

# LiteWS: A Web Service Enhancing Multi-User Queries in Data Intensive Sensor Networks

Masaaki Takahashi and Bin Tang

Department of Electrical Engineering and Computer Science

Wichita State University

Wichita, KS 67260

mxtakahashi@wichita.edu, bintang@cs.wichita.edu

**Abstract**—Internet-based data intensive sensor networks (DISNs) are sensor networks wherein large volume of different types of sensory data are sensed and generated from the physical world, and queried by multiple users simultaneously. One critical issue in Internet-based DISNs is how to support multiple user queries simultaneously in an efficient manner, in terms of query response time, query loss ratio and sensor node energy consumption. In this paper, we formulate and study *multi-user data query problem in DISNs* and present our solution. Specifically, we present our design, implementation, and evaluation of LiteWS, a web service system aiming to enhance multi-user queries in Internet-based DISNs. We propose a simple caching technique and give analytical analysis of how to reducing query loss ratio in our system. Through extensive experiments based on Crossbow IRIS motes, we evaluate the system performance of LiteWS under both correlated and uncorrelated query traffic. We show the performance with data caching is much better than the one without; particularly, the average query response time of LiteWS is improved 5 to 10 times and the query loss rate is improved 2 times.

**Keywords** – Multi-user Queries, Web Services, LiteOS, Data Intensive Sensor Networks

## I. INTRODUCTION

Recently the integration of WSNs with Internet has attracted much attention in both research community and industry. Connecting sensors to the Internet makes it possible for people to monitor the physical environment of interest from anywhere and anytime. It not only gives users more visibility and control to manage and use WSNs, but also gives rise to a whole range of new sensor network applications in scientific computation [14]. One salient example is the data-intensive sensor networks (DISNs) for scientific applications, where large volume of scientific data sensed or generated are constantly queried and shared by many researchers from different parts of the world. The sensory sources include a rich collection of sensors such as video cameras, microphones, RFID readers, environmental or weather sensors, telescopes, seismometers; corresponding scientific applications include climate change, earthquake detection and characterization, and environmental monitoring of large ecosystems.

Due to its *data-intensive* and *query-based* nature, the Internet-based DISNs pose more challenges compared to traditional, standalone sensor network applications. Sensor energy

consumption, the query response time and query loss ratio are all important to consider. In this paper, we study the *multi-user data query problem* in Internet-enabled data intensive sensor networks, wherein large volume of different types of sensory data in the physical world are constantly queried by multiple users simultaneously. Our goal is to minimize the average user query response time and average user query loss ratio while reducing the energy consumption of sensor nodes. We propose a simple data caching technique to specifically address such needs for DISNs. Using some analytical model, we further show that in multi-user queries, the query loss ratio can be attributed to not only the packet collisions, but also the user queries themselves, and we improve the query loss ratio accordingly.

Recently, web services for sensor networks have been proposed and studied in both research community and industry. Web services provide structured data and programmatic access to functionalities of resource-constrained sensor nodes, thus enabling interoperability among sensor systems written in different programming languages and running on different platforms. However, in terms of design and implementation, most of the existing work are based on TinyOS [1], the de facto standard embedded operating systems for sensor networks. Even though TinyOS is a very mature operating system with tested industrial strength, its programming paradigm, including NesC, wiring, and state-machine abstractions of program execution, introduces a learning curve for most traditional programmers.

In this paper, we present our design, implementation and evaluation of a web service called LiteWS. LiteWS is based on LiteOS [2, 4], a new operating system for sensor networks recently developed by UIUC. Our web service middleware takes advantage of the UNIX-like shell commands as well as the C programming language supported by LiteOS, which provides better programming interaction between web service middleware and sensor networks. We believe that its affinity to UNIX makes LiteOS easier to be adopted for applications such as web services.

The main contributions of our paper are as follows:

- We tackle the multi-user query problem in Internet-based DISNs, wherein multiple users request different data at

different rates. Using a web service prototype, we show how it can be solved in an efficient manner.

