Adaptive Distributed Indexing for Spatial Queries in Sensor Networks

Vladimir Dyo and Cecilia Mascolo
Department of Computer Science, University College London, UK
Email: {v.dyo| c.mascolo }@cs.ucl.ac.uk

Abstract

Sensor networks have opened new horizons and opportunities for a variety of environmental monitoring, surveillance and healthcare applications. One of the major tasks of sensor networks is the distributed collection and processing of sensor readings over extended periods of time. We propose an energy-efficient hierarchical indexing approach for spatial data in sensor networks. Our indexing technique allows roaming users to navigate through sensor networks distributed over large geographical areas and to pose spatial queries about the location of the data in the network. The major challenge in designing such indexes is the minimization of the total amount of traffic needed to create and maintain the indexes, which is a function of region activity and the actual query rates. Given the dynamic character of the setting, these parameters might in fact change during the network operation, calling for a very adaptive solution.

1. Introduction

Sensor networks are ad hoc networks consisting of tiny low-powered computing devices with extremely restricted computational, communication and battery capabilities. Each device may be equipped with a physical sensor for reading temperature, sound, pressure or other physical phenomena and can operate both as a sensor and a wireless router. Scalability, self-configuration, ease of deployment and low cost have made sensor networks very attractive for a wide range of environmental monitoring, distributed surveillance, healthcare and control applications.

Energy is the most critical factor when designing sensor networks. Measurements suggest that sending 1bit is equivalent to performing approximately 1000 CPU instructions [10]. Therefore, any solution devised for sensor networks has to minimize the amount of communication overhead it imposes on the network. In this paper we present an energy-aware indexing approach for spatial data retrieval in sensor networks. There are a large number of applications where a requester needs to know the location of data sources available within a given geographical area. A typical query for this scenario could be "What are the locations of all critical events within a range R from a point with coordinates (X, Y)"? Also, similar information could be used by a mobile user to quickly find relevant information and efficiently navigate through geographically vast areas. Answering these questions would require a distributed index of data of entire network. Furthermore, given the variability of the data and query rates, an index would need to be dynamically reconfigured.

The problems of energy-efficient data retrieval and distributed indexing in sensor networks have been previously addressed in [4][11][13][14][16]. In general, there are two techniques that help creating and maintaining an index infrastructure: what we call a proactive approach and a reactive one. In *proactive mode*, sensors periodically report all events to a cluster head[4]. A cluster head maintains up-to-date information about all events in the area and provides a quick response to lookup queries. This approach has some drawbacks: excessive communication overhead in the case of dynamic data sources; also, an index has to be maintained even when there are no lookup queries. In *reactive mode* data is pulled from sensors only in response to an incoming query [15]. However, because the query has to be flooded over the network the cost of each query can be prohibitively high, so this technique is recommended only when the data is highly dynamic and the proactive approaches would not perform well.

In this paper, we present a distributed index that adapts to local event and lookup query rates to minimize the amount of energy needed to create and maintain the infrastructure. Our approach is based on quad-trees, which are created on demand. All data sources use a combination of proactive and reactive modes for index updates. Our design goal is to (i) minimize the amount of traffic needed to create and maintain a distributed index, by adapting to local event and lookup query rates and (ii) provide a bounded response time to lookup queries. To the best of our knowledge this is the first work on adaptive distributed indexing for sensor networks.

The rest of the paper is organized as follows: Section 2 formulates our assumptions and illustrates the system model. Section 3 describes the details of our approach. Section 4 illustrates our dynamic adaptation technique, while Section 5 contains a discussion and related work.

We conclude by a summarizing the results and by identifying the directions for future work.

2. Assumptions and System Model

We consider a static sensor network distributed over a flat area. All sensors are aware of their geographical position. Each sensor could be equipped with GPS device or use location estimation techniques such as [2]. We rely on geographical routing protocols such as GPSR [9] for routing packets. When sending the query to multiple nodes we assume the presence of existing multicast protocols [1] for efficient data dissemination. We do not make strong assumptions about spatial distribution of events, event generation rates and incoming queries rates. In our scenario, this means that certain network regions might have more dynamic data rates than others. It also means that certain network regions might receive more lookup queries than others.

