2009 13th Panhellenic Conference on Informatics

Query Sensitive Storage for Wireless Sensor Networks

Alexis Papadimitriou
Dept. of Informatics
Aristotle University
Thessaloniki, 54124 Greece
Email:papajim@delab.csd.auth.gr

Dimitrios Katsaros
Computer and Communication
Engineering Department
University of Thessaly
Volos, Greece
Email:dkatsar@inf.uth.gr

Yannis Manolopoulos
Dept. of Informatics
Aristotle University
Thessaloniki, 54124 Greece
Email:manolopo@delab.csd.auth.gr

***Abstract*—Storage management in wireless sensor networks is an area that has started to attract significant attention, and several methods have been proposed, such as Local Storage (LS), Data-Centric Storage (DCS) and more recently Location-Centric Storage (LCS). Several modern applications, like context-dependent information dissemination for pervasive computing, on-demand warning in surveillance sensor networks and roadway safety warning, require that each originating event is stored around its point of origin. LCS is a suitable approach for such applications. Though, LCS does not take into consideration the origin of the queries, which is equally important to the storage method, because it has immediate influence on the experienced latency. This paper proposes a simple yet effective way of reducing the network latency, namely the Query Sensitive Storage (QSS) protocol. QSS makes certain that not only will the queries be answered, but all subsequent queries that originated in the same area will be answered faster. The experimental evaluation using the J-Sim simulator attests that with the proposed QSS protocol we can achieve smaller network latency at a minimum storage cost as compared to its state-of-the-art competitor, namely LCS.**

I. Introduction

Wireless sensor networks (WSNs) have emerged as an efficient way of monitoring the physical world and have been an area of significant research in recent years [1]. The advances in technology have made it possible to build sensors, which are devices with microprocessors, memory, sensing and radio communication capabilities. The low cost of these devices enables deploying them in large numbers in the application environment. There is an increasing number of applications and fields where sensors can be used [6]. Some examples would be to monitor the environment [11] (severe weather, fires, earthquakes, volcanoes), to help in constructions (energy preserving buildings, building integrity) and agriculture (effective watering), to use for military purposes [3], [8] (sense mines, target detection and tracking) and many more. All these applications have different characteristics, which mean that often different protocols need to adapt the network architecture to satisfy each one.

Storage management is an area that has started to attract significant attention in the context of wireless sensor networks. In all the forementioned applications, it is apparent that the storage management approach for the WSN should address efficiently the following two goals:

- *Minimize the size of stored data (per sensor).* Since the sensors are devices with inherently limited capabilities, minimizing the size of data that need to be stored leads to improved data retention, since the network can continue storing data for longer periods of time.
- *Perform efficient query execution.* Whether the queries are dynamically generated or static, storage management can influence the efficiency of query execution. Query execution efficiency can be measured in various terms, but two of the most important are the *completeness*, i.e., whether the querer gets any answers, and the second is the incurred *latency*, i.e., the time elapsed until the querer gets the responses.

Although storage minimization is very significant, the recent advances in NAND flash storage [9] (i.e., they become cheaper, larger and more common) have made the former goal less important as compared to the latter. Recent market predictions forecast that within the next three years the capacity of the flash memories (e.g., NAND) will be at the order of Terabytes [1]. Presently, the NAND flash devices can store a few gigabytes data [2]. Therefore, the goal of minimizing the latency prevails over the storage minimization, and thus we could trade some storage to improve query latency. In the next subsection, we briefly outline the motivation and contributions of the present work.

A. Motivation and article's contributions

There exist four major families of data storage techniques in the context of WSNs, namely *External Storage* (ES), *Local Storage* (LS), *Data-Centric Storage* (DCS), and *Location-Centric Storage* (LCS). In the sequel, we will briefly describe these approaches; a more thorough description can be found in Section II.