- We design and implement a web service called LiteWS, as a module in LiteOS. We identify and improve some existing limitation of LiteOS for the purpose of efficient multi-user query.
- We propose and implement a data caching sublayer LiteWS for performance improvement, and give analytical analysis of how to reducing query loss ratio in our system.
- Using real implementation based on Crossbow IRIS motes, we extensively evaluate the system performance of LiteWS. The average query response time of LiteWS can be improved 5 to 10 times and the average query loss ration 2 times with the data caching algorithm.

## II. RELATED WORK

Muller and Alonso [13] tackle the problem of multi-user support in sensor networks by transforming it into a multi-query optimization problem. In their work, the submitted user queries are merged into a network query which is then sent to the sensor network for execution. Their goal is to minimize the energy consumption of sensor nodes and query loss ratio. We tackle the multi-user program by using proposing a programmer friendly web service system and consider query response time, query loss ratio and sensor node energy consumption in a holistic way. Yates et al. [17] study the benefit and cost of caching data in sensor network monitoring, and evaluate several approaches of querying and caching. They show that data accuracy and data delay both play important roles in reducing query cost, the sensor energy consumption.

Priyantha et al. [16] propose to use web services to support interoperable and evolvable sensor networks. Their work is the first to show that web services improve the programmability of the Internet-based sensor networks. They identify design choices that optimize the web service operation on resource constrained sensor nodes and implement the web service on each individual sensor. Their web services are TinyOS based. We adopt LiteOS for better programming interaction between web service middleware and sensor networks. To mitigate the energy constraint of sensor networks, we implement the LiteWS middleware on the resource-rich gateway nodes.

To query or access data generated by the sensor nodes, the sensor network can be viewed as a distributed database. There have been lots of work to study and develop the database middleware to support in-network database query operators such as grouping, aggregation, and joins (for example, [3, 6, 12]). Our work is to design a web service middleware system which facilitates the interaction between the users and the sensor networks.

LiteOS [2, 4] is a newly developed operating system for embedded sensor networks. It creates a familiar UNIX-like environment for users where they can interactively command the entire sensor network to perform tasks such as reprogramming, data retrieval, or network reconfiguration. This could

potentially expand the circle of sensor network developers by leveraging their knowledge such as Unix and threads. Our experience proves that LiteOS serves as a good platform to develop the web service. Another way to reduce the learning curve of sensor network programming is to treat each individual sensor as a full-fledged IP node. Arch Rock [5] provides a Web-based access to individual sensor nodes using the standard TCP/IP protocols, and enables users to configure, monitor and manage a sensor network from a secure browser. The authors in [7–9] also treat sensor nodes as IP or IPv6 nodes to support network layer interoperability. We use LiteWS to focus on the application middleware to improve the query response time and reduce query loss rate in data intensive sensor networks.

Surprisingly, not much research has been done specifically for DISNs despite their potential scientific applications. So far, researchers address the challenges of DISNs mainly by studying the medium access control (MAC) protocols [10, 11, 15]. The authors either study the medium access using simulations, or study the trust management in DISNs. Obviously the efficiency of MAC plays a key role in the achievable throughput of a high data rate wireless network. We study DISNs from a different angle of optimizing multi-user data queries, and focus on designing techniques to reduce query response time and query loss ratio under correlated and uncorrelated query traffic, using real implementation. Our work is orthogonal to above approaches, and can be used in combination with them to further address the challenges in data intensive sensor applications.

## III. MULTI-USER DATA QUERY PROBLEM IN DATA INTENSIVE SENSOR NETWORKS

In this section, we first present our network model and sensory data model, and multi-user query problem in DISNs. Then we discuss our data caching algorithm. Finally, we derive the query loss ratio based on the data caching algorithm and propose solution to further reduce it.

### A. Models and Problem Statement

Network model and data model. The network model of our data query problem in DISNs is illustrated in Figure 1. There are  $N$  sensor nodes in the sensor field and one base station<sup>1</sup>, which serves as the communication hub between sensor network and its user queries. We assume that the sensor nodes are commodity-manufactured and each sensor only has one radio interface. There is one wireless channel available in the DISNs. In the physical environment, there are  $t$  different types of data (or physical phenomena) such as temperature, light, magnet and acceleration to be sensed. We can also consider these as the scientific data collected and queried by domain scientists in scientific applications, or medical data of patients in health monitoring systems. Each sensor node can

<sup>1</sup>We use base station and gateway node interchangeably.

sense any data type, and sense only one type at a time. The varying of the values of data type  $i$  ( $1 \leq i \leq t$ ) with respect to time is characterized by function  $f^i(j)$ , where  $j = 0, 1, 2, \dots$  is time slot of fixed-length interval.

