KDDCS: A Load-Balanced In-Network Data-Centric Storage Scheme for Sensor Networks[†]

Mohamed Aly, Kirk Pruhs, and Panos K. Chrysanthis
Dept. of Computer Science, University of Pittsburgh
Pittsburgh, PA, USA
maly@cs.pitt.edu, kirk@cs.pitt.edu, panos@cs.pitt.edu

ABSTRACT

We propose an *In-Network Data-Centric Storage* (INDCS) scheme for answering ad-hoc queries in sensor networks. Previously proposed In-Network Storage (INS) schemes suffered from *Storage Hot-Spots* that are formed if either the sensors' locations are not uniformly distributed over the coverage area, or the distribution of sensor readings is not uniform over the range of possible reading values. Our K-D tree based Data-Centric Storage (KDDCS) scheme maintains the invariant that the storage of events is distributed reasonably uniformly among the sensors. KDDCS is composed of a set of *distributed* algorithms whose running time is within a poly-log factor of the diameter of the network. The number of messages any sensor has to send, as well as the bits in those messages, is poly-logarithmic in the number of sensors. Load balancing in KDDCS is based on defining and distributively solving a theoretical problem that we call the *Weighted Split Median* problem. In addition to analytical bounds on KDDCS individual algorithms, we provide experimental evidence of our scheme's general efficiency, as well as its ability to *avoid* the formation of storage hot-spots of various sizes, unlike all previous INDCS schemes.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Distributed databases, Query processing*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Sensor Network, Power-Aware, Distributed Algorithms

1. INTRODUCTION

Sensor networks provide us with the means of effectively monitoring and interacting with the physical world. As an illustrative example of the type of sensor network application that concerns us here, consider an emergency/disaster scenario where sensors are deployed in the area of the disaster [17]. It is the responsibility of the sensor network to sense and store events of potential interest. An *event* is composed of one or more attributes (e.g. temperature, carbon monoxide level, etc.), the identity of the sensor that sensed the event, and the time when the event was sensed. As first responders move through the disaster area with hand-held devices, they issue queries about recent events in the network. For example, the first responder might ask for the location of all sensor nodes that recorded high carbon monoxide levels in the last 15 minutes, or he might ask whether any sensor node detected movement in the last minute. Queries are picked up by sensors in the region of the first responder. The sensor network is then responsible for answering these queries. The first responders use these query answers to make decisions on how to best manage the emergency.

The *ad-hoc queries* of the first responders will generally be *multi-dimensional range queries* [9], that is, the queries concern sensor readings that were sensed over a small time window in the near past and that fall in a given range of the attribute values. In-Network Storage (INS) is a storage technique that has been specifically presented to efficiently process this type of queries. INS involves storing events locally in the sensor nodes. Storage may be in-network because it is more efficient than shipping all the data (i.e., raw sensor readings) out of the network (for example to base stations), or simply because no out-of-network storage is available. All INS schemes already presented in literature were *Data-Centric Storage (DCS)* schemes [15]. In any In-Network Data-Centric Storage (INDCS) scheme, there exists a function from events to sensors that maps each event to an owner sensor based on the value of the attributes of that event. The owner sensor will be responsible for storing this event. The owner may be different than the sensor that originally generated the event. To date, the Distributed Index for Multi-dimensional data (DIM) scheme [9] has been shown to exhibit the best performance among all proposed INDCS schemes in dealing with sensor networks whose query loads are basically composed of ad-hoc queries .

In DIM [9], the events-to-sensors mapping is based on a K-D tree [3], where the leaves $\mathcal{R}$ form a partition of the coverage area, and each element of $\mathcal{R}$ contains either zero or one sensor. The formation of the K-D tree consists of rounds. Initially, $\mathcal{R}$ is a one element set containing the whole coverage area. In each odd/even round r, each region $R \in \mathcal{R}$ that contains more than one sensor is bisected horizontally/vertically. Each time that a region is split, each sensor in that region has a bit appended to its address specifying which side of the split the sensor was on. Thus, the length of a sensor's address (bit-code) is its depth in the underlying K-D tree. When a sensor generates an event, it maps such event to a binary code based on a repetitive fixed uniform splitting of the attributes' ranges in a round robin fashion. For our purposes, it is sufficient for now to

[†]This work has been funded in part by NSF grants ANI-0325353, CCF-0448196, CCF-0514058, and IIS-0534531.

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.
Copyright 2006 ACM 1-59593-433-2/06/0011 ...$5.00.

consider the cases that the event consists of only one attribute, say temperature. Then, the high order bits of the temperature are used to determine a root-to-leaf path in the K-D tree, and if there is a sensor in the region of the leaf, then this sensor is the owner of this event. Due to the regularity of regions in this K-D tree, the routing of an event from the generating sensor to the owner sensor is particularly easy using Greedy Perimeter Stateless Routing (GPSR) [6]. Full description of DIM is presented in Section 2.

Though it is the best DCS scheme so far, DIM suffers from several problems. One problem is that events may well be mapped to orphan regions that contain no sensors. Thus, DIM requires some kludge to assign orphan regions to neighboring sensors.

Another major problem in DIM is that of *storage hot-spots*. Storage hot-spots may occur if the sensors are not uniformly distributed. A *storage hot-spot* occurs when relatively many events are assigned to a relatively small number of the sensors. For example, if there was only one sensor on one side of the first bisection, then half of the events would be mapped to this sensor if the events were uniformly distributed over the range of possible events. Due to their storage constraints, the presence of a storage hot-spot leads to increasing the dropping rate of events by overloaded sensors. Clearly, this has a significant impact on the quality of data (QoD) generated by the sensor network. Queries for events in a storage hot-spot may be delayed due to contention at the storage sensors and the surrounding sensors. More critically, the sensors in and near the hot-spot may quickly run out of energy, due to the high insertion/query load imposed to them. This results in a loss of the events generated at these sensors, the events stored at these sensors, and possibly a decrease in network connectivity. Increased death of sensors results in decreasing the coverage area and causes the formation of coverage gaps within such area. Both of which consequently decrease QoD. Certainly, it is not desirable to have a storage scheme whose performance and QoD guarantees rest on the assumption that the sensors are uniformly distributed geographically.

Storage hot-spots may also occur in DIM if the distribution of events is not uniform over the range of possible events. It is difficult to imagine any reasonable scenario where the events are uniformly distributed over the range of all possible events. Consider the situation where the only attribute sensed is temperature. One would expect that most temperature readings would be clustered within a relatively small range rather than uniform over all possible temperatures. Without any load balancing, those sensors responsible for temperatures outside this range would store no events.

In this paper, we provide a load-balanced INDCS scheme based on K-D trees, that we, not surprisingly, call *K-D tree based DCS (KDDCS)*. In our KDDCS scheme, the refinement of regions in the formation of the K-D tree has the property that the numbers of sensors on both sides of the partition are approximately equal. As a result of this, our K-D tree will be balanced, there will be no orphan regions, and, regardless of the geographic distribution of the sensors, the ownership of events will uniformly distributed over the sensors if the events are uniformly distributed over the range of possible events. We present a modification of GPSR routing, namely *Logical Stateless Routing (LSR)*, for the routing of events from their generating sensors to their owner sensors, that is competitive with the GPSR routing used in DIM. In order to maintain load balance in the likely situation that the events are not uniformly distributed, we present a re-balancing algorithm that we call *K-D Tree Re-balancing (KDTR)*. Our re-balancing algorithm guarantees load balance even if the event distribution is not uniform. KDTR has essentially minimal overhead. We identify a problem, that we call the *weighted split median* problem, that is at the heart of both the construction of the initial K-D tree, and the re-balancing of the K-D tree. In the weighted split median problem, each sensor has an associated weight/multiplicity, and the sensors' goal is to distributively determine a vertical line with the property that the aggregate weight on each side of the line is approximately equal. We give a distributed algorithm for the weighted split median problem, and show how to use this algorithm to construct our initial K-D tree, and to re-balance the tree throughout the network lifetime.