We assume that sensors spend most of their energy on communication[10]. The communication cost depends on a number of factors such as transmission power, medium access protocol, channel conditions and network topology. For simplicity, the communication cost is calculated as linear function of geographical distance from the node to a cluster head and message size. We believe that this will capture the most salient features of the setting, while keeping the model clear.

3. Description of the Approach

We decompose the entire network into a set of disjoint hierarchical square cells, where each cell consists of four smaller cells and so on (Figure 1). A cell at each level of hierarchy has a cluster head responsible for indexing data in that cell. Cluster heads at the lowest level of hierarchy collect information directly from sensors. Similarly to [4], the availability of data in the cell is represented as a binary value and is forwarded to a cluster head at a higher level of a hierarchy in the form of a histogram. There can be several of such histograms, one for each event type in the network. Consequently, cluster heads at the highest level of hierarchy have a complete view of all types of events in the area.

Each cluster head is located inside its cell at a fixed relative location. In the example given in Figure 1 the cluster head is located in the centre of each cell. This location in the picture was chosen quite arbitrarily, but it has to be uniform throughout all cells. Since all the cluster heads are located at predetermined coordinates there is no need for cluster selection algorithms or advertising cluster head location. This allows us to create a distributed indexing hierarchy without any communication overhead. Given any pair of coordinates

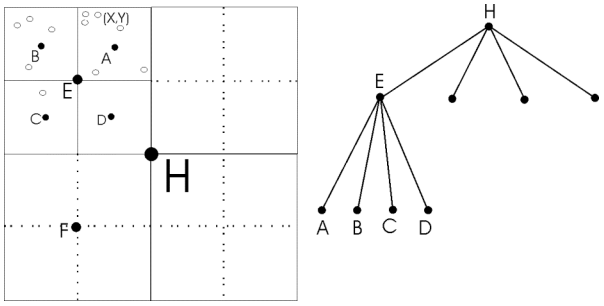

Figure 1. Space decomposition with hierarchical grids.

(X,Y), the location of all cluster heads responsible for this area, at any level of the hierarchy, is easy to calculate. In the example given in Figure 1, the cluster heads responsible for region (X,Y) are A, E, H, respectively. There is always a possibility that a predetermined geographical location does not contain any nodes or the node at that location has failed. In this case, a node which is geographically closest to a predetermined location is chosen as cluster head instead. A similar choice is made in [13].

We now show how the index tree is created and how it adapts to query and local event rates. Maintaining an index involves a certain amount of communication overhead as sensors keep sending all event information periodically to cluster heads. To minimize this overhead the index tree is created *on demand*; that is, it is created and maintained only for those parts of the network that do receive queries. The tree is also adaptive in the way it resolves incoming queries. Each branch of the tree can be in on the following three states: i) proactive ii) reactive iii) pruned. *Proactive* branches periodically send information about all the events to their parent. The parent cluster head then always has up-to-date information about the availability of data and can use that information to answer queries. *Reactive* branches do not send updates to their parents, however, they periodically send keep-alive requests which piggyback binary information about the availability of each data type in that region. *Pruned* branches are not active because they have never been used or there has been no recent activity on them. The difference between reactive and pruned modes is that, in reactive mode, cluster heads still send binary information about each event type to their parent. In pruned mode the cluster heads do not send any information to their parent, although they can still serve local queries. The state of each branch depends on the rate of event updates and lookup queries for data in the relevant cell. In general, our approach uses reactive mode requests for branches with high data update rates, proactive mode for children with less frequent update rates and it prunes inactive branches.

As an example, let us consider how a query is formed and resolved. A requester sends the query to a cluster head responsible for the minimum bounding square

covering the area of interest. As was noted earlier, the location of all cluster heads is predefined, so it should be easy to calculate. The query should contain information about an area of interest (in the form of a polygon), the type of events it is interested in and the maximum tolerated latency. Upon receiving a query a cluster head creates a tree, if this does not exist, and forwards the query down all the branches covering the spatial area of interest. If the tree does already exist, it checks the mode of each of its branches before forwarding the query. In case of proactive branches, there is no need to forward the query, as all information should be already available. The cluster head has to forward the query to reactive and pruned branches. For the reactive branches it will check if the branch contains the given data type before forwarding the query. As we have seen from the example, the tree branches can have different operating modes, depending on the rate of queries and event updates. The cluster head constantly evaluates those rates and switches to the mode which imposes the smallest communication overhead. Assuming that sensors are long-lived processes with relatively constant arrival rate, we need to identify criteria for mode switching for tree branches and for individual sensors. Our approach is presented in the next section.

