CSN: A Network Protocol for Serving Dynamic Queries in Large-Scale Wireless Sensor Networks

Muneeb Ali and Zartash Afzal Uzmi
Computer Science Department, LUMS
{muneeb,zartash}@lums.edu.pk

Abstract

A fundamental problem that confronts future applications of sensor networks is how to efficiently locate the sensor node that stores a particular data item. It is known that distributed hash table (DHT) based Internet peer-to-peer (P2P) protocols provide near-optimum data lookup times for queries made on networks of distributed nodes [2, 23-25]. A generic mapping of these protocols to sensor networks is, however, perceived as difficult [1]. We present a novel DHT based network protocol for sensor networks—Chord for Sensor Networks (CSN)—for which bounded times for data lookup, in the order of $O(\log N)$ messages, can be achieved in an energy efficient manner. CSN makes system lifetime of the sensor network proportional to its effective use. Furthermore, CSN scales well to large-scale sensor networks when the information about other nodes logarithmically increases with an increase in the number of sensor nodes.

1. Introduction

Advances in IC fabrication, RF design, and MEMS-based technology have made it possible to integrate DSP and sensing in a single chip with low development cost [9-10]. These chips or sensors observe a physical phenomena and report data about that phenomena to the interested observer. A sensor generally consists of sensing hardware, memory, battery, embedded processor, and transceiver [14]. A network of these sensors, called a sensor network (SN), bridges the gap between the physical and the computational world by providing reliable, scalable, fault tolerant, and accurate monitoring of physical phenomena. There are many applications of SN including reliable location tracking, battlefield surveillance, habitat monitoring, machinery prognosis, and bio-sensing [11, 12]. From an architectural perspective, the organization of SN is tri-layered consisting of physical, network, and application layers [13].

In previous work, it is usually assumed that sensors report data at a fixed rate [17-20]. Near-optimum network layer protocols for such preprogrammed sensors are well known [3-6], while the area of querying a sensor network is much less explored. These sensors do not store the data and are preprogrammed to report data after a specific time interval. A drawback of this assumption is that end users cannot change the system behavior on the fly without causing significant overhead [15]. Further, if the actual queries from the user are not frequent, considerable battery power is wasted in transmitting data that the user does not require. Sensors reporting data only in response to a user query, instead of reporting data at a constant rate, overcomes these problems and, with other things constant, makes system lifetime proportional to its effective use. There have been attempts at applying concepts from the database community to assist in querying a SN [15-16] but those concepts are applicable to the application layer while the focus of our work is on network layer support for queries.

Using a DHT based network protocol to serve queries in SNs becomes an important problem to study as a) It holds the potential to outperform the existing SN models in terms of performance metrics like latency, system life time, scalability, and fault tolerance b) Bounded times for data lookups could be achieved in SNs and guarantees could be provided to the applications running atop. However, a generic mapping of DHT based Internet protocols to SNs is considered difficult as: a) These protocols typically interconnect nodes independently of their proximity in the physical network topology which is not suitable for energy-constrained sensor networks as neighbors in the DHT logical identifier space may actually be far apart and each logical hop within a DHT may cost energy of many packet transmissions [1] b) In the energy-constrained SN environments, particularly in large scale SNs, maintaining routing information among all pairs of nodes becomes expensive [1] c) The unique IP-address of each node in the Internet is used for obtaining unique node identifiers, for each node, required by DHT Internet protocols [2, 23-25] whereas individual sensors are generally not named in SN and there is no concept of unique IP-addresses for each sensor.

We propose a DHT based network layer protocol CSN,

Proceedings of the Second Annual Conference on Communication Networks and Services Research (CNSR'04)
0-7695-2096-0/04 $20.00 © 2004 IEEE

with which bounded times on data lookup, for dynamic queries, can be achieved in a SN. We show that CSN handles all the above mentioned issues and takes the SN research a step forward. CSN follows the design principles of Chord [2], an Internet peer-to-peer (P2P) system supporting data lookup. CSN is a hierarchical clustering approach; each cluster in CSN is formed in a logical ring formation. Each CSN sensor node maintains a finger table of $O(logN)$ other nodes in the ring and resolves all lookups by sending $O(logN)$ messages to other nodes.

The rest of the paper is organized as follows. Section 2 gives a brief background on sensor networks. Section 3 identifies the SN protocol evolution that leads to CSN and section 4 outlines some technical tradeoffs. Section 5 presents the CSN protocol in detail. Section 6 presents simulation and testing results. We outline future work and summarize conclusions in section 7.

2. Background on Sensor Networks

SN environments inherently different from the Internet, pose some unique challenges to systems researchers. Systematic deployment of sensor networks (e.g., in a linear or grid arrangement) is not common. The ad-hoc deployment implies that sensors should themselves be able to cope with the distribution and form communication paths. Once the sensors are deployed they remain unattended, hence all operations e.g. topology management, data management etc. in a SN should be automatic and should not require any manual or remote assistance. Further, the environments in which sensor networks are deployed are dynamic, possibly hostile, in nature. Thus, a SN should be adaptive and should be able to cope with node or communication failures. Sensors, in a SN, are energy constrained and are regarded as dead once the battery power is insufficient to carry out computation and communication. For this reason, the network protocols need to be optimized for energy consumption. In addition to limited energy the bandwidth available to the sensors is also limited and thus large amounts of data cannot be communicated. Based on the data delivery method, sensor networks can be classified as [14]; a) *Continuous:* Sensors communicate their data, at a constant rate, to the base station b) *Event-driven*: Sensors report information only if an event of interest occurs c) *Observer-initiated*: Sensors only report data as a response to an explicit query from the user.