We are mindful of the time, message complexity, and node storage requirements, in the design and implementation of all of our algorithms. The time for all of our algorithms is within a poly-log factor of the diameter of the network. Obviously, no algorithm can have time complexity less than the diameter of the network. The number of messages, and number of bits in those messages, that any particular node is required to send by our algorithms is poly-logarithmic in number of sensors. The amount of information that each node must store to implement one of our algorithms is logarithmic in the number of sensors.

Experimental evaluation shows that the main advantages of KDDCS, when compared to the pure DIM, are:

- Achieving a better *data persistence* by balancing the storage responsibility among sensor nodes.
- Increasing the *QoD* by distributing the storage hot-spot events among a larger number of sensors.
- Increasing the *energy savings* by achieving a well balanced energy consumption overhead among sensor nodes.

The rest of the paper is organized as follows. Section 2 presents an overview of the differences between DIM and KDDCS. Section 3 describes the weighted split median problem, and our distributed solution. Section 4 describes the components of KDDCS. Section 5 presents our K-D tree re-balancing algorithm. Experimental results are discussed in Section 6. Section 7 presents the related work.

2. OVERVIEW OF DIM VS. KDDCS

In this section, we will briefly describe the components of both schemes, DIM and KDDCS, and highlight the differences between the two schemes using a simple example.

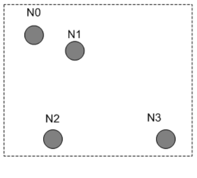
We assume that the sensors are arbitrarily deployed in the convex bounded region R. We assume also that each sensor is able to determine its geographic location (i.e., its x and y coordinates), as well as, the boundaries of the service area R. Each node is assumed to have a unique *NodeID*, like a MAC address. Sensor nodes are assumed to have the capacity for wireless communication, basic processing and storage, and they are associated with the standard energy limitations.

The main components of any DCS scheme are: the *sensor to address* mapping that gives a logical address to each sensor, and the *event to owner-sensor* mapping that determines which sensor will store the event. The components of DIM and KDDCS are:

- Repetitive splitting of the geographic region to form the underlying K-D tree, and the logical sensor addresses.
- Repetitive splitting of the attribute ranges to form the bit-code for an event.
- The routing scheme to route an event from the generating sensor to the owner sensor.

We now explain how DIM implements these components.

Let us start with the formation of the K-D tree in DIM. DIM starts the network operation with a *static* node to bit-code mapping phase. In such phase, each sensor locally determines its binary address by uniformly splitting the overall service area in a round



Sensor nodes' geographical locations

Figure 1: Initial network configuration

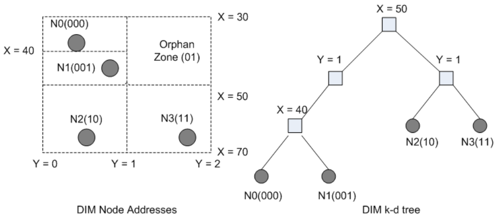


Figure 2: DIM K-D tree

robin fashion, horizontally then vertically, and left shifting its bit-code with every split by 0 (or 1) bit when falling above (or below) the horizontal split line (similarly, by a 0 bit if falling on the left of the vertical split line, or a 1 bit otherwise). Considering the region as partitioned into zones, the process ends when every sensor lies by itself in a *zone*, such that the sensor address is the zone bit code. Thus, the length of the binary address of each sensor (in bits) represents its depth in the underlying K-D tree. Note that from a sensor address, one can determine the physical location of the sensor. In case any orphan zones exist (zones physically containing no sensors in their geographic area), the ownership of each of these zones is delegated to one of its neighbor sensors. As an example, consider the simple input shown in Figure 1. The K-D tree formed by DIM is shown in Figure 2. In this figure, the orphan zone (01) is assumed to be delegated to node 001, which is the least loaded among its neighbors.

We now turn to the construction of an event bit-code in DIM. The generation of the event bit-code proceeds in rounds. As we proceed, there is a range R_j associated with each attribute j of the event. Initially, the range R_j is the full range of possible values for attribute j. We now describe how a round $i \geq 0$ works. Round i, determines the $(i{+}1)^{th}$ high order bit in the code. Round i depends on attribute $j = i \bmod k$ of the event, where k is the number of attributes in the event. Assume the current value of R_j is $[a, c]$, and let $b = (a + c)/2$ be the midpoint of the range R_j. If the value of attribute j is in the lower half of the range R_j, that is in $[a, b]$, then the i^{th} bit is 0, and R_j is set to be the lower half of R_j. If the value of attribute j is in the upper half of the range R_j, that is in $[b, c]$, then the i^{th} bit is 1, and R_j is set to be the upper half of R_j.

To show the events to bit-code mapping in DIM, consider that the events in our example (shown in Figure 2) are composed of two attributes, temperature and pressure, with ranges $(30, 70)$ and $(0, 2)$, respectively. Let an event with values $(55, 0.6)$ be generated by Node N3(11). The 4 high-order bits of the bit-code for this event are 1001. This is because temperature is in the top half of the range $[30, 70]$, pressure is in the bottom half of the range $[0, 2]$, then temperature is in the bottom half of the range $[50, 70]$, and pressure is in the top half of the range $[0, 1]$. Thus, the event should be routed toward the geometric location specified by code 1001.

In DIM, an event is routed using Greedy Perimeter Stateless Routing (GPSR) [6] to the geographic zone with an address matching the high order bits of the event bit-code. In our example, the sensor 10 will store this event since this is the sensor that matches the high order bits of the bit-code 1001. If there is no sensor in this region, then, the event is stored in a neighboring region.

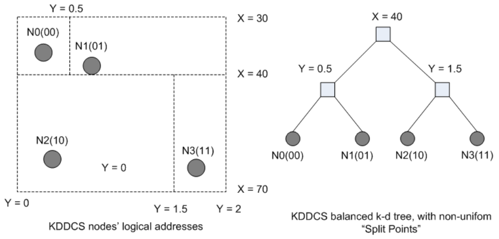


Figure 3: KDDCS K-D tree

We now highlight the differences between our proposed KDDCS scheme, and DIM. The *first* difference is how the splitting is accomplished during the formation of the K-D tree. In KDDCS, the split line is chosen so that there are equal numbers of sensors on each side of the split line. Recall that, in DIM, the split line was the geometric bisector of the region. Thus, in KDDCS, the address of a sensor is a logical address and does not directly specify the location of the sensor. Also, note that the K-D tree in KDDCS will be balanced, while this will not be the case in DIM if the sensors are not uniformly distributed. This difference is illustrated by the K-D tree formed by KDDCS shown in Figure 3 for the same simple input shown in Figure 1. The *second* difference is that in determining the owner sensor for an event, the range split point b need not be the midpoint of the range R_j. The value of b is selected to balance the number of events in the ranges $[a, b]$ and $[b, c]$. Thus, in KDDCS, the storage of events will be roughly uniform over the sensors. The *third* difference is that, since addresses are not geographic, KDDCS needs a routing scheme that is more sophisticated than GPSR.