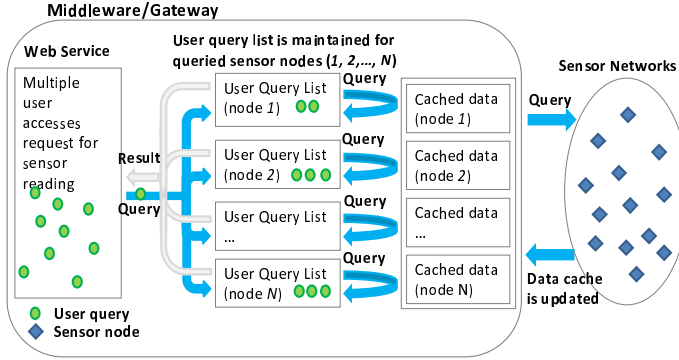


Fig. 1. Multi-user data query problem in data intensive sensor networks.

User Queries. The DISNs are **query invoked** – sensors are asleep until they are activated by queries, and then they sense and send data to base station. The base station implements the web service middleware, which handles all the user queries. Like that of the sensory data, the user queries arrive in a time-slotted manner (we introduce more detailed query arrival models in Section IV). There are  $m$  user queries: query  $i$  has the form  $(i, sensor_i, arrivaltime_i, type_i, number_i, interval_i)$ , where  $sensor_i$  is the sensor the user wants to query,  $arrivaltime_i$  is the time slot at which query  $i$  arrives at the base station,  $type_i$  is the type of data readings requested by query  $i$ ,  $number_i$  is the number of data readings requested by query  $i$  starting from time slot  $arrivaltime_i$ , and  $interval_i$  is the interval, or sampling period (in number of time slots) of data readings requested by query  $i$ .

Multi-user Data Query Problem in DISNs. Since there is only one channel and one radio interface, the base station can communicate with one sensor node at a time. When multiple queries are present at the base station simultaneously, each query's response time may increase significantly, and some number of the requested data readings may get lost due to packet collision. We define the following metrics.

- For each user query, we define the **query response time** as the time elapse between the slot when the base station receives the query and the slot when the base station receives the first data reading for that query. To focus on our web service performance, query response time excludes the time taken from the web users to base station.
- To characterize the data reading loss, we define the **query loss ratio** of each query as the ratio of the number of incorrect (or lost) readings divided by the total number of requested readings of that query.

We call our problem *multi-user data query problem in data intensive sensor networks*. The goal of the problem is to minimize the average user query response time and average query loss ratio of all the queries received while reducing the energy consumption of sensor nodes, under the constraint that there is single channel and single radio interface of each sensor node. Below we first present some caching technique used in our LiteWS.

## B. Data Caching Algorithm

The data caching technique works as follows. There are two kinds of caches: **Node Cache** and **BS Cache**. Each sensor node maintains a Node Cache to store the latest transmitted data reading of each type (if the sensor node is not yet invoked to sense and transmit data, the cache is empty). The base station maintains a BS Cache, which saves the most recent reading received for each data type from each sensor node. Below we explain the data caching algorithm in details.

Node side. After each sensor node senses a new reading, if the difference between the new reading and the one in its Node Cache is bigger than some threshold, the sensor node updates the Node Cache and then transmits the reading to the base station. (As the future work, we plan to adopt some statistical methods to more accurately characterize the threshold and decide whether two consecutive readings are close or similar enough.)

Base station side. To decide the freshness of the data readings in the BS Cache, the base station maintains a queue for each queried sensor node and each queried data type, called **current query list**, as shown in Figure 1. The current query list of each data type of each sensor node stores all the user queries currently requesting that data type from that sensor node in the order of their arrival time. The empty queue of a type indicates that currently no query is requesting that data type from that sensor node, thus the data in the corresponding BS Cache may not be fresh.