3. Related Work

With each node flooding data in the network, as done in *classic flooding*, a node can receive the same information from two or more different paths; called *implosion* problem. Another problem, known as *overlap*, occurs when the geographic monitoring area of two or more sensors overlaps. Information from such sensors will have some common information. The main contribution of SPIN [17] is to overcome redundant information transfer caused by implosion and overlap. SPIN uses meta-data keys to describe data. These meta-data keys are unique for unique data and are same for similar data from different nodes. Every SPIN node advertises its meta-data keys, based on these advertisements a node requests only the data that it does not have. CSN overcomes implosion and overlap problems in a manner similar to SPIN.

Local clustering of sensor nodes, with one node as the head, performs better than non-clustering approaches (e.g. classic flooding) as it makes local coordination among nodes more efficient. Further, as number of nodes increase in a SN, clustering can result in increased scalability. Estrin *et al.* [18] gave a two level clustering algorithm that can be extended to build cluster hierarchies. We refer to this kind of clustering as *static clustering* because the responsibility of being a cluster head is not rotated among the cluster member nodes. Static clustering has the possible drawback that cluster heads die quickly as they consume more energy than normal nodes. LEACH [19] proposed a solution to this problem by randomly rotating the cluster heads within a cluster. This divides the high energy consumption tasks of cluster heads among all nodes and results in extending the lifetime of the SN.

PEGASIS [20], is proposed for the same SN environment as LEACH. Each node in PEGASIS identifies its closest neighbor by sending a power signal to its neighbor nodes and gradually reducing the power signal till it is heard by only one node. Each node then communicates only with its closest neighbor and takes turns, in a round robin fashion, to transmit data to the base station. This reduces the power required to transmit data per round and hence further increases the lifetime of the SN. In TEEN [21], the sensed data value is compared with a user defined threshold in order to decide if the data should be transmitted or not. TEEN identifies that LEACH and similar protocols are not suitable when the data delivery model is event-driven. We extend this argument by proposing that these protocols are also not suitable when the data delivery model is observer-initiated. *Directed diffusion* [22], differs from the above mentioned protocols as the data delivery model is observer-initiated. In directed diffusion, each sensor names data that it generates using one or more attributes. A sink node is the node from where the interest, similar to the concept of query, about certain data is generated. Intermediate nodes locally propagate these interests towards the data source at each hop using geographic information. Interests establish gradients of data towards the sink that expressed that interest and hence data is transmitted to the sink using

such gradients. As the nodes only have local knowledge, the same data or same interest could be transmitted to a node from multiple different paths. Directed diffusion attempts at minimizing these redundant paths but the effects of such attempts are not optimum. CSN does not suffer from such redundancy as there is only one path of query propagation and data delivery.

All above mentioned protocols except directed diffusion assume that data is stored external to the SN, while directed diffusion stores data locally. The main thesis of *GHT* [1] is that if data within the network was stored according to the type of data then queries for data could be directly sent to the node storing the named data. This work is unique in the sense that it uses the concept of efficiently querying a SN by having some routing information other than the geographic location. They do a cost analysis of external storage, local storage and data-centric storage to show that data-centric storage is most efficient for querying. However, we believe that under certain constraints, local storage may prove to be more efficient than data-centric storage.

4. Technical Tradeoffs

There are many distributed hash table (DHT) based Internet protocols and systems available e.g. Chord, CAN, Pastry, Tapestry, Freenet, Gnutella, Oceanstore, and Ohaha system [2, 23-25, 7-8, 27-29]. We decided to follow the design principles of Chord over other systems because Chord provides strong guarantees, unlike many systems, and its lookup function runs in predictable time and always results in success or definite failure. Some other systems, e.g. Oceanstore [28], provide stronger guarantees than Chord but Chord is substantially less complicated and is better in handling concurrent node joins and failures. We believe that in the possibly hostile sensor network environment successfully recovering from node failures is very important and Chord was the best available choice that we had for achieving this. Further, scalability of Chord was another factor that helped us making the decision of choosing Chord over systems like Gnutella [27] that makes widespread use of broadcasts. In deciding between having a clustering or non-clustering network protocol there was not much debate as local clustering, with one node as the head, makes local coordination among nodes more efficient and results in increased scalability [18].

5. The CSN Protocol

CSN has two modes of operation; the energy-efficient mode (EE_{mode}) and the robust mode (R_{mode}). We consider an environment in which sensors are deployed in an ad-hoc manner, the base-station is located far away from the sensors, data delivery model is observer-initiated and the sensors are static.

5.1. Initial Setup

In sensor networks, each node needs to communicate with its nearest neighbor in order to optimize energy efficiency. This constraint of SN environments implies the following problem:

The Ring Problem: We want to form a logical ring of sensor nodes where the clockwise successor of each node is geographically closest to it.

We propose two approximate solutions to the NP-complete ring problem: the chain method and the set-average method (SAM). The chain method makes the successor of each sensor α_i in a cluster C the geographically closest sensor available to it. This means that when α_i communicates with $\alpha_i.successor$, the least possible amount of energy would be spent. This property holds true $\forall\alpha_i$, except for the link where the last sensor α_m makes the first sensor α_1 its successor. This gives rise to a problem that α_m and α_1 of the chain would die quickly when compared to $\forall\alpha_i : i = 2,3,\ldots,m-1$, as they would be communicating among themselves over a possibly larger geographical distance, as shown in figure 1. A slight modification in the chain method can remedy this problem. If we mark the link $\alpha_m.successor$ as high energy consuming and make α_m communicate with α_1 by routing messages anti-clockwise in the ring, then at each hop the least amount of energy would be consumed.