3. THE WEIGHTED SPLIT MEDIAN PROBLEM

Before presenting our KDDCS scheme, we first define the *weighted split median* problem in the context of sensor networks and present an efficient distributed algorithm to solve the problem. Each sensor s_i initially knows w_i associated values $v_1, \ldots v_{w_i}$. Let $W = \sum_{i=1}^{n} w_i$ be the number of values. The goal for the sensors is to come to agreement on a split value V with the property that approximately half of the values are larger than V and half of the values are smaller than V.

We present a distributed algorithm to solve this problem. The time complexity of our algorithm is $O(\log n)$ times the diameter of the communication network in general, and $O(1)$ times the diameter if n is known a priori within a constant factor. Each node is required to send only $O(\log n)$ sensor ID's. The top level steps of this algorithm are:

1. Elect a leader sensor s_ℓ, and form a breadth first search (BFS) tree T of the communication network that is rooted at s_ℓ.
2. The number of sensors n, and the aggregate number of values W is reported to s_ℓ.
3. The leader s_ℓ collects a logarithmically-sized uniform random sample L of the values. The expected number of times that a value from sensor s_i is included in this sample is $\Theta\left(\frac{w_i \log n}{W}\right)$.
4. The value of V is then the median of the reported values in L, which s_ℓ reports to all of the sensors.

We need to explain how these steps are accomplished, and why the algorithm is correct.

We start with the first step. We assume that each sensor has a lower bound k on the number of sensors in R. If a sensor has no idea of the number of other sensors, it may take $k = 2$.

Then, each sensor decides independently, with probability $\Theta\left(\frac{\ln k}{k}\right)$, to become a candidate for the leader. Each candidate sensor s_c initiates the construction of a BFS tree of the communication graph rooted at s_c by sending a message Construct(s_c) to its neighbors. Assume a sensor s_i gets a message Construct(s_c) from sensor s_j. If this is the first Construct(s_c) message that it has received, and s_c's ID is larger than the ID of any previous candidates in prior Construct messages, then:

- s_i makes s_j its tentative parent in the BFS tree T, and
- forwards the Construct(s_c) message to its neighbors.

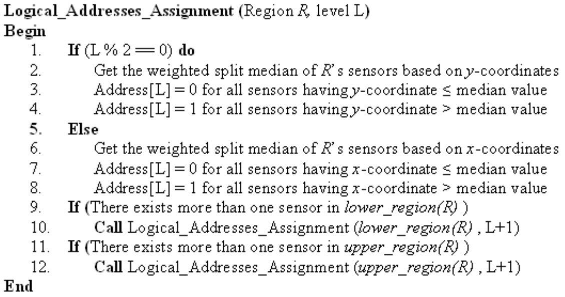
If the number of candidates was positive, then, after time proportional to the diameter of the communication network, there will be a BFS tree T rooted at the candidate with the largest ID. Each sensor may estimate an upper bound for the diameter of the communication graph to be the diameter of R divided by the broadcast radius of a sensor. After this time, the sensors know that they have reached an agreement on T, or that there were no candidates. If there were no candidates, each sensor can double its estimate of k, and repeat this process. After $O(\log n)$ rounds, it will be the case that $k = \Theta(n)$. Once $k = \Theta(n)$, then, with high probability (that is, with probability $1 - \frac{1}{\text{poly}(n)}$), the number of candidates is $\Theta(\log n)$. Thus, the expected time complexity to accomplish the first step is $O(n \log n)$. Assuming that each ID has $O(\log n)$ bits, the expected number of bits that each sensors has to send is $O(\log^2 n)$ since there are are likely only $O(\log n)$ candidates on the first and only round in which there is a candidate. A $\log n$ factor can be removed if each sensor initially knows an estimate of n that is accurate to within a multiplicative constant factor.

The rest of the steps will be accomplished by waves of root-to-leaves and leaves-to-root messages in T. The second step is easily accomplished by a leave-to-root wave of messages reporting on the number of sensors and number of values in each subtree. Let T_i be the subtree of T rooted at sensor s_i, and W_i the aggregate number of values in T_i. The value W_i that s_i reports to its parents is w_i plus the aggregate values reported to s_i by its children in T. The sensor count that s_i reports to its parents is one plus the sensor counts reported to s_i by its children in T.

The third step is also accomplished by a root-to-leaves wave and then a leaves-to-root wave of messages. Assume a sensor s_i wants to generate a uniform random sample of L_i of the values stored in the sensors in T_i. The value of L_ℓ for the leader is $\Theta(\log n)$. Let $s_{i_1}, \ldots, s_{i_d}$ be the children of s_i in T. Node s_i generates the results to L_i Bernoulli trials, where each trial has $d + 1$ outcomes corresponding to s_i and its d children. The probability that the outcome of a trial is s_i is $\frac{w_i}{W_i}$, and the probability that the outcome is the child s_{i_j} is $\frac{w_{i_j}}{W_i}$. Then, s_i informs each child s_{i_j} how often it was selected, which becomes the value of $L_{i_j} \cdot s_i$, then waits until it receives samples back from all of its children. s_i then unions these samples, plus a sample of values of the desired size from itself, and then passes that sample back to its parent. Thus, each sensor has to send $O(\log n)$ ID's.

The leader s_ℓ then sets V to be the median of the values of the sample L, then, in a root-to-leaves message wave, informs the other sensors of the value of V.

We now argue that, with high probability, the computed median of the values is close to the true median. Consider a value $\hat{V}$ such that only a fraction $\alpha < \frac{1}{2}$ of the values are less than $\hat{V}$. One can think of each sampled value as being a Bernoulli trial with outcomes *less* and *more* depending on whether the sampled value is less than $\hat{V}$. The number of *less* outcomes is binomially distributed with mean αL. In order for the computed median to be less than $\hat{V}$, one needs the number of *less* outcomes to be at least $L/2$, or equivalently $(\frac{1}{2} - \alpha)L$ more than the mean αL. But the probability that a binomially distributed variable exceeds its mean μ by a factor of $1 + \delta$ is at most $e^{\frac{-\delta^2 \mu}{3}}$. Thus, by picking the multiplicative constant in the sample size to be sufficiently large (as a function of α), one can guarantee that, with high probability, the number of values less than the computed median V cannot be much more than $L/2$. A similar argument shows that the number more than the computed median V can not be much more than $L/2$.



```
Logical_Addresses_Assignment (Region R, level L)
Begin
    If (L % 2 == 0) do
        Get the weighted split median of R's sensors based on y-coordinates
        Address[L] = 0 for all sensors having y-coordinate ≤ median value
        Address[L] = 1 for all sensors having y-coordinate > median value
    Else
        Get the weighted split median of R's sensors based on x-coordinates
        Address[L] = 0 for all sensors having x-coordinate ≤ median value
        Address[L] = 1 for all sensors having x-coordinate > median value
    If (There exists more than one sensor in lower_region(R) )
        Call Logical_Addresses_Assignment (lower_region(R) , L+1)
    If (There exists more than one sensor in upper_region(R) )
        Call Logical_Addresses_Assignment (upper_region(R) , L+1)
End
```

Figure 4: Logical address assignment algorithm

If the leader finds that n is small in step 2, it may simply ask all sensors to report on their identities and locations, and then compute V directly.

Now that we solved the weighted split median problem, we present the components of the KDDCS scheme in the next section.

4. KDDCS

We now present our KDDCS scheme in details. We explain how the initial K-D tree is constructed, how events are mapped to sensors, and how events are routed to their owner sensors.

4.1 Distributed Logical Address Assignment Algorithm

The main idea of the algorithm is that the split lines used to construct the K-D tree are selected so that each of the two resulting regions contain an equal number of sensors. The split line can be determined using our weighted split median algorithm with each sensor having unit weight, and the value for each sensor is either its x coordinate or its y coordinate. The recursive steps of the algorithm are shown in Figure 4. We now describe in some greater detail how a recursive step works.