In External Storage, data are transferred outside the network to a sink, which compared to a sensor is not resource-starving; data can then be processed as in traditional database systems. The obvious drawback of this method is the communication overhead that the sensors incur, especially the ones closest

[1]http://www.techworld.com/storage/news/index.cfm?NewsID=4387.

to the sink as they are more likely to become hotspots. In Local Storage the data is stored locally at the sensor that sensed the event. Thus the queries must flooded inside the sensornet in order to find the sensors that sensed the event of interest. In Data-Centric Storage [10], the data is stored according to its name and location; it uses geographic hash tables to map data to specific locations in the network. Thus, an event could be stored at a location that is far from the location where the event originated. Considering the mine-warning application described earlier, we can easily deduce that this kind of storage could have fatal consequences. Besides, similar to LS, DCS creates hot spots. Finally, Location-Centric Storage [14] stores the generated events in concentric circles with gradually larger radii, around the sensor where the event was originated. Although this storage protocol is suitable for our target applications, it suffers from the drawback that a querer can be informed about the events of his/her interest only when s/he approaches the origin sensors; thus his/her queries will probably experience high latency.

In light of the preceding discussion, we can conclude that for the target applications of this work, none of the existing storage management protocols is appropriate, in the sense that they can not efficiently avoid hot-spots and at the same time guarantee low latency.

The contribution of the present article is to design a dynamic on-demand storage protocol that stores data closer to the user that initiated a specific query so that subsequent queries instigated by users can receive an answer at a smaller latency. Specifically, the article's contributions can be summarized as follows:

- Description of a dynamic on-demand storage protocol that improves upon the state-of-the-art relevant protocols.
- Performance evaluation of the protocol and comparison with the state-of-the-art method, using an established simulation package (J-Sim).

The rest of the paper is organized as follows. In Section IIwe will discuss the relative research that has been done on the subject of storage. Section III presents our method description in detail, while Section IV evaluates the proposed method against the state-of-the-art relevant method. Finally, the paper concludes with Section V.

II. Previous Work

Ratnasamy et al. propose the third protocol, DCS [10], which uses hash tables to match events with storage locations. This way, events with similar names can be stored at the same or near by locations. Thus the necessity for query flooding is avoided since the user knows where the respective data to his query are stored. Therefore, the data is stored at a location which can be quite far, depending on the hash function, from the point of generation. The DCS protocol also suffers from a single point of failure. Since all data with the same name are stored at a sensor at a specific location, a sudden battery loss or malfunction of the sensor would compromise the data stored. The proposed way to solve the problem according to Ratnasamy et al. is to replicate the data locally, around the hashed location.

A variation of the DCS protocol is based on Rendezvous Regions [12]. In this case, data are mapped to a whole region rather then a single point. Data in this region are spread according to decisions made by certain elected nodes in that region. This variation is less susceptible to network changes and has better scalability but the drawback lies in the battery depletion of the elected clustered heads. Various optimization to the original approach have been described [7], [5], [4] after the introduction of the method, but the hardly heal the aforementioned drawbacks.

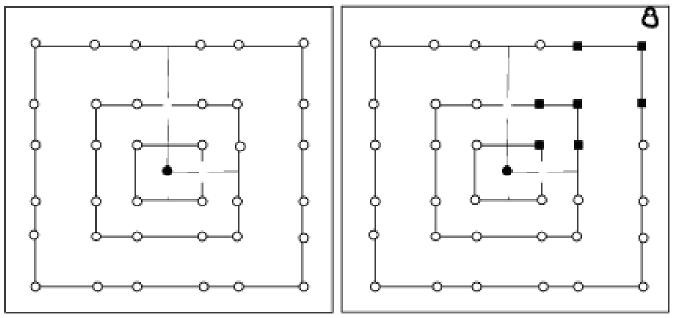

Figure 1. (Left) Location Centric Storage. (Right) Query Sensitive Storage.

LCS [14], [15] is a distributed storage protocol where each event is associated with in intensity value (σ), which determines the geographical propagation of the event record. The protocol then builds concentric circles of records with specific radii based on σ (see left part of Figure 1). The difference between LCS and the previous protocols is that none of the later take into consideration the query origin. In ES the user is querying the database in the base station. In LS data are stored locally and remain static throughout the execution of the protocol. Finally, in DCS data not only can be stored far from the originating query but also do not adapt to the query source.