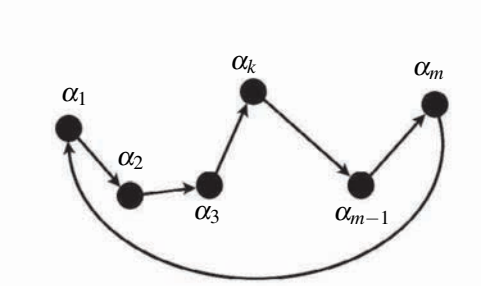

Figure 1. α_1 and α_m should not communicate directly

The set-average method, more suitable for the R_{mode}, tries to minimize the distance of each cluster node with all of the member cluster nodes. The set-average method does not guarantee that the successor of each node α_i would be the node geographically closest to it. This guarantee is not needed as in R_{mode} a node α_i seldom communicates with $\alpha_i.successor$, rather messages are directly sent to nodes occurring farther away in the logical ring arrangement. Both R_{mode} and EE_{mode} work correctly regardless of what method was used in setting up the cluster. However,

Chain Method for level 0*(Base Station* β*, Set of all Sensors* π_0*, Maximum no.of Sensors per Cluster* λ_0*)*

1. cluster head $\omega = \min(\beta)$
2. if ($\pi_0 == \delta_0$) exit, where initially $\delta_0 = \emptyset$
3. sensor $\alpha_i = \omega$
4. put α_i in sets τ and δ_0, where initially $|\tau| = 0$
5. while($|\tau| \leq \lambda_0$)
6. $\quad \alpha_i.successor = \min(\alpha_i)$
7. $\quad \alpha_i = \alpha_i.successor$
8. $\quad$ put α_i in sets τ and δ_0
9. $\alpha_i.successor = \omega$
10. $\omega = \min(\alpha_i)$
11. goto 2

Figure 2. Chain method algorithm for clustering hierarchy level 0

if the expected mode of operation is R_{mode} then the initial setup should be done according to the set-average method. Similarly if the expected mode of operation is EE_{mode} than the chain method should be preferred.

Chain Method: Let levels of cluster hierarchy be $0, 1, \ldots, m$, set π_i = {all sensors participating in level i} and set δ_i = {all sensors currently part of any cluster in level i}. Initially $\delta_i = \emptyset$. Further, π_0 = {all deployed sensors}. Let $\alpha_i =$ sensor under consideration, $\beta =$ base station and let $\alpha_i.successor$ = the clockwise successor of α_i in the ring arrangement cluster. The function $\min(X)$ gives a sensor at minimum distance from X, where X could either be a sensor or the base station. The $\min(X)$ function calculates the minimum sensor in a manner similar to PEGASIS [20], by first sending out a high power signal from X and then gradually decreasing the signal until it is heard by only one sensor. Let $\lambda_i =$ the maximum number of sensor nodes per cluster at level i, and let $\omega =$ the cluster head of cluster C. Let $\tau =$ set of sensors included in the cluster C, and $|\tau| =$ the number of elements in set τ. The chain method algorithm for level 0 is given in figure 2.

Set-Average Method: In addition to the symbols defined in the chain method, let $\nu_k =$ a list, and $\varepsilon =$ a set. The function $\min(X, M)$ is similar to function $\min(X)$ described above but instead of giving the closest sensor, it gives a list of the closest M sensors to X. The set-average method (SAM) algorithm for level 0 is given in figure 3.

5.2. Cluster Head Rotation

Unlike LEACH [19], cluster head rotation in CSN takes place in a round-robin fashion. Also there is no concept of rounds [19] in CSN and once a cluster is formed at the initial setup time, the cluster formation algorithms need not

Set-Average Method for level 0 *(Base Station* β*, Set of all Sensors* π_0*, Maximum no.of Sensors per Cluster* λ_0*)*

1. cluster head $\omega = \min(\beta)$
2. if ($\pi_0 == \delta_0$) exit, where initially $\delta_0 = \emptyset$
3. list $\nu_1 = \min(\omega, \lambda_0)$, and let $m = \lambda_0$
4. for $k = 2, 3, \ldots, m$:
5. $\quad \nu_k = \min(k_{th}$ element of $\nu_1, \lambda_0)$
6. let ν be a set of lists
7. $\nu = \{\nu_1, \nu_2, \ldots, \nu_m\}$
8. now, $1 \leq$ occurrence of a sensor α_i in $\nu \leq \lambda_0)$
9. let set $\varepsilon = \emptyset$, and variable $x = \lambda_0$
10. while($x \geq 1$ AND $|\varepsilon| \leq \lambda_0$),
11. $\quad$ for all sensors in ν:
12. $\quad\quad$ if (occurrence of sensor α_i in $\nu = x$)
13. $\quad\quad\quad$ insert sensor α_i in set ε
14. $\quad$ decrement x
15. redefine $\min(X)$ to only return a sensor $\in \varepsilon$
16. do step 3 to step 10 of the chain method
17. goto 2

Figure 3. SAM algorithm for clustering hierarchy level 0

be run again and again. Each cluster head, of cluster C at level i, makes its clockwise successor in the ring the next head when it has consumed one energy quantum E_q given by,

$$E_q = E_m/\mu \tag{1}$$

where $E_m =$ the maximum energy of the sensor and $\mu =$ a constant. By increasing the value of μ we can make cluster head rotations more frequent and vice versa. The cluster head rotation method of CSN has the benefit that no head rotation takes place in the time when the SN is inactive i.e. there are no or few queries from the application. Further, the rotation method ensures that a node is re-assigned as head after each other member node in the cluster has served as head exactly once.