When the base station receives the query  $(i, sensor_i, arrivaltime_i, type_i, number_i, interval_i)$ , it first checks the current query list of  $type_i$  for  $sensor_i$ . If it is not empty (which means there are queries currently requesting the data type  $type_i$  from  $sensor_i$ ), user query  $i$  will read the data directly from the BS Cache of  $type_i$  for  $number_i$  times, with interval as  $interval_i$ . Otherwise, the base station will directly invoke sensor node  $sensor_i$  for query  $i$ 's readings.

Discussion of the algorithm. Our data caching algorithm works the best for the scenario where data readings are not changing constantly. This can be justified by the observation that for most of the physical phenomenon, they stay the same most of the time. Still, due to the data intensive nature of our problem, collision and contention are the major reasons for data query response delay and data reading loss. As a result, the fact that the current query list is non-empty can not guarantee that the data in the corresponding BS Cache is

fresh – due to collision and contention, the readings sent from sensor nodes to the base station may get lost, thus failing to update the corresponding BS Cache. Query loss ratio we define previously is used to quantify such effect. In next subsection, we further show that the query loss ratio are attributed to not only the packet collisions, but also the multi-user queries themselves, and we improve the query loss ratio accordingly.

### C. Query Loss Ratio

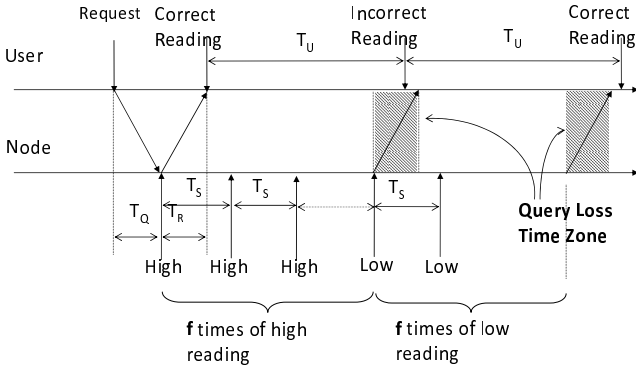


Fig. 2. Analyzing the query loss ratio for one user querying one data type.

Below we quantify the query loss ratio in above caching-based multi-user queries and propose solution to further reduce it. We use  $T_Q$  to denote the query message transmission time from a base station to a sensor node, and  $T_R$  to denote the response message transmission time from a sensor node to a base station. Considering the sizes of both query message and response message (the sensory data) are small, we assume that  $T_Q = T_R = \delta$ . For the ease of representation, we assume the values of any queried data types are changing between high and low alternatively. We believe this is a simple but well-justified model to characterize the data variation in data intensive applications. We begin by discussing multiple user queries for one sensor node with different interval (or sampling period).

We assume this data type changes between high and low values every  $f$  time slots, i.e., the sensor detects a different data reading every  $f$  readings and sends it back to the base station. We start by first analyzing the query loss ratio for one user querying one sensor node. We then show how it can be extended into multiple user queries.

**One user query for one sensor node.** We denote user query period and sensor sensing period using  $T_u$  and  $T_s$  respectively. Number of requested data readings by the user is  $n_u$ . Then the total query time is  $n_u \times T_u$ . We assume the time slot when the query receives its first reading is time slot 0. As mentioned above, after the sensor node senses a new reading, if the new reading is different from the one in its Node Cache more than some threshold, the sensor node sends a update

message with the sensory data back to the base station (in our above assumption, this happens when the data type value changes from high to low or from low to high). However, as shown in Figure 2, during the time period when an update message has already been sent out by the sensor and before it reaches the base station, if the query reads from the BS Cache, it does not get the correct reading and thus causes a query loss. We call such time periods the Potential Query Loss Time Zones. There are  $n_t = n_u \times T_u / (T_s \times f)$  such Potential Query Loss Time Zones:  $[T_s \times f - \delta, T_s \times f]$ ,  $[2T_s \times f - \delta, 2T_s \times f]$ , ...,  $[n_t T_s \times f - \delta, n_t T_s \times f]$ . Therefore, there still exists query loss even there is no contention and collision in the network. To calculate such query loss ratio, we need to find for each of the  $n_u$  user readings, whether it falls into one of the Potential Query Loss Time Zones. We use  $a_l = 1$  to indicate that the  $l^{th}$  ( $1 \leq l \leq n_u$ ) query reading incurs a query loss, and  $a_l = 0$  otherwise, as follows.

