.

Table 2 Result of comparing three scenarios with average energy efficiency and average update time for busy traffic conditions.

No	**Case**	**Average Energy Efficiency (%)**	**Average Update Time**
(i)	Energy-Efficient Method Only	77,1	ΔT
(iia)	FTR with K=1	64,6	$1{,}2\Delta T$
(iib)	FTR with K=2	52,1	$1{,}5\Delta T$

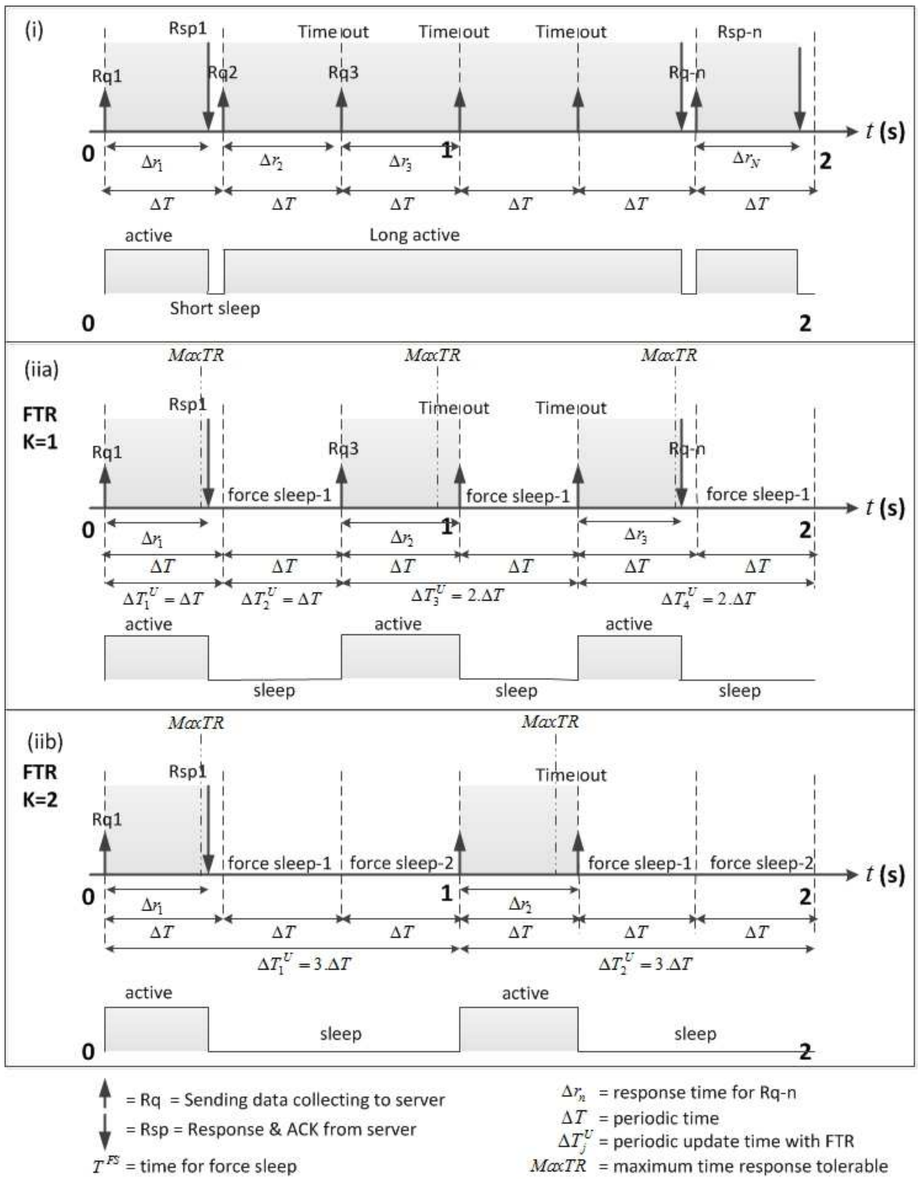

Figure 9 Comparing how the system works at very busy traffic conditions in three different scenarios: (i) Energy-Efficient Method Only, (iia) FTR with K=1, and (iib) FTR with K=2.