5.3. Hierarchical Clustering

The chain method and set-average method, given in figure 2 and figure 3 respectively, could easily be used to form higher layers of clusters. Each cluster head from the lower layer participates in the higher layer, as a super sensor Θ_i, and thus the total number of sensor nodes participating in higher layers reduces as we go up the hierarchy. The nodes participating in the higher levels would need to store more data and it is desirable, because of memory limitations, to distribute this data over a larger number of nodes per cluster. Further, as the geographical distance among higher layer cluster members increases as we go up the hierar-

chy, it is desirable to have more nodes per cluster in order to minimize the geographical distance between neighbor nodes in the higher layer clusters. Let, the levels in the cluster hierarchy be $0, 1, \ldots, m$, λ_i be maximum number of sensor nodes in a cluster C at level i, and N_i be the total number of sensor nodes participating in level i. The number of clusters φ_i at level i is given by,

$$\varphi_i = N_i/\lambda_i \tag{2}$$

Hence, the total number of clusters per layer decrease as we go up the hierarchy. The hierarchical clustering methods of CSN guarantee that,

$$N_0 > N_1 > \ldots > N_k > \ldots > N_m \tag{3}$$

$$\lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_k \leq \ldots \leq \lambda_m \tag{4}$$

$$N_0/\lambda_0 > N_1/\lambda_1 > \ldots > N_k/\lambda_k > \ldots > N_m/\lambda_m \tag{5}$$

$$\varphi_0 > \varphi_1 > \ldots > \varphi_k > \ldots > \varphi_m \tag{6}$$

5.4. Energy-Efficient Mode vs. Robust Mode

While operating in robust mode (R_{mode}), CSN functions similar to Chord [2] and can resolve lookups with a bound of $O(log N)$ messages. But this efficiency in data lookup is achieved at a cost of greater energy consumption, as each node is directly sending messages to a node possibly farther away from it. The R_{mode} is used only if the application delay requirements cannot be met by the energy-efficient mode (EE_{mode}). In the EE_{mode} the logical operation of sending a message to a node is separated from its actual routing and the routing of messages takes place by each node communicating only with its closest neighbor node.

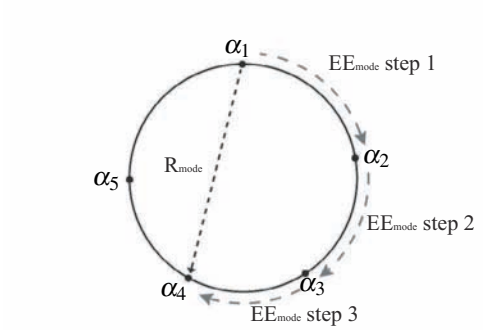

Figure 4. Difference of R_{mode} and EE_{mode}

Consider the example shown in figure 4. The cluster formation methods guarantee that $distance(\alpha_1, \alpha_4) \geq distance(\alpha_1, \alpha_2)$. Hence, if α_1 decides to send the message directly to α_4, as would happen in R_{mode}, the number of hops in the message delivery would be minimum but at a cost of greater energy consumption both by α_1 and α_4. On the other hand according to EE_{mode}, α_1 would route the message to α_4, through α_2 and α_3. The number of hops in the message delivery would be greater than R_{mode} but the energy spent at each hop would be less.

EE_{mode} is the default mode of operation and CSN switches to the R_{mode} only if the application delay requirements cannot be met by EE_{mode}. The application allowed delay, say d, is compared with a delay threshold value Γ_d and $if(d \leq \Gamma_d)\, R_{mode}\, else\, EE_{mode}$. With pure R_{mode}, all logical messages to a node are directly delivered to it. Whereas in pure EE_{mode} all logical messages to a node are indirectly routed; by each node passing the message to it's nearest neighbor. A hybrid form of EE_{mode} and R_{mode} is also possible. In the hybrid form each node makes a local decision between using the R_{mode} or the EE_{mode}.

5.5. Incremental Setup vs. Parallel Setup

The cluster head selection in parallel setup is similar to LEACH [19]. Each node α_i chooses a random number $0 \leq \gamma \leq 1$ and if$(\gamma > \Gamma_h)$ α_i decides to become a cluster head, where Γ_h is a threshold value. Γ_h is set as the required number of cluster heads at the current level i e.g. $\Gamma_h = 0.06$ gives 6% of the total nodes as cluster heads. In this example, $\forall \alpha_i$ probability of becoming a cluster head is 0.06. Once the cluster heads are decided, either chain-method or set-average method could be used to form the logical ring clusters. Unlike incremental setup, in which one cluster formation triggers the cluster formation process of the other, in parallel setup all cluster formations are carried out simultaneously.

Incremental setup takes more time but guarantees that the optimum energy efficient cluster formation would take place. On the other hand, parallel setup forms the clusters quickly but the optimum cluster formation might not take place as the cluster heads may not be uniformly distributed in the network. The decision of what mode of cluster formation to use is application specific.

Theorem 5.1 *Chain method with incremental setup takes $O(N)$ time; N = total no.of sensors in the network.*

Proof: Let the levels of cluster hierarchy be $0, 1, \ldots, m$, λ_i be the maximum number of sensors per cluster at level i, and N_i be the total number of sensors participating in level i. We know from the chain method algorithm that in forming any cluster C, each node $\alpha_i \in C$ is contacted only once. Therefore $\forall C$ in level i, time taken in forming one C, say $t_j = \lambda_i$. Let the clusters at level i be $1, 2, \ldots, p$. From equation 2 number of clusters at level i, $\varphi_i = N_i/\lambda_i$. Hence,

$$T_i = \sum_{j=1}^{p} t_j = \lambda_i \times N_i/\lambda_i = N_i.$$

$$\textit{Total Time} = \sum_{i=1}^{m} T_i = N_0 + N_1 + \ldots + N_k + \ldots + N_m.$$

where N_m denotes the number of sensors participating in the highest level of cluster hierarchy, and T_i gives the time for forming all clusters at level i. We know from equation 3 that $N_0 > N_1 > \ldots > N_k > \ldots > N_m$. Therefore,

$$O(N_0 + N_1 + \ldots + N_k + \ldots + N_m) \simeq O(N_0)$$

As, at level $0 : N_0 = N$, hence time taken by incremental setup with chain method $= O(N)$

Theorem 5.2 *Set-average method with incremental setup takes $O(N)$ time; N = total no.of sensors in the network.*

Proof: In set-average method each sensor α_i is contacted twice whereas in the chain method $\forall \alpha_i$ is contacted only once. Following the same arguments as in the proof of theorem 5.1, for set-average method,

$$\textit{Total Time} = \sum_{i=1}^{m} T_i = 2 \times N_0 + 2 \times N_1 + \ldots + 2 \times N_k + \ldots + 2 \times N_m \simeq O(N_0) = O(N)$$

