LOAD BALANCING HOTSPOTS IN SENSOR STORAGE SYSTEMS

by

Mohamed Aly

M.S. Computer Science, University of Pittsburgh, 2007

B.Sc. Computer and Systems Engineering, Alexandria University, Egypt, 2002

Submitted to the Graduate Faculty of
the Department of Computer Science in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2009

UNIVERSITY OF PITTSBURGH

DEPARTMENT OF COMPUTER SCIENCE

This dissertation was presented

by

Mohamed Aly

It was defended on

March 25, 2009

and approved by

Prof. Kirk Pruhs, Department of Computer Science

Prof. Panos K. Chrysanthis, Department of Computer Science

Prof. Alexandros Labrinidis, Department of Computer Science

Dr. Phillip Gibbons, Intel Research Pittsburgh

Dissertation Advisors: Prof. Kirk Pruhs, Department of Computer Science,

Prof. Panos K. Chrysanthis, Department of Computer Science

ABSTRACT

LOAD BALANCING HOTSPOTS IN SENSOR STORAGE SYSTEMS

Mohamed Aly, PhD

University of Pittsburgh, 2009

Sensor networks provide us with the means of effectively monitoring and interacting with the physical world. A sensor network usually consists of a large number of small inexpensive battery-operated sensors deployed in a geographic area. This dissertation considers a sensor network deployed to monitor a disaster area. First responders continuously issue *ad-hoc* queries while moving in the disaster area. In such an environment, it is often more beneficial to store sensor readings and process ad-hoc queries within rather than outside the sensor network. Recently, this led to an increased popularity of *Data-Centric Storage (DCS)*. A DCS scheme is based on a mapping function from readings to sensors based on the attribute values of each reading. This mapping function defines the DCS *index structure*.

Two significant problems arising in this DCS network model due to data and traffic skewness are *storage hotspots* and *query hotspots*. Storage hotspots are formed when many sensor readings are mapped for storage to a relatively small number of sensor nodes. Query hotspots occur when many user queries target few sensor nodes. Both types of hotspots are hard to predict. Storage hotspots result in an uncontrolled reading shedding that decreases the Quality of Data (QoD). Due to the limited wireless bandwidth of sensors, hotspots decrease QoD by increasing collisions (thus losses) of reading/query packets. When lasting long enough, hotspots affect the Quality of Service (QoS) by unevenly depleting energy in the sensor network.

This dissertation addresses both problems of hotspots through load balancing. The main dissertation hypothesis is that data migration resulting from local or global load balancing of

the DCS index structure can effectively solve the hotspot problems. The contributions of this dissertation lie in developing two schemes, namely, the Zone Sharing/Zone Partitioning/Zone Partial Replication (ZS/ZP/ZPR) scheme and the K-D tree based Data-Centric Storage (KDDCS) scheme. ZS/ZP/ZPR detects and decomposes both types of hotspots through load balancing in the hotspot area. KDDCS avoids the formation of hotspots through globally load-balancing the underlying DCS index structure. Experimental evaluation shows the effectiveness of the proposed schemes in coping with hotspots in comparison to the state-of-the-art DCS schemes.

Keywords Sensor Networks, Distributed Algorithms, Data-Centric Storage, Load Balancing, Storage Hotspots, Query Hotspots.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

With my PhD journey coming to an end, I would like to thank Allah, the Almighty God, for blessing me with the opportunity, time, health, and strength, to conduct this PhD research and put together this dissertation.

I would like to thank my advisors, Kirk Pruhs and Panos Chrysanthis, for their guidance and support throughout my PhD study. I learned a lot from their knowledge, advice, dedication, and wisdom. Individually and collectively, they taught me things that I cannot enumerate on the academic, professional, and personal levels. Kirk and Panos, I cannot find enough words to thank you.

I would also like to extend my gratitude to my committee members, Alexandros Labrinidis and Phillip Gibbons, for their support, thoughtfulness, and their insightful and constructive advice and feedback.