We can also see how the system performs under very busy traffic conditions for six duty cycle samples, as depicted in Figure 9. Note again that very busy traffic is indicated by a long response time even leading to HTTP timeout. Again, we compare the three scenarios: (i) Energy-Efficient Method Only, (iia) FTR with K=1, and (iib) FTR with K=2.

From Figure 9 we can compare the active and sleep periods for the three scenarios, as shown in Figure 10. From Figure 10 we can calculate the average energy efficiency and the average update time for all three scenarios, as summarized in Table 3.

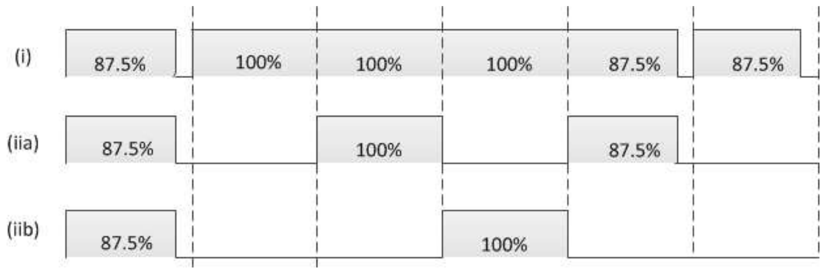

Figure 10 Summaries of active and sleep periods for very busy traffic conditions.

Table 3 Result of comparing three scenarios with average energy efficiency and average update time for very busy traffic conditions

No	Case	Average Energy Efficiency (%)	Average Update Time
(i)	Energy-Efficient Method Only	93,8	ΔT
(iia)	FTR with K=1	45,8	$1{,}5.\Delta T$
(iib)	FTR with K=2	31,3	$3.\Delta T$

From Table 3 we can see that our proposed method maintains the energy efficiency better than Energy-Efficient Method Only. Increasing K reduces the energy efficiency but also increases the average update time.

To validate our method we have conducted an experiment to get real measurement results from the network, which will be treated in the next section.

4 Experimental Setup

In this section we report the experiment we have conducted in order to validate our approach. The ideal network topology for this kind of research is depicted in Figure 11. In practice, it’s very difficult to set up and measure many nodes because we are limited by the number of nodes we actually have. Therefore, we have used a semi-real situation by combining simulation and experiment. We have simulated a number of nodes with an emulator. We created an application that could emulate node capabilities with our proposed method. It connected directly to the server via the Internet (public network) and emulated active, sleep and forced-sleep phases.

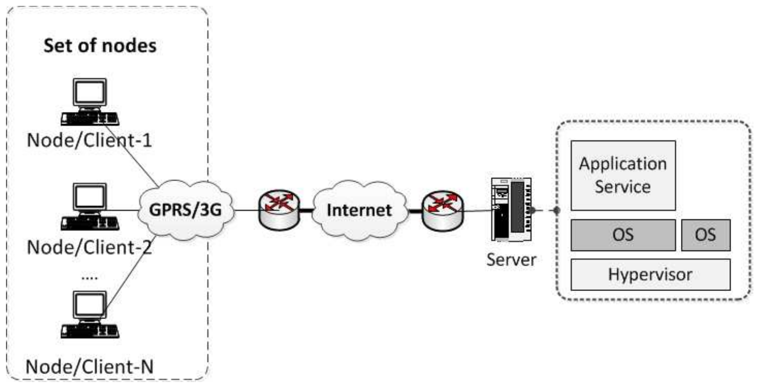

Figure 11 Ideal topology for experimental setup.

First, we set the number of emulated nodes to be used. All of them were designed to connect to the server via the Internet simultaneously for several times. The duration of measurement was 1800 seconds for each measurement in each scenario. The node emulator generated a log that recorded performance, including response time, HTTP timeout, number of active-sleep-forced sleep periods, etc. From this log we could retrieve information about average response time, average energy efficiency, average timeout experience and also average update time. We have carried out this simulation with different numbers of nodes (10 to 150).