$$a_l = \begin{cases} 1 & \text{if } (kT_s \times f - \delta)/T_u \leq l \leq (kT_s \times f)/T_u, \\ & \text{where } 1 \leq k \leq n_t \\ 0 & \text{otherwise} \end{cases}$$

Then the total query loss ratio of a single user is  $\sum_{l=1}^{n_u} a_l / n_u$ . The query losses can be avoided with the following simple algorithm. For the user's  $l^{th}$  ( $1 \leq l \leq n_u$ ) query reading, if  $(kT_s \times f - \delta)/T_u \leq l \leq (kT_s \times f)/T_u$  where  $1 \leq k \leq n_t$ , then the query waits another  $\delta$  time beyond its preset query reading time, until a new sensor reading is updated at BS Cache. Otherwise, the user follows the preset query reading time.

**Multiple user queries for one Sensor Node.** Essentially, multiple users scenario is a summation of each individual user's single user scenario. Let  $\tau(i)$  be the arrival time slot of the user and  $T_u(i)$  is the query period of the  $i^{th}$  user respectively (there are total  $m$  users). Let  $a(l, i) = 1$  indicate that the  $l^{th}$  query of the  $i^{th}$  user gets a query loss,  $a(l, i) = 0$  otherwise. Then,

$$a(l, i) = \begin{cases} 1 & \text{if } (kT_s \times f - \delta)/T_u \leq l \leq (kT_s \times f)/T_u \\ & \text{and } \tau(i) \leq T_u(i), \text{ where } 1 \leq k \leq n_t \\ 0 & \text{otherwise} \end{cases}$$

Then the total query loss ratio for  $m$  users is  $\sum_{i=1}^m \sum_{l=1}^{n_u} a(l, i) / n_u$ . Each user can use the same above algorithm to avoid such query loss.

## IV. EXPERIMENTAL EVALUATION

Our testbed comprises of five sensor nodes, one base station and one SQL server. Each sensor node is a Crossbow's IRIS Mote coupled with a sensor board (MTS310). IRIS Mote offers up to three times improved radio range and twice the program memory over previous MICA Motes, which makes them an ideal platform for our web service development.

### A. Response Time in LiteOS.

During our implementation of LiteWS, we found that the interactivity offered by LiteOS significantly contributes to the round trip time delay between the base station and the sensor nodes. In LiteOS, in order to get the sensor readings, a sequence of “ls” or “cd” commands are called. Notice that LiteOS is a stateless operating system. Whenever commands like “ls” or “cd” are executed, base station internally keeps channel and sensor node ID information rather than the sensor node keeps those information. Therefore, when “./light” command is executed, its packet information includes both channel and node ID. Consequently, the sequence of “ls” and “cd” commands can be completely removed, thus resulting much lower communication delay. By simply handling and sending a packet information of “./light” command to a sensor node, the response time dramatically improves from 2114 ms to an average of 70 ms.

### B. Experiment Settings

In this subsection, we first discuss the hardware in our experiments. We then introduce the query models we used in our experiments. We finally present our experiment results and analysis.

Varying of Sensory Data. In our testbed environment, the real sensed data values of the physical phenomena (light, temperature...) do not change much. To emulate a data intensive application wherein interesting sensory data are generated dynamically, we construct a sequence of constantly changing data values. As in the analytical analysis, we adopt that the data values are changing in a time-slotted manner, between high and low periodically with *interval* time slots. We believe this is a simple but well-justified model to characterize the data variation in data intensive applications and evaluate LiteWS. More intricate and close-to-environment sensor data models will be studied in the future.

User Query Arrival Models. User query arrival occurs at the beginning boundary of each time slot. We define the **query arrival rate** as the probability that a new user query arrives in a time slot. We use the following two query arrival models.

- Bernoulli model. The probability that there is a user query arriving in each time slot is identical and independent of any other time slot. This model refers to uncorrelated arrivals of the user queries. This probability represents the query arrival rate.
- Bursty model. User queries are generated by a 2-state Markovian process which alternates between IDLE and BUSY states. The process remains in each state for a geometrically distributed number of time slots, with expected duration  $E[B]$  and  $E[I]$ , respectively. During the BUSY state, user queries (requesting for randomly different data type) arrive continuously in consecutive time slots (one query at one time slot). No queries arrive during the IDLE

state. We set  $E[B]$  to be 16 time slots.<sup>2</sup> The query arrival rate is given by  $p = E[B]/(E[B] + E[I])$ .