I am grateful to my friends and colleagues in the PhD program, especially Mahmoud Elhaddad, Sherif Khattab, and Mohammad Hammoud, for their continuous help and support. I also appreciate my successful research collaboration with Nicholas Morsillo, John Augustine, Adel Youssef, and Anandha Gopalan.

Furthermore, I would like to thank the faculty and staff of the Computer Science Department at Pitt for the warm and familial departmental environment.

Acknowledgment is also due to the NSF for supporting my research through grants: NI-0325353, CCF0448196, CCF-0514058, and IIS-053453. Also, I would like to acknowledge the support that I received from the Andrew Mellon Doctoral Fellowship.

Finally, the most appreciation is due to my beloved family: my father Abdel-Mohsen, my mother Amal, my sister Naglaa, my brother Tarek, my wife Khadiga, and my newly-born son, Mouaaz. With gratitude, I dedicate this work to them.

1.0 INTRODUCTION

1.1 MOTIVATION

Sensor networks are autonomous self-organizing systems providing us with the means of effectively monitoring and interacting with the physical world. A sensor network usually consists of a large number of small inexpensive battery-operated sensors deployed in a geographic area defined as the *service area*. Sensors continuously gather different types of information from the environment. A *sensor reading* is composed of one or more attributes (e.g., temperature, carbon monoxide level, etc.), the identity of the sensor device sensing it, and the time it was sensed on. Periodically, sensor readings or their synopses may be directed to the *base station*, a server located within or nearby the service area, for further storage and processing. Sensor nodes are characterized with power, memory, and computational constraints. They are usually stationary unlike mobile ad-hoc network nodes. Sensor networks are used in many real-world applications such as habitat monitoring [54], surveillance [57], disaster management [7], inventory management [40], etc.

In this work, we mainly focus on sensor networks for disaster management. Consider an emergency/disaster scenario where sensors are deployed in the area of the disaster [58]. Sensor devices continuously sense and store readings of potential interest. As first responders move through the disaster area with hand-held devices that can directly communicate with sensor devices, they can query the network about recent readings. For example, a first responder might ask for the locations (or IDs) of all sensor nodes that recorded high carbon monoxide levels in the last 15 minutes, or the first responder might ask whether any sensor node detected movement in the last minute. Queries are picked up by sensor devices in the region of the first responder and 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 are generally *multi-dimensional range queries* [37], 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. As these queries arise from within the service area and target fairly small amounts of sensor readings, the base station storage technique, comprised of directing all sensor readings to be stored in base stations and all queries to be processed by these base stations, may not be the optimal storage and query processing technique for this environment. In-Network Storage (INS) is a storage technique that has been specifically proposed to efficiently process ad-hoc queries. INS involves storing events *locally* in the caches of the sensor nodes (in other words, using the sensor devices as storage devices). It is more beneficial to store data in the network because it is more efficient than shipping all the data (i.e., raw sensor readings) out of the network (to base stations), or simply because no out-of-network storage is available.

INS may take many forms depending on how to map readings to sensors for storage. The straightforward INS technique is *local storage* which consists of letting each sensor store the readings it generates. In this case, processing any range query will require flooding the whole network with retrieval messages, which is both energy and time consuming. Due to the query processing inefficiency of this scheme, INS schemes already presented in literature involved targeted data retrieval through adopting the *Data-Centric Storage (DCS)* concept [53], which is based on the traditional database concept of *access paths* (or *indexing*) [19]. A DCS scheme is based on a function from readings to sensors mapping each reading to a *storage sensor* based on the value of the attributes of this reading. The reading-to-sensor mapping defines the DCS *index structure*. Based on this index structure, each storage sensor will be responsible for a given attribute value range. This responsibility includes both storing any sensor reading (i.e., generated by any sensor) whose attributes fall in and answering any query targeting this value range. The storage sensor may be different than the sensor that originally generated the event.