Table 4 Experimental setup parameters for REST and EEM scenarios.

Parameter	Value
Transmission duty cycle	Periodical with ΔT = 5 seconds
HTTP timeout	Same with ΔT = 5 seconds
Time observation	1800 seconds (30 minutes)

Table 5 Experimental setup parameters for FTR scenario.

Parameter	Value
Transmission duty cycle	Periodical with ΔT = 5 seconds
HTTP timeout	Same with ΔT = 5 seconds
Time observation	1800 second (30 minutes)
MaxTR	2.5 seconds
K	3

With this simulation we compared three methods: REST-based application only (REST), Energy-Efficient Method Only (EEM), and Finite Time Response (FTR). Note that we also employed a REST-based application using the HTTP method GET for both EEM and FTR. However, EEM only used the Energy Efficiency Method while FTR used both the Energy Efficiency Method and the performance-aware method. The experimental setup parameters for REST, EEM and FTR are given in Table 4 and Table 5.

5 Simulation Results

In this section we summarize and discuss the simulation results. Figure 12 shows the impact of the number of nodes on the average response time. The graphic shows that increasing the number of nodes was followed by an increase of the average response time. For a large number of nodes, our proposed method (FTR) had a better performance than the conventional REST-based application. For example, when the number of nodes was higher than 100, FTR had a better average response time than REST/EEM.

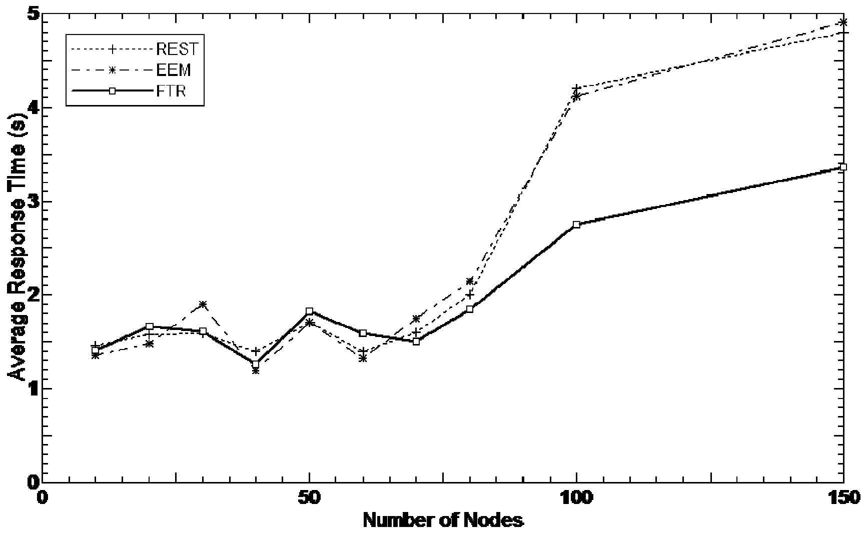

Figure 12 Impact of increasing number of nodes on average response time.

The increasing average response time was caused by the characteristic of HTTP as a best-effort service. The higher number of nodes concurrently sending data to the same destination may create more traffic. More traffic can create congestion in the network, which can actually be handled by TCP flow control. But TCP flow control has a mechanism to reduce packet data transmission whenever it detects congestion indicated by packet loss. So, when the number of nodes becomes higher it also increases the average response time. Our proposed method (FTR) has a better performance because flow control is not only handled at TCP level but also at the application layer. We propose a system that is designed to decide to send a message or pend it at the application layer. The decision is simple because it only uses the last response time as brief information about traffic conditions. FTR maintained better performance than the conventional REST-only method, especially for larger numbers of nodes.

The average response time is actually related to the average timeout experience in most cases. We know that when the response time is very long it can cause an HTTP timeout, because we have set HTTP timeout at a certain value (for this experiment: 5 seconds). Figure 13 shows that the average timeout experience for the conventional REST-only method increased when the number of nodes increased. However, our proposed method (FTR) reduced the average timeout experience to a lower value. Both Figure 12 and Figure 13 show that FTR maintained better average performance. Our method contributed to maintaining better performance parameters for both average response time and average timeout experience.