The algorithm starts by partitioning the complete region R horizontally. Thus, the distributed weighted split median algorithm (presented in section 3) is applied for R using the y-coordinates of the sensors to be sent to the BFS root. Upon determining weighted split median of R, sensors having lower y-coordinate than the median value (we refer to these sensors as those falling in the lower region of R) assign their logical address to 0. On the other hand, those sensor falling on the upper region of R assign themselves a 1 logical address. At the end of the first recursive step, the terrain can be looked at as split into two equally logically loaded partitions (in terms of the number of sensors falling in each partition).

At the next step, the weighted split median algorithm is applied locally in each of the sub-regions (lower/upper), while using the sensors' x-coordinates, thus, partitioning the sub-regions vertically rather than horizontally. Similarly, sensors' logical addresses are updated by left-shifting them with a 0 bit for those sensors falling in the lower regions (in other words, sensor nodes falling on the left of the weighted median line), or with a 1 bit for sensor nodes falling in the upper regions (i.e., sensor nodes falling on the right of the weighted median line).

The algorithm continues to be applied distributively by the different subtrees until each sensor obtains a unique logical address, using x and y coordinates of sensors, in a round robin fashion, as the criterion of the split. The algorithm is applied in parallel on the different subtrees whose root nodes fall at the same tree level. At the i^{th} recursive step, the algorithm is applied at all intermediate nodes falling at level $i-1$ of the tree. Based on the definition of the weighted split median problem, the algorithm results in forming a balanced binary tree, such that sensors represent leaf nodes of this tree (intermediate nodes of the tree are logical nodes, not physical sensors). The algorithm terminates in $\log n$ recursive steps. At the end of the algorithm, the size of the logical address given to each sensor will be $\log n$ bits.

Recall that the time complexity of our weight split median algorithm is $O(d \log n)$, where d is the diameter of the region. Thus, as the depth of our K-D tree is $O(\log n)$, we get that the time complexity for building the tree is $O(d \log^2 n)$. If the sensors are uniformly distributed, then, as the construction algorithm recurses, the diameters of the regions will be geometrically decreasing. Thus, in the case of uniformly distributed sensors, one would expect the tree construction to take time $O(d \log n)$. As our weighted split median algorithm requires each sensor to send $O(\log n)$ ID's, and our K-D tree has depth $O(\log n)$, we can conclude that during the construction of our K-D tree, the number of ID's sent by any node is $O(\log^2 n)$.

4.2 Event to Bit-code Mapping

In this section, we explain how the event to bit-code mapping function is determined. Recall that the main idea is to set the split points of the ranges so that the storage of events is roughly uniform among sensor nodes. To construct this mapping requires a probability distribution on the events. In some situations, this distribution might be known. For example, if the network has been operational for some period of time, a sampling of prior events might be used to estimate a distribution. In cases where it is not known, say when a network is first deployed, we can temporarily assume a uniform distribution.

In both cases, we use the balanced binary tree as the base tree to overlay the attribute-specific K-D tree on (Recall that a K-D tree is formed by k attributes). This is basically done by assigning a range for each of the k attributes to every intermediate node in the tree. Note that the non-leaf nodes in the K-D tree are logical nodes that do not correspond to any particular sensor. One may think of non-leaf nodes as *regions*. Any split point p of a node x of tree level l, where $l\%k = i$, represents a value of attribute i. Such split point partitions the range of attribute i falling under responsibility of node x into two subranges such that the the subrange lower than p is assigned to the left child of x, while the other range is assigned to x's right child. Note that the other $k-1$ ranges of node x, corresponding to the remaining $k-1$ attributes, are simply inherited by both children of x.

Knowing the data distribution, the split points of the tree should be predefined in a way to cope with any expected irregularity in the load distribution among the K-D tree leaf nodes. For example, given an initial temperature range $(30, 70)$ and knowing that 50% of the events will fall in the temperature range $(65, 70)$, the root split point should be defined as 65 (assuming that the temperature is the first attribute in the event). Therefore, based on the selected root split point, the left child subtree of the root will be responsible of storing events falling in the temperature range $(30, 65)$, while the right child subtree will store events falling in the range $(65, 70)$. Figure 3 gives an example of non-uniform initialization of split points.

We finish by describing what information is stored in each sensor node. Each sensor node corresponds to a leaf in the K-D tree. Each sensor knows its logical address in the tree. Further, each leaf in the K-D tree knows all the pertinent information about each of its ancestors in the tree. The pertinent information about each node is:

- The geographic region covered.
- The split line separating its two children.
- The attribute range, and attribute split point, associated with this region.

From this information, each leaf/sensor can determine the range of events that will be stored at this sensor. Note that each sensor only stores $O(\log n)$ information about the K-D tree.

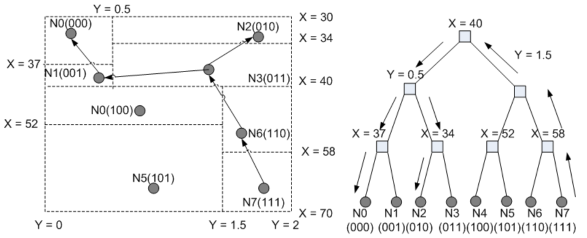
4.3 Incremental Event Hashing and Routing

Strictly speaking, the events-to-sensors mapping in DIM actually produces a geographic location. GPSR routing can then be used to route that event towards that geographic location. If the destination is contained in a leaf region with one sensor, then that sensor stores the event. If the leaf region is an orphan, then one of the sensors in the neighboring regions will store this event.

In our scheme, the events-to-sensors mapping provides a logical address. Essentially, all that the sensor generating the event can determine from this logical address is a general direction of the owner sensor. Thus, our routing protocol, which we call *Logical Stateless Routing (LSR)*, is in some sense less direct.

LSR operates in $O(\log n)$ rounds. We explain how a round works. Assume that a source sensor with a logical address s wants to route an event e to a sensor with logical address t. However, s does not know the identity of the sensor t. Recall that s knows the pertinent information about its ancestors in the K-D tree. In particular, s knows the range split values of its ancestors. Thus, s can compute the least common ancestor (LCA) of s and t in the K-D tree. Assume that the first bit of disagreement between s and t is the ℓ^{th} bit. So, the least common ancestor (LCA) of s and t in the K-D tree has depth ℓ. Let R be the region corresponding to the LCA of s and t, L the split line corresponding to this region, and R_0 and R_1 the two subregions of R formed by L. Without loss of generality, assume that $s \in R_0$ and $t \in R_1$. From its own address, and the address of t, the sensor s can conclude that t is in the region R_1. Recall that s knows the location of the split line L. The sensor s computes a location x in the region R_1. For concreteness here, let us assume that x is some point in R_1 that lies on the line intersecting s and perpendicular to L (Although there might be some advantages to selecting x to be the geometric center of the region R_1). LSR then directs a message toward the location x using GPSR. The message contains an additional field noting that this is a ℓ^{th} round message. The ℓ^{th} round terminates when this message first reaches a sensor s' whose address agrees with the address of t in the first $\ell+1$ bits. The sensor s' will be the first sensor reached in R_1. Round $\ell+1$ then starts with s' being the new source sensor.