1.2 PROBLEM DEFINITION

In a DCS sensor network model, two major problems may arise, namely *storage hotspots* and *query hotspots*. Storage hotspots represent a form of data skewness occurring when the distribution of readings is non-uniform with respect to the possible value ranges of the different attributes. This skewed reading distribution results in mapping many sensor readings for storage to a fairly small number of sensor nodes. Consequently, this results in forming one or more hot regions of attribute values (and subsequently of sensor devices). Both the hot value ranges and the hot geographic regions can be referred to as the *storage hotspots*. Storage hotspots can lead to increasing the reading shedding as the storage sensors (falling in the storage hotspots) become overloaded with data very quickly and start dropping readings once exceeding their storage capacities. This results in decreasing the *Quality of Data (QoD)* of the sensor network. We define the *QoD* to be the average proportion of the effective result size of any query (measured in terms of the number of sensor readings) to the number of sensor readings actually fulfilling the query.

Furthermore, storage hotspots result in some form of traffic skewness as a fairly small number of sensor nodes are frequently targeted for storing readings. As each sensor node has a limited wireless bandwidth, many of the packets (carrying readings) targeting nodes falling in a storage hotspot tend to be dropped due to wireless collisions. This further decreases QoD. Additionally, this traffic skewness imposes a high energy consumption overhead on sensor nodes falling in the hotspot area. Unless the number of routing paths used by the underlying routing algorithm to access the hotspot area is large, hotspots impose a high energy consumption overhead on sensor nodes falling on these routing paths as well. In case a hotspot lasts for enough time, these overloaded sensor nodes quickly deplete their energy and start dying before other sensor nodes possibly having almost full batteries. Node deaths affect QoD as the data stored in the dead nodes would be lost. Additionally, node deaths significantly affect the Quality of Service (QoS) of the sensor network by decreasing the network lifetime and may additionally result in partitioning the network. As node deaths decrease the number of functional sensors in the network, this decreases the coverage capability of the network, which results in an even larger decrease of its QoS.

Query hotspots represent another form of data skewness occurring where most of the mobile users issue queries falling, with a high percentage, in a small sub-range of the possible attribute value ranges. Consequently, most of these queries target a fairly small subset of sensor nodes in the sensor network. Query hotspots may easily occur in our setting due to the time-varying number of users and may also arise because of the difference in *popularity* between the readings of different sensor nodes based on the reading type, location, time, etc. Similar to storage hotspots, query hotspots affect the QoD of the sensor network. Many of the packets targeting nodes falling in a query hotspot tend to be dropped due to wireless collisions. As most of these packets represent queries, this results in increasing the average number of attempts incurred in sending packets in the hotspot area. Subsequently, a query whose results partially or fully fall in the hotspot area would tend to be partially or fully unanswered. This has a direct effect on decreasing the QoD of the sensor network. Also, similar to storage hotspots, node deaths that may result from query hotspots have negative effects on both QoD and QoS.

In this dissertation, we focus on the two problems of storage and query hotspots assuming a failure-free environment (where nodes only fail because of battery depletion). The formation of hotspots significantly affect the QoD of the sensor network. Additionally, it may have a secondary effect on decreasing the QoS of the sensor network.

1.3 APPROACH AND CONTRIBUTIONS

In this dissertation, we address the storage and query hotspot problems in the disaster management DCS sensor network model by means of *load balancing*. Our main *hypothesis* in this thesis is that *data migration* resulting from load balancing the underlying index structure of a DCS scheme is effective in solving the hotspot problems (both storage and query hotspots) arising in that scheme.

Our contributions in this thesis are as follows:

- We present two hotspot detection and decomposition solutions, namely the Zone Sharing (ZS) [8] and the Zone Partitioning/Zone Partial Replication (ZP/ZPR) [5] schemes, to

individually identify and decompose storage and query hotspots, respectively. Both schemes assume an underlying k-d tree based index structure and are implemented on top of the Distributed Index for Multi-dimensional data (DIM) DCS scheme [37].