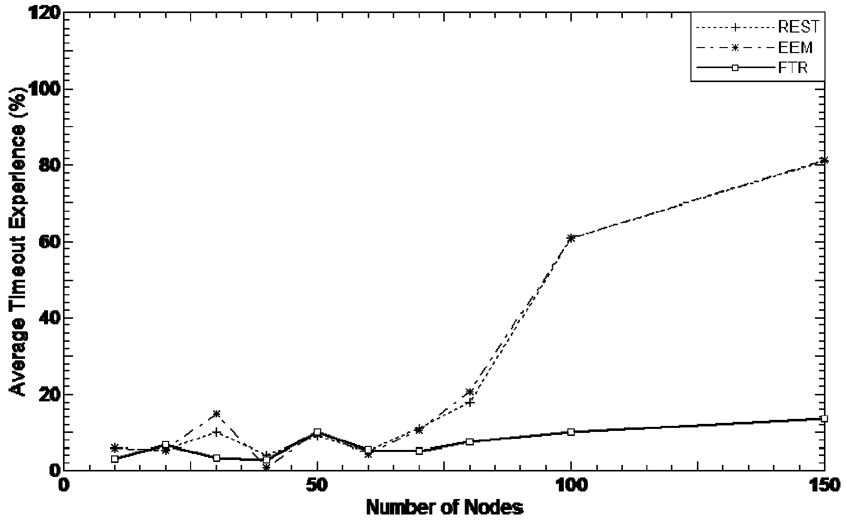

Figure 13 Impact of increasing number of nodes on average timeout experience.

Figure 14 shows the impact of the number of nodes on energy efficiency. The graphic shows that EEM and FTR had better energy efficiency management than the REST-based application. EEM has a mechanism to regulate the active and sleep states, so it performed better, especially when the number of nodes was lower. However, a large number of nodes made energy efficiency deteriorate. This occurs because when the number of nodes increases, the average response time also increases. An increase in the average response time increases the duration of the active period, which results in a decrease of energy efficiency.

FTR better maintains energy efficiency because it not only has active and sleep states, but it is more balanced through the introduction of a forced sleep state. Forced sleep occurs whenever a long duration of active sleep has occurred during the previous duty cycle. Therefore, our proposed method not only maintains better performance but also better energy efficiency.

The conventional REST method always has a value of 100% regardless of the number of nodes. This happens because of the energy-efficiency calculation that was proposed in Eq. (1). The REST-based application has the worst energy-efficiency mechanism because it doesn't have any coordination with the physical layer and so it is always defined as active. Based on our equation, it is always in the active state, regardless if the session is active or idle.

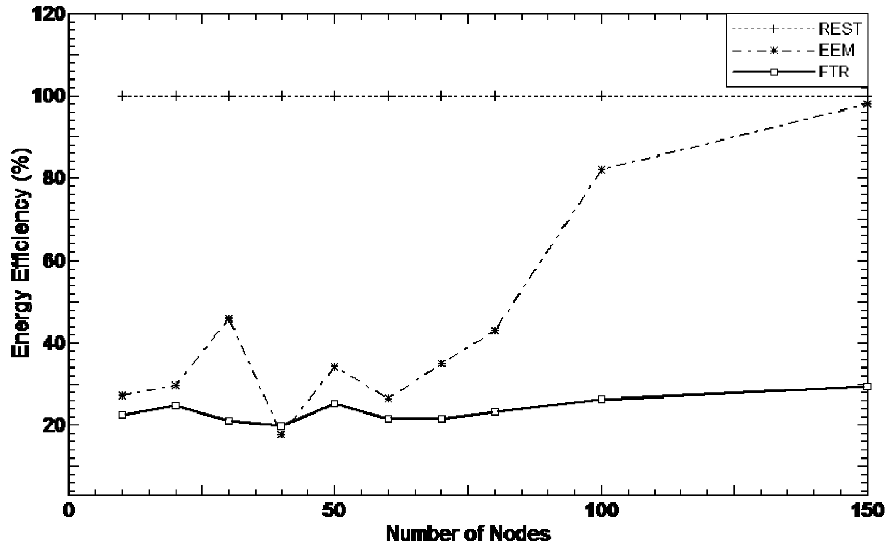

Figure 14 Impact of increasing number of nodes on average energy efficiency.