### C. Experiment Results and Discussions

Our LiteWS middleware has around 2000 lines of Java and C codes. In our experiments, we set the duration of each time slot as 100 ms. For each arriving query, it requests 10 data readings starting from the time slot it arrives. Each query randomly selects one of light, temperature, acoustic, magnet, or acceleration as its requested data type. To achieve stability in performance metrics, each of our experiments is run for a sufficiently long time (5000 time slots for our experiments). Considering the instability of wireless medium, each data point in our experiment results is an average of five time measurements. We compare the average response time and average query loss ratio between with and without data caching, under above two different query arrival models. In both models, we increase the query arrival rate from 10% to 24%, beyond which the query loss ratios for all the scenarios keep unchanged.

Figure 3 shows the comparison under Bernoulli model. Figure 3 (a) shows the performance of the response time with respect to the query arrival rate. For without data caching, since all of data is from sensor node, response time increases when query arrival rate increases. However, for with data caching, response time decreases when query arrival rate increases. This is because when query arrival rate increases, more queries access to the cached data, which decreases the average response time. Figure 3 (b) shows the performance of the query loss ratio with respect to the query arrival rate. With the increase of the query arrival rate, the query loss ratios of both with and without caching increase. However, with data caching has a much lower query loss ratio compared to without data caching. Generally, the average query response time of LiteWS is improved 5 to 10 times and the query loss rate is improved 2 times with data caching. At high query arrival rate, the average query response time with data caching is two orders of magnitude better.

Figure 4 shows the performance comparison under Bursty model. Our observation is that even though without caching performs not as well as with caching in terms of both average response time and average query loss ratio, it performs much better under Bursty traffic model than under Bernoulli traffic model, especially when the query arrival rate is between 10% and 20%. Particularly, the average response time only increases from 50 ms to 68 ms, and the query loss ratio almost keeps unchanged, when query arrival rate increases from 10% to 20%. In higher query arrival rate, the LiteWS performance is not much different compared to under Bernoulli model.

<sup>2</sup>The choice of an expected duration of 16 time slots per burst is arbitrary, but is representative. The same qualitative results are obtained for different burst lengths.

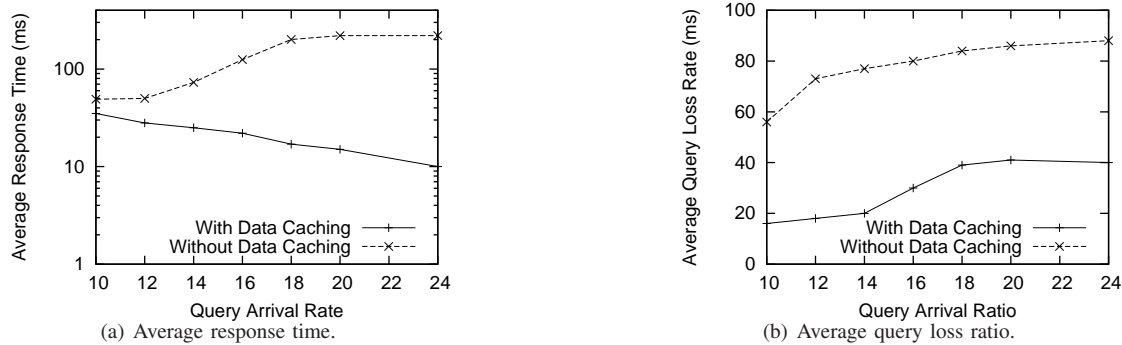


Fig. 3. Comparison between with and without data caching under Bernoulli model.