We explain how range queries are routed by means of an example. This example also essentially illustrates how events are stored. Figure 5 gives an example of a multi-dimensional range query and shows how to route it to its final destination. In this example, a multi-dimensional range query arises at node $N7(111)$ asking for the number of events falling in the temperature range $(30, 32)$ and pressure range $(0.4, 1)$ that were generated throughout the last 2 minutes. Node $N7$ knows that the range split point for the root



Flow of a query (x Є (30:32), y Є (0.4:1)) between the sensor nodes

Flow of the same query between the k-d tree levels

Figure 5: Example of routing a query on KDDCS

was temperature 40, and thus, this query needs to be routed toward the left subtree of the root, or geometrically toward the top of the region, using GPSR. The first node in this region that this event reaches is say $N3$. Node $N3$ knows that the first relevant split point is pressure = 0.5. Thus, the query is partitioned into two sub-queries, $((30, 32), (0.4, 0.5))$ and $((30, 32), (0.5, 1))$. When processing the first subquery, node $N3$ forwards it to the left using GPSR. $N3$ can then tell that the second query should be routed to the other side of its parent in the K-D tree since the range split for its parent is temperature 34. The logical routing of this query is shown on the right in Figure 5, and a possible physical routing of this query is shown on the left in Figure 5.

As LSR does not initially know the geometric location of the owner sensor, the route to the owner sensor cannot possibly be as direct as it is in DIM. But, we argue that the length of the route in LSR should be at most twice the length of the route in DIM. Assume for the moment that all messages are routed by GPSR along the direct geometric line between the source sensor and the destination location. Let us assume, without loss of generality, that LSR is routing horizontally in the odd rounds. Then, the routes used in the odd rounds do not cross any vertical line more than once. Hence, the sum of the route distances used by LSR in the odd rounds is at most the diameter of the region. Similarly, the sum of the route distances used by LSR in the even rounds is at most the diameter of the region. Thus, the sum of the route distances for LSR, over all rounds, is at most twice the diameter. The geometric distance between the source-destination pair in DIM is obviously at most the diameter. So we can conclude that the length of the route found by LSR is at most twice the length of the route found by DIM, assuming that GPSR is perfect. In fact, the only assumption that we need about GPSR to reach this conclusion is that the length of the path found by GPSR is roughly a constant multiple times the geometric distance between the source and destination. Even this factor of two can probably be improved significantly in expectation if the locations of the sensors are roughly uniform. A simple heuristic would be to make the location of the target x equal to the location of the destination sensor t if the sensors in R_1 where uniformly distributed. The location of x can easily be calculated by the source sensor s given information local to s.

5. KDTR: K-D TREE RE-BALANCING ALGORITHM

Based on the KDDCS components presented so far, KDDCS avoids the formation of storage hot-spots resulting from skewed sensor deployments, and from skewed events distribution if the distribution of events was known a priori. However, storage hot-spots may be formed if the initial presumed events distribution was not correct, or if events distribution evolves over times. We present a K-D tree re-balancing algorithm, KDTR, to re-balance the load.

In the next subsections, we first explain how to determine the roots of the subtrees that will re-balance, and then show how a re-balancing operation on a subtree works. We assume that this re-balancing is performed periodically with a fixed period.

5.1 Selection of Subtrees to be Re-Balanced

The main idea is to find the highest unbalanced node in the K-D tree. A node is unbalanced if the ratio of the number of events in one of the child subtrees over the number of events stored in the other child subtree exceeds some threshold h. This process of identifying nodes to re-balance proceeds in $O(\log n)$ rounds from the leaves to the root of the K-D tree.

We now describe how round $i \geq 1$ works. Intuitively, round i occurs in parallel on all subtrees rooted at nodes of height $i + 1$ in the K-D tree. Let x be a node of height $i + 1$. Let the region associated with x be R, the split line be L, and the two subregions of R be R_0 and R_1. At the start of this round, each sensor in R_0 and R_1 knows the number of stored events C_0 and C_1 in R_0 and R_1, respectively. The count C_0 is then flooded to the sensors in R_1, and the count C_1 is flooded to the sensors in R_0. After this flooding, each sensor in R knows the number of events stored in R, and also knows whether the ratio $\max(\frac{C_0}{C_1}, \frac{C_1}{C_0})$ exceeds h.

The time complexity per round is linear in the diameter of a region considered in that round. Thus, the total time complexity is $O(D \log n)$, where D is the diameter of the network, as there are $O(\log n)$ rounds. The number of messages sent per node i in a round is $O(d_i)$, where d_i is the degree of node i in the communication network. Thus, the total number of messages sent by a node i is $O(d_i \log n)$.

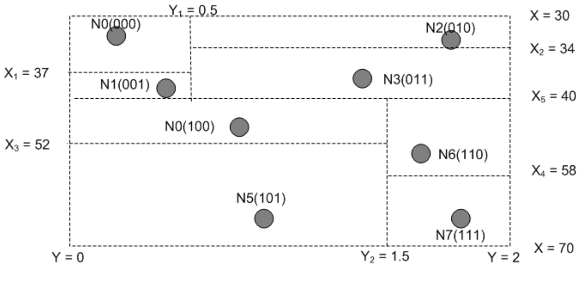
Re-Balancing is then performed in parallel on all unbalanced nodes, that have no unbalanced ancestors. Note that every leaf knows if an ancestor will re-balance, and is so, the identity of the unique ancestor that will balance. All the leaves of a node that will re-balance, will be aware of this at the same time.

5.2 Tree Re-balancing Algorithm

Let x be an internal node to the K-D tree that needs to be re-balanced. Let the region associated with x be R. Let the attribute associated with node x be the j'th attribute. So, we need to find a new attribute split L for the j'th attribute for node x. To accomplish this, we apply the weighted split median procedure, where the weight w_i associated with sensor i is the number of events stored at sensor i, and the values are the j'th attributes of the w_i events stored at that sensor. Thus, the computed attribute split L has the property that, in expectation, half of the events stored in R have their j'th attribute larger than L, and half of the events stored in R have their j'th attribute smaller than L.

Let R_0 and R_1 be the two subregions of R. Eventually, we want to recursively apply this process in parallel to the regions R_0 and R_1. But before recursing, we need to route some events from one of R_0 or R_1 to the other. The direction of the routing depends on whether the attribute split value became larger or smaller. Let us assume, without loss of generality, that events need to be routed from R_0 to R_1. Consider an event e stored at a sensor s in R_0 that needs to be routed to R_1. The sensor s picks a destination logical address t, uniformly at random, from the possible addresses in the region R_1. The event e is then routed to t using the routing scheme described in section 4.3. The final owner for e in R_1 cannot be determined until our process is recursively applied to R_1, but this process cannot be recursively applied until the events that should be stored in R_1 are contained in R_1. The fact the the destination addresses in R_1 were picked uniformly at random ensures load balance.

This process can now be recursively applied to R_0 and R_1.