Even though FTR offers a better solution in view of performance and energy efficiency, it has a trade-off with another aspect: the average update time to the server, as depicted in Figure 15. The conventional REST services (REST and EEM) have a consistent update time, i.e. a fixed period that is always identical to the duty cycle (ΔT). From the figure, we can see that the update time for

REST and EEM always had value 1, meaning that it is equal to 1 x duty cycle (ΔT). FTR, on the other hand, can increase the update time, especially if the number of nodes increases. For example, if the number of nodes is 150, it has an update time of twice the duty cycle (2 x ΔT).

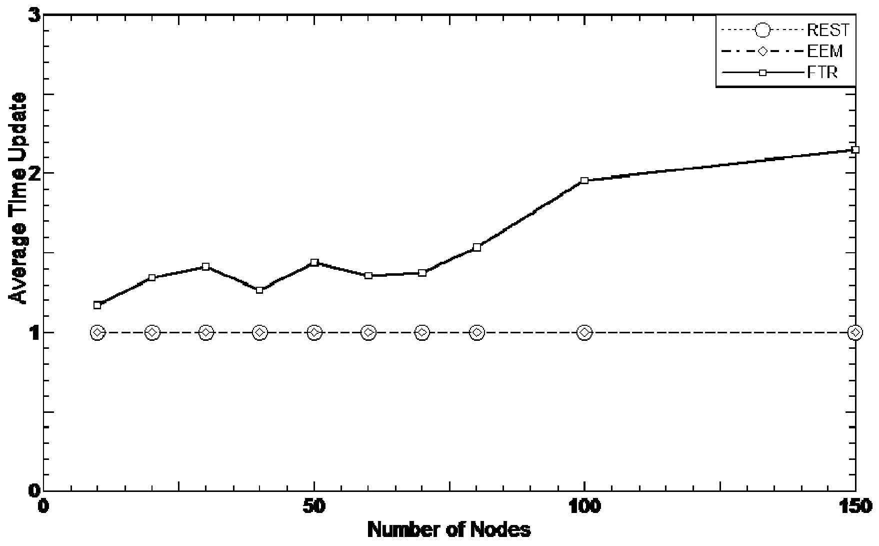

Figure 15 Impact of increasing number of nodes on average update time.

A higher value for the average update time means that the resolution of the data sent from the nodes to the server becomes lower. For example, if the system captures data from river water level sensors once every minute, it can change to once every 2 minutes. Even though the resolution can be set based on need or user preference, we must still be aware how much it deviates from the previously set value. We can find the optimum value between better performance and better energy efficiency, but still have to consider a maximum value for the update time as well. In other words, if we need a system with better performance and energy efficiency, we may have to sacrifice data resolution in terms of a higher average update time.

6 Conclusions

In this paper we have proposed a new method for communication between sensor networks into server via the Internet. We have proposed a modification of HTTP and a cross-layer approach to coordinate with the physical layer. At the physical layer the active state or sleep state will be set. We have also proposed a forced-sleep state that is triggered by the last performance measurement at the application layer. If the previous performance response time has a value larger than a certain acceptance range, then the node enters forced sleep mode during the next duty cycle. Our proposed method is called Finite Time Response (FTR).

We use a push-based system for network system updating. A push-based system is suitable for capturing data periodically, for example for environment or disaster monitoring. However, when the number of nodes becomes larger there could be new problems in terms of performance degradation, such as increased delay or response time, increased timeout messages and more power consumption. FTR is proposed as a communication protocol that can maintain better performance and better energy efficiency.

We have compared FTR with a conventional REST-based application and a method using energy-efficiency control only. The results of our experiments have shown that FTR maintains better performance, indicated by a lower average response time and a lower average timeout experience. But FTR has to be traded off with the average update time. Even though FTR has successfully maintained performance with a better response time, the average update time also increased. Optimization is still needed to gain better performance, better energy efficiency while also considering a maximum value for the average update time.