III. Query Sensitive Storage

The key idea of the proposed QSS method, which is described in this section, is to propagate the records in such a way so that it is certain that an incoming query will be answered and also that it will be answered with the smallest latency possible. The protocol consists of two components: a) the storage subprotocol, and b) the querying subprotocol.

Storage management. The first part of QSS involves the propagation of the records. A record is built by the home sensor with an initial TTL and intensity σ values. The home sensor broadcasts the record whose intensity value will determine how far away from the home sensor it will travel. The sensors that receive the record will only store it if (a) their X coordinate ϵ $\{x+2^1-1, x+2^2-1, ..., x+2^\sigma-1\}$, and their Y coordinate $\epsilon\{y+2^1-1, y+2^2-1, ..., y+2^\sigma-1\}$, where σ and (x,y) are the intensity of the generated record and its originating coordinates respectively. If the two conditions are not met then the sensor will re-broadcast the record if its distance to the home sensor is smaller than 2^{σ}, otherwise it will drop the record.

The TTL value of the record starts to decrease the moment it is stored. When the TTL value reaches zero, the packet is dropped. Each time a sensor stores a record, it increases its TTL value by 10 seconds and then re-broadcasts it if the 2^{σ} condition is met. This means that the further away the packet travels, the more the TTL value increases resulting in more packets staying alive on the outer concentric circles. Therefore it is more likely that the user will be notified in time. This is clearly shown later in our experiments.

Query management. As far as the querying phase is concerned the protocol behaves as followed:

1) The queries are generated with an intensity value σ of their own. This value determines how far the query will travel away from the user issuing the query.
2) When a sensor receives a query, it checks to see if the answer is stored in its memory. If the answer does not exist then it decreases the intensity and re-broadcasts the query. If the answer record exists then it is sent back to the user.
3) Each sensor that receives this answer record on the way to the user, will store the record, increase its TTL value and then re-forward it to the user. This way the packet leaves a “trail" with increasing TTL values, while travelling to the user.
4) Each sensor has a small cache where a certain number of queries is stored. Whenever a new record is received, it is checked to see if it is an answer to a previously received query. If it is an answer, then the record is sent to the user, otherwise the normal procedure is followed. This small cache is updated with new queries every second.

A. Comparison of QSS to LCS

We mentioned earlier, that one of the main motives for the development of QSS is the need to make the sensornet to adapt also to the origin of the queries and not only to replicate the events around their “birth" location. Let us illustrate the major differences between QSS and LCS with a concrete example, i.e., the deployment of sensors in a minefield in search of situated mines. According to the LCS protocol, the sensors would propagate the data by placing it in concentric circles with the maximum radius being dictated by the σ value. The moment that data are stored at some sensor, its TTL value starts to decrease. When it reaches the value 0, data are dropped. This approach is depicted in the left part of Figure 1, where the sensor at the center (black dot) propagates the records, which in turn are stored at the positions denoted as white circles.

On the other hand, in the right part of Figure 1 we can see the effects of using QSS in the sensornet. At the beginning of the propagation, data are stored at the same positions as the initial LCS protocol dictates. The difference is that the TTL value of each record is increased as the packets travel further away from the source. This means that the packets on the outer circles will stay alive for longer and consequently users will have a better chance of being notified of a certain event. In our example, the user will have a greater chance of being notified of a nearby mine as he walks toward it.

Moreover, QSS is able to exploit the spatial locality of queries, i.e., a query for an event will probably be followed by another one for the same event. In our minefield example, since several army units pass through the minefield, it makes sense to try and notify faster the units that follow the first one. This procedure is pointed out in the right part of Figure 1 with the black dots. As the answer travels toward the user (at the upper right corner), data are stored at every sensor that it passes by without considering the rules that LCS protocol dictates, as to where the data is to be stored. Also, the TTL value of the answer data increases as the record travels toward the user.

IV. Evaluation of Query Sensitive Storage

We evaluated the performance of the QSS protocol through simulation experiments and compared its performance to that of the state-of-the-art storage protocol for surveillance sensor networks, namely LCS [14].