Hence time taken by incremental setup with set-average method $= O(N)$

The parallel setup reduces the cluster setup time, of both chain and set-average methods, at each level i by λ_i. Therefore similar to proof of theorem 5.1, the parallel setup time T_i for setting up all clusters in a level i is given by,

$$T_i = \sum_{j=1}^{p} t_j = N_i / \lambda_i \tag{7}$$

5.6. Naming of Sensor Nodes

The hashing of nodes in CSN does not require the nodes to know their geographical location, instead CSN assigns unique names to nodes in a scalable manner. There are two ways in which this naming of nodes could take place:

Incremental Naming: In incremental naming the naming of nodes takes place in a bottom-up fashion during the cluster setup phase. Let i be current level of cluster hierarchy, ρ be a variable initialized randomly, λ_i be the maximum number of nodes per cluster at level i, τ be the set of sensors included in the cluster C, and $|\tau|$ give the number of elements in τ. Initially, $\tau = \emptyset$. The first cluster head, in chain method or set-average method, ω = min(*base station* β), as described in figure 2and figure 3. This unique sensor is assigned the name $[\,i.\rho.|\tau|\,]$. Generally, $\forall \alpha_i \in C$: name of $\alpha_i = [\,i.\rho.|\tau|\,]$, a sensor is placed in set τ after it has been named, and when the completion of one cluster formation triggers the formation process of another; ρ is incremented. Example node names assigned by the node naming algorithm could be 0.23.5, 0.25.2, 2.37.0 etc. Lower level details on the specific name address format are beyond the scope of the paper.

Parallel naming(Maximum level of hierarchy m, Maximum no.of Sensors per Cluster λ_i)

1. Initially, variable $k = m$
2. sensor α_i = only super sensor Θ_i at level m
3. α_i.name = $[\,k.\rho.|\tau|\,]$, initially $|\tau| = \emptyset$
4. $\forall \alpha_i \in$ level k, ρ = hash(name of α_i)
5. if($k = 0$) exit
6. $k = k - 1$
7. $\forall \alpha_i \in k$ that have ρ initialized, do in parallel:
8. while($|\tau| \leq \lambda_k$)
8. $\alpha_i = \alpha_i.successor$
9. α_i.name = $[\,k.\rho.|\tau|\,]$
10. put α_i in set τ
11. goto 4

Figure 5. Parallel naming algorithm

Parallel Naming: The parallel naming starts, after the cluster hierarchy has be formed, by naming the sensors in the highest layer and then simultaneously triggering the naming process of the all lower-layer clusters. In addition to the terms defined in the incremental naming algorithm, let k be a variable denoting the current level of clustering hierarchy. Parallel naming algorithm is described in figure 5. The incremental naming has no extra overhead of naming the nodes as the naming phase is completed with the setup phase. However, parallel naming has some overhead as it starts after the parallel setup of clusters is completed. Similar to equation 7 the parallel naming time at any layer i is: $T_i = N_i / \lambda_i$.

Without unique naming, CSN can not perform correctly. Therefore, it is important that the naming algorithms guarantee that each node would be assigned a global unique name. Let A.B.C represent the name format of the sensor nodes. From the naming algorithms it could be followed that C would be unique for each sensor within a cluster, B would be unique for each cluster within a layer, and A would be unique for each layer. Hence, the combination A.B.C would be globally unique for all sensors.

5.7. Hashing of Nodes and Keys

Once the naming of the nodes is complete, the hashing of nodes based on these names takes place. This hashing is similar to that of Chord [2] but instead of using IP-addresses for hashing the nodes, we use the names assigned to the sensor nodes at setup time. The hashing of keys takes place in CSN locally at each cluster C at level i. Hashing of keys in CSN is also similar to Chord [2]. Each node lo-

cally names the data it observers and generates a meta-data key associated with the named data. Hashing this meta-data key produces a *key identifier*, while hashing the name of the node produces a *node identifier*. We use the SHA-1 [26] hash function, both for hashing the keys and the nodes. The key identifiers are assigned to nodes in a manner, that *key identifier* $\leq$ *node identifier*. Each node $\alpha_i \in$ cluster C at level i, maintains information of $O(log\,\lambda_i)$ other nodes in the cluster in its local finger table, where λ_i is the number of nodes in the local cluster at level i. The key identifier can be made globally unique by including the globally unique name of the node in the hash function.

5.8. Lookup Operation

Each sensor say α_i participating in highest level m, can listen to user queries. The user sends its query to the $\alpha_i \in$ level m geographically closest to it. According to R_{mode}, the sensor $\alpha_i \in$ level m upon receiving the query first checks if the queried key is present in its own local storage. If the key is not found then, α_i checks its local finger table. The finger table of each node α_i contains data information about $O(log\,\lambda_i)$ other nodes. If the finger table also does not contain the required key then the query is sent to the node closest to the target according to the finger table. Say α_k is that node. The same procedure is repeated at any such α_i. With hierarchical clustering, when a key mapping is found by a node say α_f at level $i \neq 0$, α_f then performs the lookup operation in its member cluster at level $i-1$. This procedure is repeated till we reach level 0, in which case the result of the lookup operation would locate the node actually storing the data associated with the queried key.