4. Dynamic Adaptation

Our primary goal is to minimize the amount of communication overhead in the network. As noted earlier some sensors can be very active, whereas others can generate very few events. The basic idea of our approach is that each data source constantly compares the potential amount of traffic generated in proactive and reactive modes and always switches to the best mode. There are at least two times when sensors have to do that: i) after initial deployment to adapt to a local environment and lookup rates for data. ii) when there is a change in either the environment or the demand.

Let us illustrate our idea with an example shown on Figure 2. The sensors A, B, C and D are distributed over a large space and measure pollution levels in the air. An inspector visits an area daily and queries a local index F for the location of all sensors with abnormal pollution levels. Sensor D is located near a highway and therefore its measurements change more frequently, on average every hour. Sensors A, B and C are relatively stable and report changes on average once per day. It is easy to see that in this scenario sensors A, B and C should operate in proactive mode immediately reporting any changes to an index F whereas D should stay in reactive mode all the time. However, when the requirements of an environmental application change, for example, an inspector needs to take measurements every 5 minutes, then sensor D might need to switch to

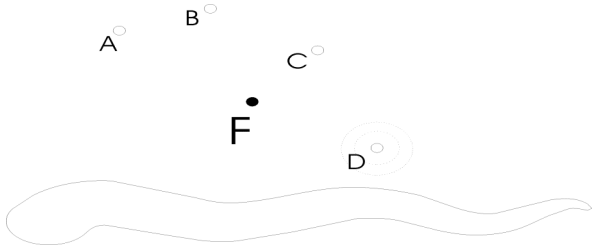

Figure 2. Measuring air pollution

proactive mode. In the other extreme case, if it turns out that the measurements have to be taken only once per week, then all of the sensors should switch to a reactive mode.

The rest of the section identifies a condition for switching between reactive and proactive modes in more detail. The communication cost of operating in proactive and reactive modes depends primarily on the rates of event update and incoming lookup. We assume that all data sources are long-lived processes with relatively constant arrival rates. In the example with air pollution, the readings of sensor D would change on average every hour depending on traffic on the road. Given this assumption, the communication costs of operation in proactive and reactive modes, for each individual data source, are:

$$C_{proactive(i)} = C_{up\,(i)}\,\lambda_{up\,(i)} \quad (1)$$

$$C_{reactive(i)} = \lambda_{req}\,(C_{up\,(i)} + C_{req\,(i)}) \quad (2)$$

Where $C_{up(i)}$ – cost of sending an update or reply to a query, C_{req} – cost of sending a request to a sensor, λ_{up} – average rate of event updates monitored by each node, λ_{req} – average rate of lookup queries received by a cluster head. The rate of incoming requests λ_{req} is the same for the entire cluster, whereas λ_{up} is different for each individual data source.

The ratio of costs of operation in proactive and reactive modes is:

$$T = C_{proactive} / C_{reactive} \quad (3)$$

Intuitively, when $T > 1$, the cost of operating in proactive mode is higher and the node should switch to reactive mode. Similarly, when $T < 1$ the node should switch to proactive mode. The sensors can use condition (3) to switch between reactive and proactive modes. To avoid the continuous flapping between the modes, data sources can switch to a reactive mode when the ratio T exceeds a certain upper threshold $\theta_{reactive}$ and to proactive mode when T goes below a $\theta_{proactive}$ threshold. The value of thresholds $\theta_{proactive}$ and $\theta_{reactive}$ will affect the level of system responsiveness. For example, a value of $\theta_{proactive}$ close to one will make the system very responsive to the slightest changes in query or event

update rates. On the other hand, higher values of $\theta_{proactive}$ would tell the system to switch to proactive mode only when it will lead to substantial benefits in terms of communication overhead. These thresholds can be programmed into the sensors at deployment time.