Initial assignment of "Split Points" based on KDDCS

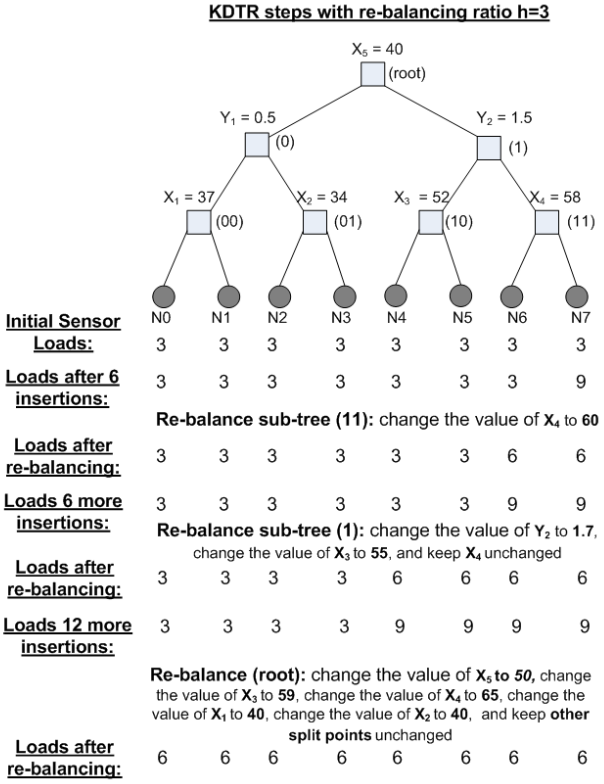
Figure 6: KDDCS original K-D tree

We now discuss the complexity of this procedure. We break the complexity into two parts: the cost of performing the weighted split median operation, and the cost of migrating the events. One application of the weighted split median has time complexity $O(D \log n)$, where D is the diameter of the region, and messages sent per node of $O(\log^2 n)$ messages. Thus, we get time complexity $O(D \log^2 n)$ and messages sent per node of $O(\log^3 n)$ for all of the applications of weighted split median. Every period re-balance requires each event to travel at most twice the diameter of the network (assuming that GPSR routes on a direct line). The total number of events that can be forced to migrate as a result of k new events being stored is $O(k \log k)$. Thus, the amortized number of migrations per event is logarithmic, $O(\log k)$ in the number of insertions. This amount of re-balancing per insertion is required for any standard dynamic data structure (e.g. 2-3 trees, AVL trees, etc.).

Figures 6 and 7 show a detailed example illustrating how KDTR works. Continuing on the same example we presented in Section 4.2, we monitor how KDTR maintains the K-D tree balancing in the course of successive insertions. Starting with an equal number of 3 events stored at each sensor, a storage hot-spot arises in node $N7$ after 6 event insertions. By checking the ratio of $N7$ storage to that of $N7$, KDTR identifies the subtree rooted at node 11 as an unbalanced subtree. As none of node 11's ancestors is unbalanced at this point, KDTR selects 11 to be re-balanced. However, the storage load remains skewed toward subtree 1, thus, after another 6 insertions, KDTR re-balances the subtree rooted at 1. After 12 more insertions aiming the right subtree of the root, KDTR re-balances the root of the tree, basically changing the attribute-based split points of almost all internal nodes, in order to maintain the balance of the tree. Note that, as long as the average loads of sensors which are falling outside the hot-spot area increases, the frequency of re-balancing decreases.

We digress slightly to explain a method that one might use to trigger re-balancing, as opposed to fixed time period re-balancing. Each sensor s_i knows the number of events that are stored in each region corresponding to an ancestor of s_i in the K-D tree when this region was re-balanced. Let C_j be the number of events at the last re-balancing of the region R_j corresponding to node of depth j on the path from the root to s_i in the K-D tree. Assume that the region R_j has elected a leader s_j. Then, the number of events that have to be stored in R_j, since the last re-balancing, to cause another re-balancing in R_j is something like hC_j, where h is the unbalancing ratio that we are willing to tolerate. Then, each insertion to s_i is reported by s_i to s_j with probability something like $\Theta\left(\frac{\log n}{hC_j}\right)$. Thus, after seeing $\Theta(\log n)$ such notifications, the leader s_j can be confident that there have been very close to hC_j insertions into the region R_j, and a re-balancing might be warranted. Note that the role of leader requires only receiving $O(\log n)$ messages.

KDTR steps with re-balancing ratio h=3



	N0	N1	N2	N3	N4	N5	N6	N7
Initial Sensor Loads:	3	3	3	3	3	3	3	3
Loads after 6 insertions:	3	3	3	3	3	3	3	9

Re-balance sub-tree (11): change the value of **X₄** to **60**

	N0	N1	N2	N3	N4	N5	N6	N7
Loads after re-balancing:	3	3	3	3	3	3	6	6
Loads 6 more insertions:	3	3	3	3	3	3	9	9

Re-balance sub-tree (1): change the value of **Y₂** to **1.7**, change the value of **X₃** to **55**, and keep **X₄** unchanged

	N0	N1	N2	N3	N4	N5	N6	N7
Loads after re-balancing:	3	3	3	3	6	6	6	6
Loads 12 more insertions:	3	3	3	3	9	9	9	9

Re-balance (root): change the value of **X₅ to *50*,** change the value of **X₃** to **59**, change the value of **X₄** to **65**, change the value of **X₁** to **40**, change the value of **X₂** to **40**, and keep **other split points** unchanged

	N0	N1	N2	N3	N4	N5	N6	N7
Loads after re-balancing:	6	6	6	6	6	6	6	6

Figure 7: KDTR example

6. EXPERIMENTAL RESULTS

In order to evaluate our KDDCS scheme, we compared its performance with that of the DIM scheme, that has been shown to be the best among current INDCS schemes [9].

In our simulation, we assumed having sensors of limited buffer and constrained energy. We simulated networks of sizes ranging from 50 to 500 sensors, each having an initial energy of 50 units, a radio range of $40m$, and a storage capacity of 10 units. For simplicity, we assumed that the size of a message is equal to the size of a storage unit. We also assumed that the size of a storage unit is equal to the size of an event. When sent from a sensor to its neighbor, a message consumes 1 energy unit from the sender energy and 0.5 energy unit from the receiver energy. The service area was computed such that each node has on average 20 nodes within its nominal radio range.

As each sensor has a limited storage capacity, it is assumed to follow a *FIFO* storage approach to handle its cache. Thus, a sensor replaces the *oldest* event in its memory by the newly incoming event to be stored in case it is already full when receiving this new event.

We modeled a network of temperature sensors. The range of possible reading values was $[30, 70]$. We modeled storage hot-spots by using a random uniform distribution to represent sensors' locations, while using a skewed distribution of events among the attributes ranges. Note that the regular sensor deployment assumption does not affect our ability to assess the effectiveness of KDDCS as the storage hot-spot can result from either skewed sensor deployments, or skewed data distributions, or both. The storage hot-spot size is characterized by the *skewness dimensions*, which are the percentage of the storage hot-spot events to the total number of events generated by the sensor network and the percentage of the read-

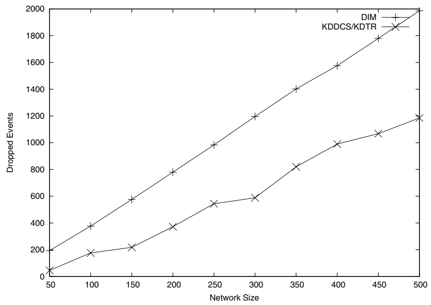


Figure 8: Number of dropped events for networks with a 80%-10% hot-spot

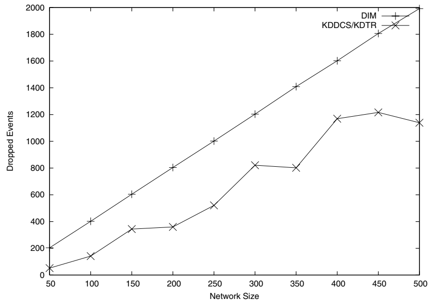


Figure 9: Number of dropped events for networks with a 80%-5% hot-spot

ings' range in which the hot-spot events fall to the total possible range of temperature readings. We assumed that a single storage hot-spot is imposed on the sensor network. To follow the behavior of KDDCS toward storage hot-spots of various sizes, we simulated, for each network size, a series of hot-spots where a percentage of 10% to 80% of the events fell into a percentage of 5% to 10% of the reading' range. Note that we always use the term x%-y% hot-spot to describe a storage hot-spot where x% of the total generated events fall into y% of the readings' range.