Theorem 5.3 *Lookup operation in R_{mode}, say T_R, would take $O(log\,N)$ time; N = no.of nodes at the maximum hierarchy level.*

Proof: We know that the lookup operation within a single ring with N nodes would take $O(log\,N)$ time [2]. In the R_{mode} of CSN m number of ring clusters would be searched before locating the key, where m is the number of hierarchical layers. λ_i is the maximum number of sensor nodes in a cluster C at level i. Then,

$$T_R = O(log\,\lambda_1 + log\,\lambda_2 + \ldots + log\,\lambda_k + \ldots + log\,\lambda_m)$$

From equation 4,

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_k \leq \ldots \leq \lambda_m$$
$$log\,\lambda_1 \leq log\,\lambda_2 \leq \ldots \leq log\,\lambda_k \leq \ldots \leq log\,\lambda_m$$

Therefore,

$$T_R = O(log\,\lambda_m + log\,\lambda_m + \ldots + log\,\lambda_m + \ldots + log\,\lambda_m)$$
$$T_R = O(m \times log\,\lambda_m)$$

At level m there is only one cluster, hence $N = \lambda_m$. Also m has a constant upper bound therefore, $T_R \simeq O(log\,\lambda_m) = O(log\,N)$ Hence, time taken by CSN in R_{mode} is $O(log\,N)$, where $N = \lambda_m =$ number of nodes at maximum hierarchy

In the EE_{mode} the logical operation of sending a message is separated from its actual delivery and a logical message from a sensor say α_i sent to another sensor say α_k, is routed along the energy-efficient path. This energy-efficient path is setup by each sensor α_i sending the message to its $\alpha_i.successor$ only. The messages can travel up and down in the cluster hierarchy as well.

Theorem 5.4 *Lookup operation in EE_{mode} would take $O(M \times log\,N)$ time; N = no.of nodes at the maximum level of hierarchy , and M = maximum path length of the energy efficient path between two nodes + 1.*

Proof: We have proved in theorem 5.3 that a lookup with R_{mode} would complete in $O(log\,N)$ time. As the logical operation of sending a message from a node α_i to another node say α_k is separated in the EE_{mode} from its actual routing, we need to find the upper bound on this overhead of routing the message along the energy-efficient path. It is easy to see that for every such logical message this bound comes to be M_k, where M_k is the path length of the energy efficient path. The number of such $M_k = log\,N$ and let $M_m =$max$(\forall M_k)$. Therefore,

$$Overhead\ of\ EE_{mode} = O(M_m \times log\,N)$$
$$Total\ Time\ of\ EE_{mode} = O(M_m \times log\,N + log\,N) = O((M_m + 1) \times log\,N)$$

Hence, total time taken by $EE_{mode} = O(M \times log\,N)$ where $M = M_m + 1$.

5.9. Node Failures and Node Joins

CSN handles individual node failures by data replication on backup nodes α_b. Such α_b can be deployed along with other nodes but they are not included in $\pi_0 = \{$all nodes in the network$\}$. Hence, they are not part of any cluster after the hierarchical cluster setup. Each α_i would locate available α_b geographically close to it and use them for replicating data. Alternatively, data replicas could be made on other member nodes of the cluster as well. Similar to Chord [2], each sensor node α_i maintains a successor list of size r and if $\alpha_i.successor$ is not responding α_i can try to contact the next successor on the list r and so on. The probability of all sensors on the successor list R of α_i failing simultaneously can be made arbitrarily small with modest values of R [2]. The following theorems are proved in the Chord paper [2]:

Theorem 5.5 *If we use a successor list of length $r = O(log\,N)$ in a network that is initially stable, and then ev-*

ery node fails with probability $\frac{1}{2}$, then with high probability find_successor returns the closest living successor to the query key.

Theorem 5.6 *In a network that is initially stable, if every node then fails with probability $\frac{1}{2}$, then the expected time to execute find_successor is $O(\log N)$.*

In theorem 5.5 and 5.6 the function *find_successor* refers to the Chord [2] procedure that returns the immediate successor of a node say α_i. A node voluntarily leaving the node can be treated as a node failure with the added constraint that the node α_i that is about to leave, possibly because of low energy, must transfer its keys to $\alpha_i.successor$ before it leaves the ring formation cluster. In CSN node joins do not occur independently. A node join can only occur if there is a backup node α_b available to join the ring formation cluster and the node α_i fails voluntarily. Node join is different from the case of α_i voluntarily leaving the cluster as now α_i would pass all its information along with its successor pointers to the α_b. Such a node join is much simpler than the node join of Chord [2].

6. Simulation and Experimental Results

The simulation of CSN was done by developing a novel simulator for sensor networks in C language. The simulator uses a simple first order radio model [19] for wireless communications. Let $E_{electric}$ be the energy dissipated by the transmitter-receiver and $E_{amplifier}$ be the energy dissipated by the transmit amplifier. Then,

$$E_{Transmit}(k,d) = E_{electric} \times k + E_{amplifier} \times k \times d^2 \quad (8)$$

$$E_{Receive}(k) = E_{electric} \times k \quad (9)$$

Where $E_{electric}$ and $E_{amplifier}$ have values $50nJ/bit$ and $100pJ/bit/m^2$ respectively, k is the data rate in bits per packet and d is the distance. The base station is located far away from the test bed and thus communicating with the base station is a high energy operation.

The sensors are randomly deployed on a test bed of 200x200 meters. One such randomly deployed 128-node network is shown in figure 6. The small dots are normal sensors and the circles are cluster heads at the lowest level of hierarchical clustering. Figure 6 illustrates the distribution of cluster heads according to the iterative setup algorithm (ISA). The number of cluster heads are set to be 6% of the total sensors participating in the layer. The cluster head distribution of ISA was seen to be better than that of the parallel setup algorithm (PSA), where the later had a random distribution of cluster heads with many heads concentrating in small areas.