A. Simulation model

We have developed a simulation model based on the J-Sim simulator [13]. In our simulations, the AODV routing protocol is deployed to route the data traffic in the wireless sensor network. We use IEEE 802.11 as the MAC protocol and the free space model as the radio propagation model. The wireless bandwidth is assumed to be 2 Mbps. We performed a large number of experiments with various sensor network topologies, and parameter setting, but for the interest of space, we present here a subset of the experiments with a setting that is similar to that of [14].

The sensors were placed in two grids. The dimensions of the two grids were 23×23 and 32×32, having 529 and 1024 sensors respectively. The sensors in both grids were placed in the middle of each cell. The simulation details are as followed:

- The simulation time was 300 seconds. The records were produced in the first half of the simulation time while the queries were sent in the second half.
- The intensity σ of each record in the 23×23 grid was randomly chosen from [0,3]. The respective value in the 32×32 grid was [0,6]. The reason behind the change in values is that in a smaller network, having a large intensity value would mean that the record could be stored in a large area making the results difficult to interpret, since the originated records would be stored in almost each and every sensor of the sensornet.
- The TTL value was randomly chosen from [150,160] in the 23×23 grid, while the values in the 32×32 grid were [150,180]. A value smaller 150 would imply that the records would to be dropped inside the first half of the simulation.
- While the record is propagated in the network, its TTL value is increased by 10 seconds each time the record is stored at a sensor. As soon as a record is stored

Parameter	Default value	Values
# sensors	1024	529, 1024
λ_e	0.512	0.128, 0.256, 0.512, 0.768
λ_q	0.512	0.128, 0.256, 0.512, 0.768

Table I
SIMULATION PARAMETERS.

at a sensor, its TTL value starts to decrease by 1 on each second passed. When the TTL value reaches zero, the record is dropped and space is freed in the internal storage.

- The events and queries are generated according to a Poisson distribution with the rates λ_e and λ_q taking the values $0.128, 0.256, 0.512$ and 0.768.
- The queries originate at sensors whose geographical position follows the Zipfian distribution, i.e., some sensor generate more queries than others.

Our measured quantities include the number of stored records at each sensor in the network, the number of queries generated, the number of answers received, the query latency, the number of messages transmitted and so on. In the figures that follow, we plot the impact of the event generation rate λ_e on query latency, the effectiveness of the examined storage protocols described as the *Hit Ratio* (HR), i.e., the percentage of queries which receive an answer, and finally, the efficiency of the protocols in avoiding hot spots, defined as the ratio of HR to the maximum number of event records stored by any sensor during the course of the simulation. Table I summarizes the simulation parameters.

B. Evaluation

Impact of event generation rate on HR. Our first experiment investigated the influence of the event generation rate on the effectiveness of the competing storage protocols (Figure 6, Figure 7). QSS is practically insensitive to the event generation rate, and manages to deliver the same percentage of answers irrespectively of the size of the sensornet. It is 30% to 70% better than LCS and the performance gap widens in the case of larger networks. For large sensornets, the fact that LCS stores the events only around the neighborhood where they were generated results in a poor HR, whereas QSS exploits the spatial locality of queries to deliver high HR.

Impact of event generation rate on query latency. Our second experiment investigated the impact of the event generation rate on the experienced latency. The obtained results are illustrated in Figures 4 and 5. For small networks, when the event generation rate increases, so does the latency since more packets are travelling inside the network and congest it. For larger networks though, the situation is somehow different since there are alternative paths to avoid congested paths and latency decreases, until the event generation rate becomes really large and the network can not deal with the traffic. In summary, there is a threshold in the event generation rate that beyond that value, the sensornet becomes congested. In all cases, QSS outperforms LCS at factor that ranges from 15% to 100%.

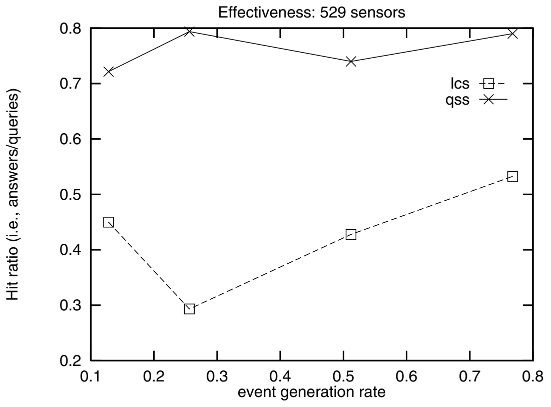

Figure 2. Hit ratio for 529 sensors.