References

[1] Al-Ali, A.R., Zualkernan, I. & Aloul, F., *A Mobile GPRS-Sensors Array for Air Pollution Monitoring*, IEEE Sensors Journal, **10**(10), pp. 1666-1671, 2010.

[2] Keoduangsine, S. & Goodwin, R., *A GPRS-Based Data Collection and Transmission for Flood Warning System: The Case of the Lower Mekong River Basin*, International Journal of Innovation, Management and Technology, **3**(3), pp. 217-220, 2012.

[3] Luckenbach, T., Gober, P. & Arbanowski, S., *TinyREST – a Protocol for Integrating Sensor Networks into the Internet*, in Proc. of REALWSN, pp. 101-105, 2005.

[4] Shelby, Z., Frank, B. & Sturek, D., *Constrained Application Protocol (CoAP), Internet-Draft*, available at: http://tools.ietf.org/html/draft-ietf-core-coap-04, 2010 (February 17, 2013).

[5] Colitti, W., Steenhaut, K. & De Caro, N., *Integrating Wireless Sensor Networks with the Web*, In IP+SN, 2011.

[6] Suakanto, S., Supangkat, S.H., Suhardi & Saragih, R., *Performance Measurement of Cloud Computing Services*, International Journal on

Cloud Computing: Services and Architecture (IJCCSA), **2**(2), pp. 9-20 2012.

[7] Vemuri, S.R., Satyanarayana, N. & Prasanna, V.L., *Generic Integrated Secured WSN- Cloud Computing U-life care*, International Journal of Engineering Science & Advanced Technology [IJESAT], **2**(4), pp. 897-907, 2012.

[8] Dash, S.K., Mohapatra, S. & Pattnaik, P.K., *A Survey on Applications of Wireless Sensor Network Using Cloud Computing*, International Journal of Computer Science & Emerging Technologies, **1**(4), pp. 50-55, 2010.

[9] Gaynor, M., Moulton, S.L., Welsh, M., LaCombe, E., Rowan, A. & Wynne, J., *Integrating Wireless Sensor Networks with the Grid*, IEEE Internet Computing Magazine July **-** August 2004.

[10] Wang, W., Lee, K. & Murray, D., *Integrating Sensors with the Cloud Using Dynamic Proxies*, IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), 2012.

[11] Suakanto, S., Supangkat, S.H., Suhardi, Saragih, R., Nugroho, T.A. & Nugraha, I.G.B.B., *Environmental and Disaster Sensing Using Cloud Computing Infrastructure*, International Conference on Cloud Computing and Social Networking, April 2012, IEEE Catalog Number CFP1201T-ART, 2012.

[12] Ye, W., Heidemann & Estrin, D., *An Energy Efficient MAC Protocol for Wireless Sensor Networks*, In 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM' 02), New York, United States, 2002.

[13] Zheng, T., Radhakrishnan, S. & Sarangan, V., *PMAC: An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks*, Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005.

[14] Dunkels, A., Voigt, T. & Alonso, J., *Making TCP/IP Viable for Wireless Sensor Networks*, In Work-in-Progress Session of the first European Workshop on Wireless Sensor Networks (EWSN 2004), Berlin, Germany, January 2004.

[15] Di Francesco, M., Anastasi, G., Conti, M., Das, S.K. & Neri, V., *Reliability and Energy-Efficiency in IEEE 802.15.4/ZigBee Sensor Networks: An Adaptive and Cross-Layer Approach*, IEEE Journal on Selected Areas in Communications, **29**(8), September 2011.

[16] Alam, M.M., Berder, O., Menard, D. & Sentieys, O., *TAD-MAC: Traffic-Aware Dynamic MAC Protocol for Wireless Body Area Sensor Networks*, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, **2**(1), pp. 109-119, March 2012.

[17] Bose, R. & Helal, A.(S)., *Sensor-Aware Adaptive Push-Pull Query Processing in Wireless Sensor Networks*, pp. 243-248, 2010 Sixth International Conference on Intelligent Environments, 2010.