In the simulation, two network topologies were simultaneously created from the set of deployed sensors. One

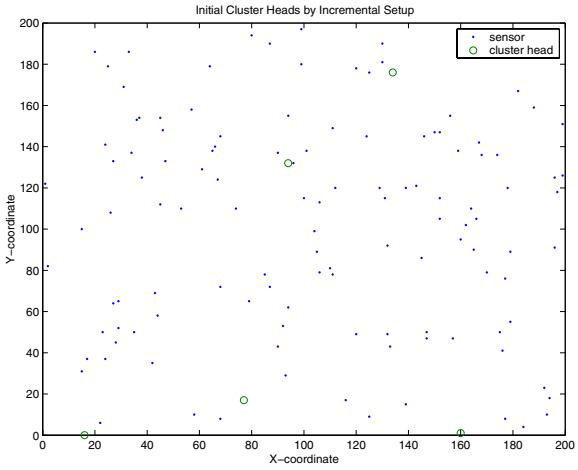

Figure 6. A random deployment of a 128 node network using Iterative Setup Method

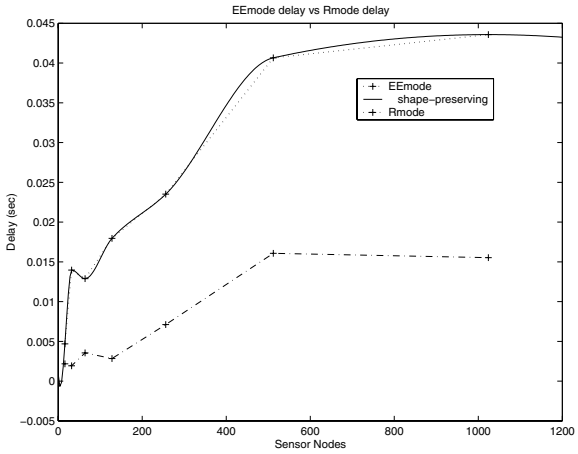

Figure 7. EE_{mode} and R_{mode} delay time

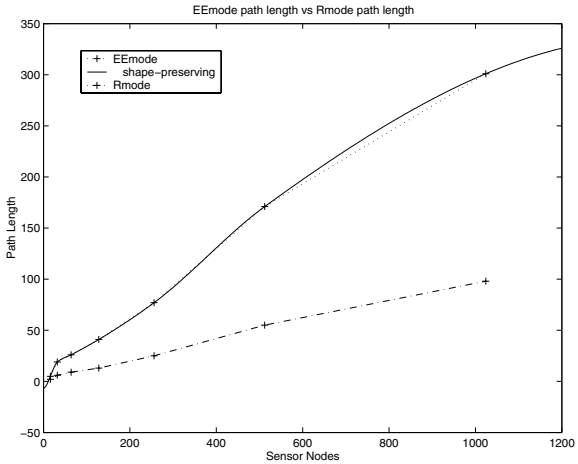

Figure 8. EE_{mode} and R_{mode} path length

topology was formed according to the ISA and the other according to the PSA respectively. On both the topologies the initial network density was 16 nodes. The network density was gradually increased by a factor of 2 during the test runs up to the maximum of 1024 nodes in the 200x200 meters test bed network space. From 300 to 500 test runs of EE_{mode} and R_{mode} each were done before increasing the network density. On every test run, the sensors were again randomly deployed and random data queries were generated to obtain mean values of the selected performance metrics. In order to incorporate the harsh environment conditions typical to actual deployed SNs 3% of random nodes fail after every 20 seconds in the simulation. Further, the cluster hierarchy was set to two-levels.

Figure 7 presents the mean values of delay observed at different network densities. R_{mode} outperforms EE_{mode}, as expected, in the average delay time i.e. the time between the query is sent and the response is generated by the network. The delay time for R_{mode} becomes stable at approximately 0.01 seconds mark where as for EE_{mode} this value is approximately 0.04 seconds.

Also related to delay is path length, figure 8, which we defined as the number of sensor nodes on the single way path from the query to the actual data. Path length shows how many redundant nodes were contacted before the query was successful. The average path length of R_{mode} is much smaller than that of EE_{mode}, as expected. The average path length of R_{mode} for a network of 1000 nodes is 98, which depicts the good performance of CSN; specially in R_{mode}.

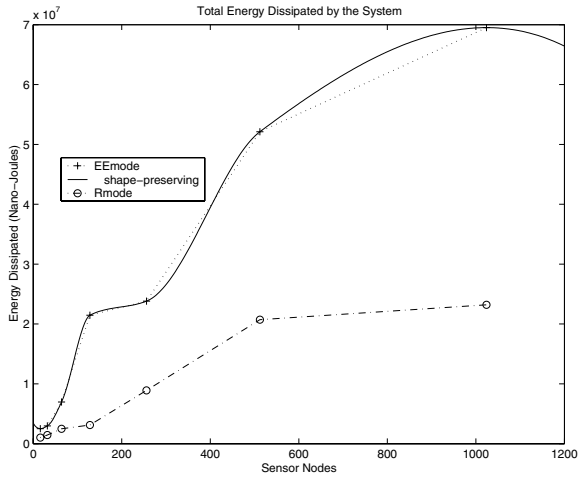

Figure 9. Total energy dissipated by the network

EE_{mode} performed worse than R_{mode} according to the total energy dissipated by the system; figure 9. The actual difference between the EE_{mode} determined best neighbor node and R_{mode} $O(log\,N)$ style jump node was not very significant. So, the fact that the path length of R_{mode} is very small is more advantageous than the slightly more distance of the next node is harmful, when compared to EE_{mode}.

Total energy dissipation results are slightly misleading in the sense that they give the image that EE_{mode} nodes would die quicker as the system is spending more energy than R_{mode}. However, from figure 10 we see that the individual energy dissipation of EE_{mode} nodes is less than that of R_{mode} nodes. We define the individual energy dissipation to be the average energy dissipated by each node participating in the lookup operation. As EE_{mode} nodes consume less energy on average, the network lifetime with EE_{mode} is greater than that of R_{mode}.