As we have just seen, our approach relies on the ability of data sources to estimate their average arrival rate of event updates $\lambda_{up(i)}$. This value might be evaluated over a certain time window, with more weight assigned to more recent values. In general, we cannot make any assumptions about the statistical properties of event and query rates, which makes the estimation of those parameters a difficult task. But in practice, the estimation algorithm could be application specific. For example, in case of linear stochastic processes, Kalman filter can be effectively used to predict the value of the next outcome [8]. This is particularly attractive for sensor networks because it is not computationally expensive, does not require storing of history information and therefore requires very little memory.

Let us look in detail on how this technique works. After deployment, the network starts operating in reactive mode to prevent indexing network data when it is not needed. When the cluster head can estimate the average request rate, it broadcasts an estimated lookup request rate in the cell. Depending on local event update rates, sensors then decide whether to select reactive or proactive mode; sensors which decide to switch the mode notify the cluster head on their decision. When resolving queries, a cluster head forwards the query to all reactive nodes in the cell. During operation a cluster head constantly monitors the incoming query rate. When λ_{req} crosses an upper threshold $\theta_{reactive}$ or lower threshold $\theta_{proactive}$ a cluster head sends notification to relevant sensors specifying a new request rate so that they can switch their mode. For example, when λ_{req} increases, some reactive nodes will need to switch to proactive mode. In case λ_{req} decreases, it would be more efficient for some proactive nodes to become reactive.

Finally we apply the same adaptation technique not only to individual data sources, but also to all cluster heads which have parent clusters. We now observe that each cluster head acts as a data source for its parent cluster which allows us to apply the same adaptation technique to each cluster head. Let us consider an example. The cluster head A at the lowest level of hierarchy receives lookup queries from its parent cluster head B. B estimates the incoming query rates and periodically sends this information to A so that A can decide whether to use proactive or reactive mode of updates. When the request rate drops below a certain threshold, B sends a notification request to A so that it switches to reactive or pruned mode.

5. Discussion and Related Work

In this section we discuss the advantages and some drawbacks of our approach and then discuss how it relates to other works in the field.

The combination of reactive and proactive approaches should perform better than proactive or reactive modes alone [5] because it will always choose a mode which generates less traffic. This optimisation comes at a certain cost. Firstly, the cluster head has to keep the state information about all reactive sensors in its area. This requires little memory as the state of the sensor can be represented as 1bit of information. The potential problem is that the state has to be recreated after a cluster head failure. Secondly, the adaptation technique introduces a certain amount of signalling traffic into the network. For example, a cluster head needs to notify data sources about changes in the incoming query rates. However, the amount of overhead can be minimized by adjusting the thresholds $\theta_{proactive}$ and $\theta_{reactive}$. Our solution should perform better when the rates are more predictable, as in this case, it will reduce flapping between the modes and minimize signalling traffic. Lastly, we plan to address the problem of cluster heads becoming disproportionately heavily loaded and exhausting their batteries: some kind of load balancing mechanism is needed to address this issue.

In general there is very little work dedicated to spatial indexing in sensor networks. However, the techniques that we use are somehow similar to those used in adaptive ad hoc routing protocols, location management for cellular scale networks and service discovery. Here we present a list of similar works and how they relate to our research.

Reactive and proactive route discovery have been heavily researched in the area of ad hoc networks. In [5] Haas proposes a hybrid routing protocol, which uses a combination of proactive and reactive modes. Some service discovery protocols [6] also use a a mixture of proactive and reactive approaches to cope with the variability of pervasive networks. We draw an analogy between discovering routes and discovering data in sensor networks and apply the same hybrid approach for building a distributed index for sensor networks.

There has been previous work on energy efficient distributed indexing and data retrieval in sensor networks. A centralized approach where all data are sent to a single place in the network suffers from scalability, reliability and excessive power consumption problems. Transferring updates across the entire network can quickly drain the energy of all the nodes along the routes. As queries might originate from any part of the network, an index placed at a specific location does not make much sense. Therefore, the index information should be located close to data to

minimize the amount of overhead to maintain an index. The authors of Directed Diffusion [7] propose a data-centric routing paradigm, where data is named using attribute-value pairs and the queries are expressed in terms of named data. A query sets up gradients showing a direction towards a sink as it is flooded throughout the network. Data matching the query start flowing along those gradients towards the sink. This approach works well when there is a need to collect all generated data from the network that match a certain interest. However, it relies on flooding to spread the query. Also, it assumes that the data need to be gathered in a static place in the network: this is not acceptable, given our scenario with roaming users.