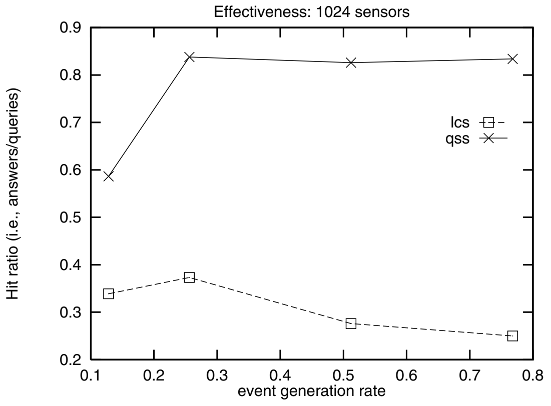

Figure 3. Hit ratio for 1024 sensors.

Finally, since QSS increases the number of stored event records inside the sensornet in order to deliver improved performance in terms of latency and hit ration, we investigated whether this increase in storage pays off. In Figures 2 and 3 we illustrate the ratio of HR to the maximum number of stored event records in any sensor network. We observe that only the network is large but produces very few events, is the LCS better than QSS, which is expected since QSS capitalizes

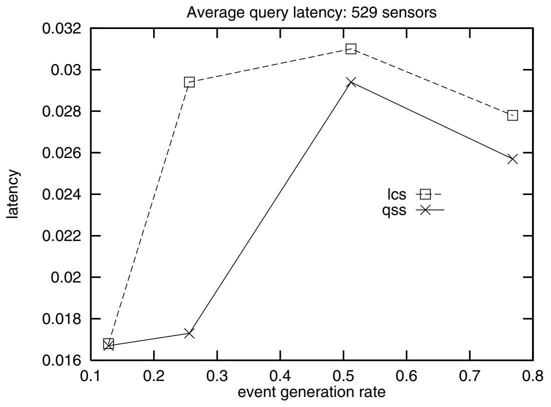

Figure 4. Latency for 529 sensors.

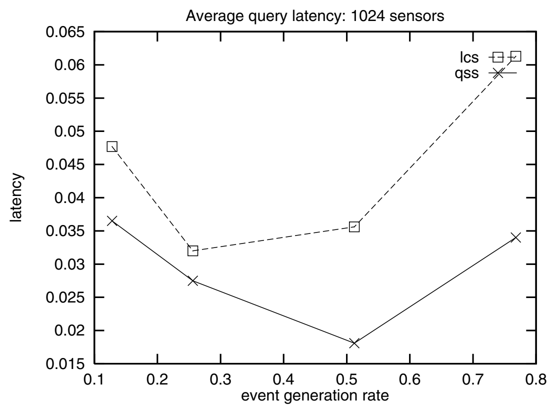

Figure 5. Latency for 1024 sensors.

on the spatial locality of queries in order to achieve high performance.

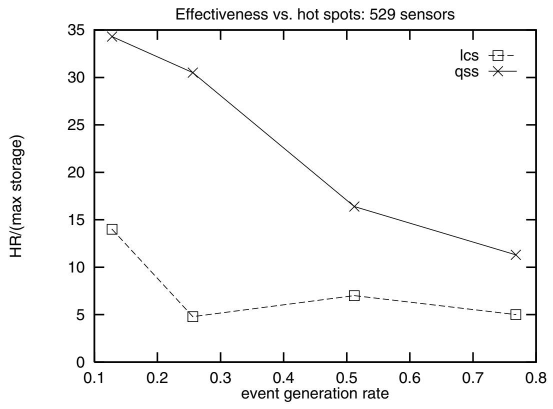

Figure 6. Protcol's efficiency for 529 sensors.