We used a uniform split points initialization to setup the attribute range responsibilities of all internal nodes of the K-D tree. For the re-balancing threshold, we used a value of 3 to determine that a specific subtree is unbalanced. Node failures were handled in the same way as DIM. When a node fails, its stored events are considered lost. Futher events directed to the range responsibility of such node are directed to one of its close neighbors.

We ran the simulation for each network size and storage hot-spot size pair. Each simulation run consisted of two phases: *the insertion phase* and *the query phase*. During the insertion phase, each sensor generates (i.e. reads) 5 events, according to the predefined hot-spot size and distribution, and forward each of these event to its owner sensor. In the query phase, each sensor generates queries of sizes ranging from 10% to 90% of the $[30, 70]$ range. The query phase is meant to measure the damages, in terms of QoD and energy losses, caused by the storage hot-spot.

The results of the simulations are shown in the Figures 8 to 17. In these figures, we compare the performance of our KDDCS scheme versus that the DIM scheme with respect to various performance measures. Note that we only show some of our findings due to space constraints.

R1. Data Persistence: Figures 8 and 9 present the total number events dropped by all network nodes in networks with 80%-10% and 80%-5% hot-spots, respectively. By analyzing the difference between KDDCS and DIM, we can find out that the number of dropped events in the first is around 40% to 60% of that in the second. This can be interpreted by the fact that KDDCS achieves a better load balancing of storage among the sensors. This leads to decreasing the number of sensors reaching their maximum storage, and decreasing the total number of such nodes compared to that in the pure DIM. This directly results in decreasing the total number of dropped events and achieving a better data persistence.

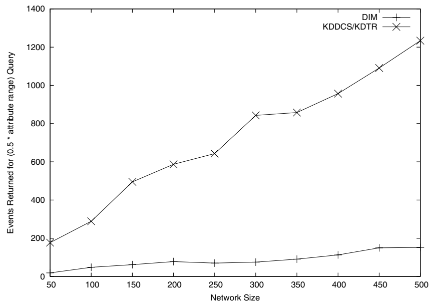


Figure 10: Query size of a 50% query for networks with a 80%-10% hot-spot

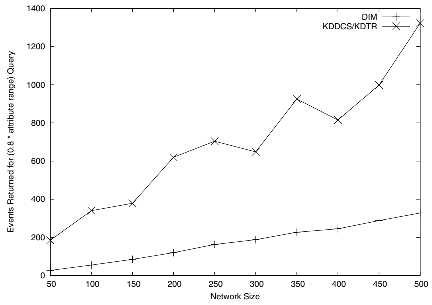


Figure 11: Query size of a 80% query for networks with a 80%-5% hot-spot

Another important remark to be noted based on the two figures is that decreasing the size of the hot-spot by making the same number of events to fall into a smaller attributes' range does not highly affect the overall performance of KDDCS compared to that of DIM.

R2. Quality of Data: Figures 10 and 11 show the average query sizes of 50% and 80% of the attribute ranges for networks with a 80%-10% and 80%-5% hot-spots, respectively. It is clear that KDDCS remarkebly improves the QoD provided by the sensor network. This is mainly due to dropping less information (as pointed at in R1), thus, increasing the number of events resulting in each query. The gap between DIM and KDDCS, in terms of resulting query sizes, is really huge for in both graphs, which indicates that KDDCS outperforms DIM for different storage hot-spot sizes.

This result has a very important implication on the *data accuracy* of the sensor readings output from a network experiencing a hot-spot. The success of the KDDCS in avoiding hot-spots results in improving the network ability to keep a higher portion of the hot-spot data. This ameliorates the degree of correctness of any aggregate functions on the network readings, for example, an average of the temperature or pressure values where a high percentage of the data is falling within a small range of the total attributes' range. We consider this to be a good achievement compared to the pure DIM scheme.

R3. Load Balancing: Figures 12 and 13 show the average node storage level for networks with 70%-10% and 60%-5% hot-spots,

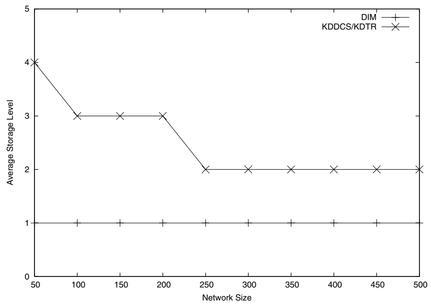


Figure 12: Average node storage level for networks with a 70%-10% hot-spot (numbers rounded to ceiling integer)

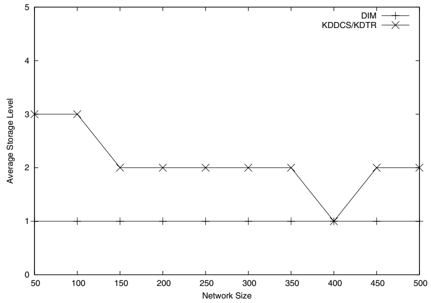


Figure 13: Average node storage level for networks with a 60%-5% hot-spot (numbers rounded to ceiling integer)

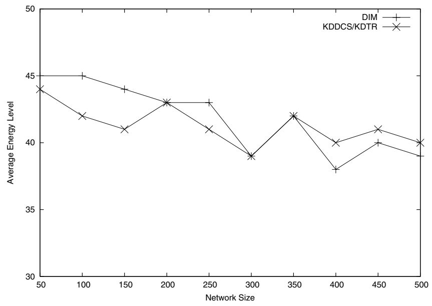


Figure 15: Average sensors' energy levels for networks with a 50%-5% hot-spot

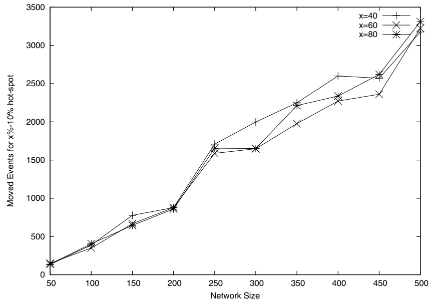


Figure 16: Number of event movements for networks with a x%-10% hot-spot

respectively. By a node storage level, we mean the number of events stored in the node's cache. The figures show that KDDCS has a higher average storage level than DIM, especially for less skewed hot-spots. This can be interpreted as follows. When a storage hot-spot arises in DIM, the hot-spot load is directed to a small number of sensors. These nodes rapidly reach their storage maximum, while almost all other sensor nodes are nearly empty. Therefore, the load distribution is highly skewed among nodes leadind to a low average storage level value. However, in KDDCS, the number of nodes effectively storing events increases. Subsequently, the average storage load value increases. This gives us a truthful figure about the better storage balancing the network. It is worth mentioning that each of the values in both figures is rounded to the ceiling integer. Thus, in both cases, the average in DIM does not exceed one event per sensor for all network sizes.

R4. Energy Consumption Balancing: Figures 14 and 15 show the average node energy level at the end of the simulation for networks with 70%-10% and 50%-5% hot-spots, respectively. The figures show that this average generally decreases with the increase of the network size for both schemes. The interesting result that these figures show is that both KDDCS and DIM result in fairly close average energy consumption among the sensors. However, as we mentioned in R3 and based on the way DIM works, most of the energy consumed in DIM is effectively consumed by a small number of nodes, namely those falling in the hot-spot region. On the other hand, the number of nodes consuming energy increases in KDDCS due to the better load balancing KDDCS achieves, while the average energy consumed by each sensor node decreases. Thus, although the overall energy consumption is the same in both KDDCS and DIM, this result is considered as a positive result in terms of increasing the *overall network lifetime*, as well as avoiding the early death of sensor nodes, which leads to avoid *network partitioning*.