In [13] Ratnasamy et all describe a novel Geographical Hash Table (GHT) system which hashes keys into geographical coordinates. In GHT the data is stored at a node with location determined by a geographical hash function of its name. The advantage of this system is that it allows to lookup the location of data by its name. Its problem is that the location defined by a GHT function can be quite far from the data source. Spatially related data might become scattered across the network which can also be a problem.

In DIFS [4] Greenstein et all propose a distributed index based on quad-trees optimised for efficient range queries. However, DIFS uses proactive updates which does not adapt to local conditions. We used some techniques from that work, such as using binary histograms for representing availability of data. The main difference is that our indexing structure is adaptive and, therefore, minimizes energy consumption. In [3] Demirbas et all propose an index for spatial queries based on distributed R-trees. We did not choose R-tree, but preferred quad-trees as they are more robust and offer a simpler solution.

To the best of our knowledge, none of the existing approaches adapts to the level of actual data update and data lookup query rates to minimize the amount of messaging overhead.

Conclusion and Future Work

We have presented a novel distributed index for spatial queries in sensor networks which adapts to local conditions and reduces the amount of overhead. A combination of proactive and reactive updates to maintain an index allows us to reduce the message overhead and, consequently, energy consumption.

Further work is necessary to address some of the design issues such as load balancing and selecting the optimal values of thresholds $\theta_{proactive}$ and $\theta_{reactive}$. Our next objective is to simulate this system using an ns-2 simulator, and compare its performance with existing approaches.

References

[1] S. Bhattacharya, H. Kim, S. Prabh, and T. Abdelzaher, Energy-Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks, 1st Int. Conf. on Mobile Systems, Applications, and Services (MobiSys), San Francisco, 15 CA, May 2003

[2] N. Bulusu, J. Heidemann, and D. Estrin, Gps-less low cost outdoor localization for very small devices. IEEE Personal Communications, Special Issue on Smart Spaces and Environments, 7(5):28–34, Oct. 2000. (Mobicom'00), pages 56–67, 2000.

[3] M. Demirbas and H. Ferhatosmanoglu, Peer-to-peer spatial queries in sensor networks. In 3rd IEEE Int. Conf. on Peer-to-Peer Computing, (P2P'03). Linkoping, Sweden, Sep. 2003.

[4] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, S. Shenker, DIFS: A Distributed Index for Features in Sensor Networks, In the Proc. of 1st IEEE Int. Workshop on Sensor Network Protocols and Applications, May 2003.

[5] Z. Haas, A new routing protocol for the reconfigurable wireless networks. In Proc. of the IEEE Int. Conf. on Universal Personal Communications, Oct. 1997.

[6] R. Harbird, S. Hailes, and C. Mascolo, Adaptive Resource Discovery for Ubiquitous Computing. In 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing, Toronto, Canada, Oct. 2004.

[7] C. Intanagonwiwat, R. Govindan and D. Estrin, Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks, ACM MobiCom'00.

[8] R. E. Kalman, A new approach to linear filtering and prediction problems. Transactions of the ASME Journal of Basic Engineering, Mar. 1960.

[9] B. Karp and H.T. Kung, GPSR: Greedy Perimeter Stateless Routing for Wireless networks. Mobicom'00.

[10] D. Culler, D. Estrin and M. Srivastava, Overview of Sensor Networks, IEEE Computer, Special Issue in Sensor Networks, Aug. 2004.

[11] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. OSDI, Dec. 2002.

[12] M. Musolesi, S. Hailes and C. Mascolo, Adaptive Routing for Intermittently Connected Mobile Ad Hoc Networks. In Proc. of IEEE 6th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM'05). Taormina, Italy. Jun. 2005.

[13] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, F. Yu, Data Centric Storage in SensorNets with GHT, a geographic hash table. Mobile Networks and Applications 8, 427–442, 2003. 2003 Kluwer Academic Publishers.

[14] E.M. Royer and C-K Toh, A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. IEEE Personal Communications, Apr. 1999.

[15] TinyDB project, http://telegraph.cs.berkeley.edu/tinydb/

[16] Y. Yao and J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks," SIGMOD'02.