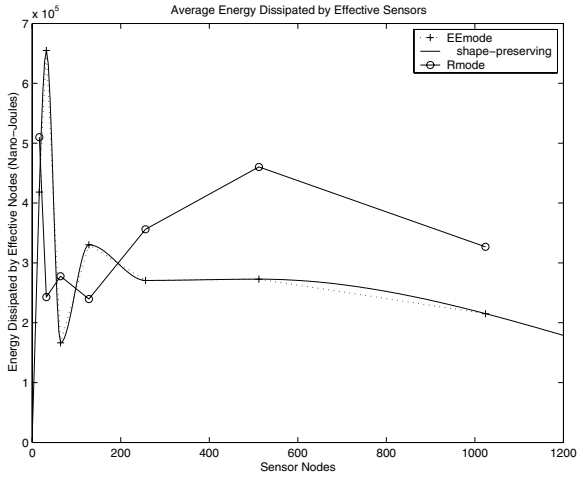

Figure 10. Average energy dissipated by participating nodes

7. Conclusion & Future Work

These days we are carrying out experiments in more realistic environments. Mobility poses unique challenges to CSN because the protocol requires strict arrangement of nodes and communication paths. An example of a mobile SN environment could be sensors deployed in taxicabs in order to monitor traffic. One possible way to handle mobility could be to incorporate a fast update method in CSN that would keep the finger tables up-to-date.

In CSN each node at level $i \neq 0$ needs to store the finger table of not only the $O(log\,N)$ member nodes of the cluster C at level i but also of all member nodes at all lower levels . This increases the storage requirements of the sensor nodes and the storage limitations of these nodes can potentially become a bottleneck for scalability. We are exploring efficient ways to increase the storage area of physical sensors. Another work currently in progress is the comparison study of CSN with LEACH [19] and PEGASIS [20].

CSN provides means for efficiently locating data in wireless sensor networks. The sensors in CSN operate in low-power mode unless a query from the application is made. This greatly increases the lifetime of SN, especially when actual queries from the application are less frequent. The performance of R_{mode}, of CSN , is impressive both when compared to the EE_{mode} and in general. CSN in R_{mode} performs lookup of data with a bound of $O(log\,N)$; which provides strong guarantees to the applications running atop. Finally, CSN scales well to large-scale SNs.

References

[1] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, Data-Centric Storage in Sensornets with GHT, A Geographic Hash Table, *Mobile Networks and Applications* (MONET), Special Issue on Wireless Sensor Networks, Kluwer, mid-2003.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17-32, February 2003.

[3] K. Kalpakis, K. Dasgupta, and P. Namjoshi, Efficient Algorithms for Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 42 , iss. 6, pp. 697-716, August 2003.

[4] M. Bhardwaj, A. Chandrakasan, T. Garnett, Upper Bounds on the Lifetime of Sensor Networks. *IEEE International Conference on Communications*, pp. 785-790, 2001.

[5] S. Lindsey, C. Raghavendra, and K. Sivalingam, Data Gathering in Sensor Networks using the Energy*Delay Metric. *IEEE Transactions on Parallel and Distributive Systems*, special issue on Mobile Computing, pp. 924-935, April 2002.

[6] M. Handy, M. Haase, and D. Timmermann, Low Energy Adaptive Clustering Hierarchy with Deterministic Cluster-Head Selection, *IEEE MWCN*, Stockholm, 2002.

[7] I. Clarke, A distributed decentralised information storage and retrieval system. *Master's thesis*, University of Edinburgh, 1999.

[8] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, Freenet: A distributed anonymous information storage and retrieval system. Proc. *ICSI Workshop on Design Issues in Anonymity and Unobservability*, June 2000.

[9] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, Energy aware wireless microsensor networks, *IEEE Signal Processing Magazine*, vol. 19, iss. 2, pp. 40-50, March 2002.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, System architecture directions for networked sensors, *9th ASPLOS*, pp. 93-104, 2000

[11] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. Proc. *ICASSP*, Salt Lake City, Utah, May 2001.

[12] A. Schmidt, and K. V. Laerhoven, How to build smart appliances? *IEEE Personal Communications*, pp. 66-71, August 2001.

[13] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, Infrastructure Tradeoffs for Sensor Networks, *WSNA02*, Atlanta, Georgia, September 2002.

[14] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, A Taxonomy of Wireless Micro-Sensor Network Models, *MC2R*, vol. 6, no. 2, April 2002.

[15] Y. Yao, and J. Gehrke, The Cougar Approach to In-Network Query Processing in Sensor Networks, *SIGMOD Record* 31(3): 9-18 (2002).

[16] S. R. Madden, R. Szewczyk, M. J. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc sensor networks. Proc. *WMCSA*, 2002.

[17] W.R. Heinzelman, J. Kulik and H. Balakrishnan, Adaptive protocols for information dissemination in wireless sensor networks, Proc. *MOBICOM*, pp. 174-185, Seattle, 1999.

[18] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, Next century challenges: Scalable coordination in sensor networks, Proc. *MOBICOM*, pp. 263-270, Seattle, 1999.

[19] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, An Application-Specific Protocol Architecture for Wireless Microsensor Networks, *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660-670, October 2002.

[20] S. Lindsey, and C. S. Raghavendra, PEGASIS: Power-Efficient Gathering in Sensor Information Systems, *International Conference on Communications*, 2001.

[21] A. Manjeshwar, and D.P Agrawal, TEEN: a routing protocol for enhanced efficiency in wireless sensor networks. Proc. *15th Parallel and Distributed Processing Symposium*, pp. 2009-2015, 2001.

[22] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. Proc. *MOBICOM*, pp. 56-67, Boston, MA, August 2000. ACM Press.

[23] A. Rowstron, and P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-topeer systems, *Lecture Notes in Computer Science*, vol. 2218, pp. 329-350, 2001.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. Proc. *ACM SIGCOMM*, August 2001.

[25] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.

[26] FIPS 180-1. Secure Hash Standard. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.

[27] Gnutella. http://gnutella.wego.com/.

[28] J. Kubaiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, OceanStore: An architecture for global-scale persistent storage. Proc. *9th ASPLOS*, pp. 190-201, November 2000.

[29] http://www.ohaha.com/design.html.