R5. Events Movements: Figures 16 and 17 show the number of migrated events for networks with $x\% - 10\%$ and $x\% - 5\%$ hot-spots, respectively, where x varies from 40 to 80. For both sets of hot-spot sizes, the number of event movements lineraly increases with the network size. The important result to be noted in both

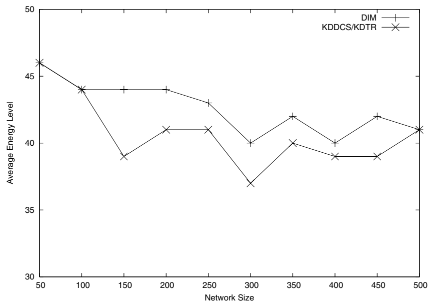


Figure 14: Average sensors' energy levels for networks with a 70%-10% hot-spot

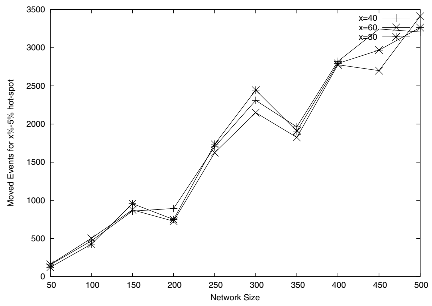


Figure 17: Number of event movements for networks with a x%-5% hot-spot

figure is that the total number of movements is not highly dependent on the hot-spot size. This is mainly because KDDCS avoids storage hot-spots in their early stages instead of waiting for a large storage hot-spot to be formed, and then try to decompose it. Therefore, most of the event movements are really done at the start of the formation of the storage hot-spot. This leads to the fact that, for highly skewed data distributions, (i.e. large hot-spot sizes), the number of event movements does not highly change with changing the exact storage hot-spot size.

7. RELATED WORK

Many approaches have been presented in literature defining how to store the data generated by a sensor network. In the early age of sensor networks research, the main storage trend used consisted of sending all the data to be stored in base stations, which lie within, or outside, the network. However, this approach may be more appropriate to answer *continuous queries*, which are queries running on servers and mostly processing events generated by all the sensor nodes over a large period of time [4, 10, 18, 14, 12, 11].

In order to improve the lifetime of the sensor network, as well as the *QoD* of ad-hoc queries, INS techniques have been proposed. All INS schemes presented so far were based on the idea of *DCS* [15]. These INDCS schemes differ from each other based on the events-to-sensors mapping used. The mapping was done using hash tables in DHT [15] and GHT [13], or using K-D trees in DIM [9].

The formation of storage hot-spots due to irregularity, in terms of *sensor deployment* or *events distribution*, represent a vital issue in current INDCS techniques [5]. Some possible solutions for *irregular sensors deployments* were highlighted by [5], such as routing based on virtual coordinates, or using heuristics to locally adapt to irregular sensor densities. Recently, some load balancing heuristics for the *irregular events distribution* problem were presented by [2, 8]. Such techniques were limited in their capability to deal with storage hot-spots of large sizes as they were basically acting like storage hot-spots detection and decomposition schemes, rather than storage hot-spots avoidance schemes like KDDCS. To the best of our knowledge, no techniques have been provided to cope with both types of irregularities at the same time. A complentary work to our paper is that on exploting similarities in processing queries issued by neighboring sensors in a DCS scheme [16].

Query Hot-Spots is another important problem that is orthogonal to the storage hot-spots problem. The problem arizes when a large percentage of queries ask for data stored in few sensors. We identified the problem in an earlier paper [1] and presented two algorithms, Zone Partitioning (ZP) and Zone Partial Replication (ZPR), to locally detect and decompose query hot-spots in DIM. We believe that KDDCS is able to cope with query hot-spots provided minor changes are added to the scheme. We aim at addressing this problem in the KDDCS testbed that we plan to develop.

Recently, Krishnamurthy et al. [7] presented a novel DCS scheme, called RESTORE, that is characterized by real time event correlation. It would be interesting to compare the performance of both KDDCS and RESTORE in terms of load balacing.

8. CONCLUSIONS

Sensor databases are becoming embedded in every aspect of our life from merchandise tracking, healthcare, to disaster responds. In the particular application of disaster management, it has been argued that it is more energy efficient to store the sensed data locally in the sensor nodes rather than shipping it out of the network, even if out-of-network storage is available.

The formation of *Storage Hot-Spots* is a major problem with the current INDCS techniques in sensor networks. In this paper, we presented a new load-balanced INDCS scheme, namely *KDDCS*, that avoids the formation of storage hot-spots arising in the sensor network due to irregular sensor deployment and/or irregular events distribution. Further, we proposed a new routing algorithm called *Logical Stateless Routing*, for routing events from the generating sensors to the storage sensors, that is competitive with the popular GPSR routing. Our experimental evaluation has confirmed that our proposed KDDCS both increases the quality of data and the energy savings by distributing events of the storage hot-spots among a larger number of sensors.

Acknowledgments

We would like to thank Mohamed Sharaf for his useful feedback. We would also like to thank the anonymous referees for their helpful comments.

9. REFERENCES

[1] M. Aly, P. K. Chrysanthis, and K. Pruhs. Decomposing data-centric storage query hot-spots in sensor networks. In *Proc. of MOBIQUITOUS*, 2006.

[2] M. Aly, N. Morsillo, P. K. Chrysanthis, and K. Pruhs. Zone Sharing: A hot-spots decomposition scheme for data-centric storage in sensor networks. In *Proc. of DMSN*, 2005.

[3] J. L. Bentley. Multidimensional binary search trees used for associative searching. In *CACM, 18(9)*, 1975.

[4] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of MDM*, 2001.

[5] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. In *Proc. of HotNets-II*, 2003.

[6] B. Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless sensor networks. In *Proc. of ACM Mobicom*, 2000.

[7] S. Krishnamurthy, T. He, G. Zhou, J. A. Stankovic, and S. H. Son. Restore: A real-time event correlation and storage service for sensor networks. In *Proc. of INSS*, 2006.

[8] X. Li, F. Bian, R. Govidan, and W. Hong. Rebalancing distributed data storage in sensor networks. Technical Report No. 05-852, CSD, USC, 2005.

[9] X. Li, Y. J. Kim, R. Govidan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proc. of ACM SenSys*, 2003.

[10] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of OSDI*, 2002.

[11] S.-J. Park, R. Vedantham, R. Sivakumar, and I. F. Akyildiz. A scalable approach for reliable downstream data delivery in wireless sensor networks. In *Proc. of MobiHoc*, 2004.

[12] T. Pham, E. J. Kim, and W. M. Moh. On data aggregation quality and energy efficiency of wireless sensor network protocols. In *Proc. of BROADNETS*, 2004.

[13] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govidan, and S. Shenker. GHT: A grographic hash table for data-centric storage. In *Proc. of WSNA*, 2002.

[14] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *Proc. of MobiDE*, 2003.

[15] S. Shenker, S. Ratnasamy, B. Karp, R. Govidan, and D. Estrin. Data-centric storage in sensornets. In *Proc. of HotNets-I*, 2002.

[16] P. Xia, P. K. Chrysanthis, and A. Labrinidis. Similarity-aware query processing in sensor networks. In *Proc. of WPDRTS*, 2006.

[17] T. Yan, T. He, and J. A. Stankovic. Differentiated surveillance for sensor networks. In *Proc. of SenSys*, 2003.

[18] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proc. of CIDR*, 2003.