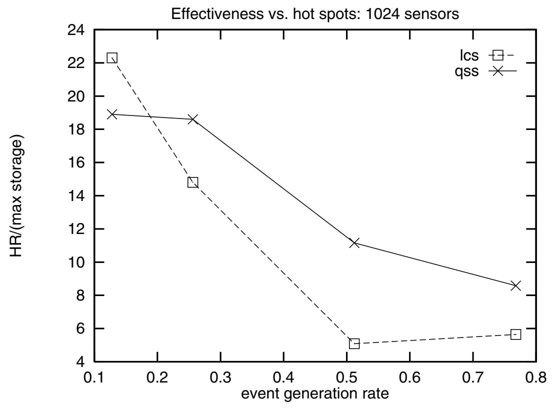

Figure 7. Protocol's efficiency for 1024 sensors.

V. Conclusions

Storage management in wireless sensor networks is an area that has started to attract significant attention. The need for storage management arises mainly in the class of WSNs where the information collected by the sensors need not or should not be relayed to outside-of-the-network observers at first place. The three canonical storage models, namely External Storage, Local Storage and Data-Centric Storage can not support efficiently such applications, whereas the recently proposed Location-Centric Storage is query-agnostic. The present article identified the drawbacks of the aforementioned storage methods and proposed the Query Sensitive Storage protocol, which capitalizes on the spatial locality of queries in order to deliver improved performance in terms of percentage of answered queries and latency. The proposed protocol was evaluated using an established simulation tool, i.e., J-Sim, and the experimental evaluation attested the superiority of QSS over LCS.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey of wireless sensor networks. *IEEE Communications magazine*, 40(8), 2002.

[2] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy. Rethinking data management for storage-centric sensor networks. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 22–31, 2007.

[3] M. Ding, D. Chen, A. Thaeler, and X. Cheng. Fault-tolerant target detection in sensor networks. In *Proceedings of the IEEE International Conference on Wireless Communications and Networking Conference (WCNC)*, 2005.

[4] D. Dudkowski, P. J. Marron, and K. Rothermel. An efficient resilience mechanism for data centric storage in mobile ad hoc networks. In *Proceedings of the IEEE International Conference on Mobile Data Management (MDM)*, 2006.

[5] C. T. Ee, S. Ratnasamy, and S. Shenker. Practical data-centric storage. In *Proceedings of the USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, pages 325–338, 2005.

[6] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2006.

[7] T. N. Le, W. Yu, X. Bai, and D. Xuan. A dynamic geographic hash table for data-centric storage in sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, volume 4, pages 2168–2174, 2006.

[8] Q. Li, R. Peterson, M. De Rosa, and D. Rus. Reactive behavior in self-reconfiguring sensor networks. *ACM Mobile Computing and Communication Review*, 7(1):56–58, 2003.

[9] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *Proceedings of the ACM International Conference on Information Processing in Sensor Networks (IPSN)*, pages 374–381, 2006.

[10] S. Ratnasamy, B. Karp, L. Yin, and F. Yu. Data-centric storage in sensornets with GHT, a Geographic Hash Table. *ACM Mobile Networks and Applications*, 8(4):427–442, 2003.

[11] S. Ren, Q. Li, Wang H., X. Chen, and X. Zhang. Analyzing object detection quality under probabilistic coverage in sensor networks. In *Proceedings of the International Workshop on Quality of Service (IWQoS)*, 2005.

[12] K. Seada and A. Helmy. Rendezvous Regions: A scalable architecture for service location and data-centric storage in large-scale wireless networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, pages 218–225, 2004.

[13] A. Sobeih, J. C. Hou, L.-C. Kung, N. Li, H. Zhang, W.-P. Chen, H.-Y. Tyan, and H. Lim. J-Sim: A simulation and emulation environment for wireless sensor networks. *IEEE Wireless Communications magazine*, 13(4):104–119, 2006.

[14] K. Xing, X. Cheng, and J. Li. Location-centric storage for sensor networks. In *Proceedings IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pages 492–501, 2005.

[15] K. Xing, X. Cheng, and J. Li. On the performance of location-centric storage in sensor networks. In *Proceedings IEEE International Conference on Wireless Algorithms, Systems and Applications (WASA)*, pages 79–86, 2007.