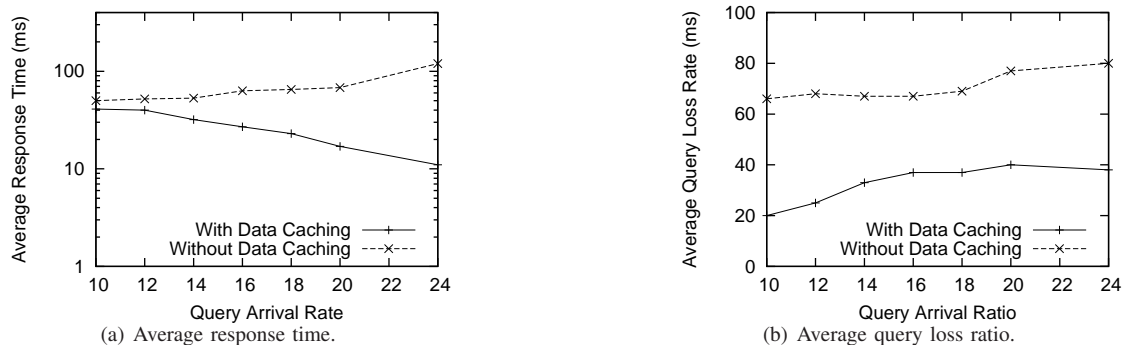


Fig. 4. Comparison between with and without data caching under Bursty model.

## REFERENCES

- [1] TinyOS: <http://www.tinyos.net>.
- [2] LiteOS: <http://www.liteos.net>.
- [3] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of the International Conference on Mobile Data Management (MDM 2001)*.
- [4] Qing Cao, Tarek Abdelzaher, John Stankovic, and Tian He. The liteos operating system: Towards unix-like abstractions for wireless sensor networks. In *Proc. of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2008)*, pages 233–244.
- [5] Arch Rock Corporation. Demo abstract: A sensor network architecture for the ip enterprise. In *Proc. of the 6th International Conference on Information Processing in Sensor Networks (IPSN 2007)*, pages 575–575.
- [6] Amol Deshpande, Suman Nath, Phil Gibbons, and Srini Seshan. Cache-and-query for wide area sensor databases. In *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*.
- [7] A. Dunkels. Full tcp/ip for 8-bit architectures. In *Proc. of the 1st International Conference on Mobile systems, Applications and Services (MobiSys 2003)*, pages 85 – 98.
- [8] Mathilde Durvy, Julien Abeillé, Patrick Wetterwald, Colin O’Flynn, Blake Leverett, Eric Gnoske, Michael Vidales, Geoff Mulligan, Nicolas Tsiftes, Niclas Finne, and Adam Dunkels. Poster abstract: Making sensor networks ipv6 ready. In *Proc. of the 6th ACM conference on Embedded network sensor systems (SenSys 2008)*, pages 421–422.
- [9] Jonathan W. Hui and David E. Culler. Ip is dead, long live ip for wireless sensor networks. In *Proc. of the 6th ACM conference on Embedded network sensor systems (SenSys 2008)*, pages 15–28.
- [10] C.-K. Lin. Channel access management in data intensive sensor networks. 2008. University of Pittsburgh, Ph.D. Dissertation.
- [11] C.-K. Lin, V. Zadorozhny, , and P. Krishnamurthy. Grid-based access scheduling for mobile data intensive sensor networks. In *Proc. of the 9th International Conference on Mobile Data Management (MDM’08)*.
- [12] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1), 2005.
- [13] Rene Muller and Gustavo Alonso. Efficient sharing of sensor networks. In *Proc. of IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2006)*.
- [14] Rohan Narayana Murty and Matt Welsh. Towards a dependable architecture for internet-scale sensing. In *Proc. of the 2nd conference on Hot Topics in System Dependability (HOTDEP 2006)*.
- [15] V. Oleshchuk and V. Zadorozhny. Trust-aware query processing in data intensive sensor networks. In *Proc. of International Conference on Sensor Technologies and Applications (SensorComm 2007)*.
- [16] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. Tiny web services: Design and implementation of interoperable and evolvable sensor networks. In *Proc. of the 6th ACM International Conference on Embedded Network Sensor Systems (SenSys 2008)*, pages 253–266.
- [17] David Yates, Erich Nahum, Jim Kurose, and Prashant Shenoy. Data quality and query cost in wireless sensor networks. In *Proc. of Fifth Annual IEEE International Conference Pervasive Computing and Communications Workshops (PerCom Workshops 2007)*.