- We blend the two solutions to form the ZS/ZP/ZPR scheme to detect and decompose both types of hotspots, either separately or simultaneously.
- We develop a generic *globally load-balanced* DCS scheme, namely the K-D tree based Data-Centric Storage (KDDCS) scheme [9], that is able to *avoid* the formation of either type of hotspots. We apply KDDCS to *individually* cope with storage and query hotspots.
- We develop a systematic approach for using KDDCS to *simultaneously* avoid both types of hotspots.

To measure the efficiency of our schemes, we experimentally study the performance of our schemes for the different hotspot types through extensive simulations. When designing our simulations, we tried as much as possible to model real-world hotspot scenarios possibly arising in disaster management applications. In our simulations, we use different performance metrics. Our metrics lie in three main categories: *QoD*, *load balancing*, and *energy consumption*. The concentration on these categories comes as a direct consequence of our main goal in this thesis, which is to improve the QoD for hotspots of different types, through load balancing, with a secondary goal of improving energy consumption (or at least, without imposing an additional energy consumption overhead). Example of metrics we use are the number of dropped sensor readings, the average node storage, the average result size of given types of queries, the number of full nodes (i.e., with caches full with readings), the number of dead nodes, and the average energy consumption per sensor node.

Our experimental evaluation showed that the localized load balancing achieved by the ZS/ZP/ZPR scheme is best suited for detecting and decomposing small to medium isolated hotspots. For these hotspot settings, ZS/ZP/ZPR achieves good QoD improvements while introducing a small energy consumption overhead on the sensor network. On the other hand, the global load balancing achieved by KDDCS offers higher QoD improvements for the different hotspot settings while imposing a slightly larger energy consumption overhead than that imposed by ZS/ZP/ZPR. More importantly, KDDCS copes with complex hotspot settings, such as large-scale single hotspots, multiple simultaneous static hotspots of either

similar or different types, and multiple moving hotspots. Furthermore, the KDDCS QoD improvements become more tangible for networks of large sizes. In summary, KDDCS is the best selection for a sensor network to maximize QoD for the different hotspot types and settings while paying a moderate energy consumption cost.

In light of these recommendations, KDDCS can be considered as the best fitting scheme for our motivating disaster management application. Recall that first responders use the sensor network data to better manage the disaster. A safe assumption about our application is that the expected network lifetime is relatively short, ranging from few hours to few days. Driven by the urgency of the sensor network data, data loss negatively affects the network performance much more than energy loss. Thus, it is better to sacrifice energy consumption for achieving the best possible QoD throughout the short network lifetime. As KDDCS maximizes the QoD for the different hotspot types while imposing an affordable energy consumption overhead on the sensor network, it is the best candidate for a disaster management application or any other application with similar characteristics such as habitat monitoring of rare phenomena and data-sensitive short-term surveillance applications.

1.4 ROADMAP

The rest of this dissertation is organized as follows. We start by describing the state-of-the-art DCS schemes (with a special concentration on the DIM scheme representing the basis of our local hotspot detection and decomposition schemes), load balancing paradigms, and point-to-point routing schemes used in sensor networks in Chapter 2. We then explain our system model and experimental platform in more details in Chapter 3.

In Chapter 4, we present our local storage hotspot detection and decomposition schemes to cope with the different types of hotspots. All schemes presented in this chapter run on top of the DIM DCS scheme. We start by presenting our ZS scheme to cope with storage hotspots. ZS locally detects a storage hotspot (in its formation location) and decomposes it by migrating sensor readings away from the hotspot area without globally changing the underlying k-d tree of the DIM scheme.

We continue by presenting the ZP/ZPR scheme to detect and decompose query hotspots arising in the DIM scheme. The scheme is based on the local detection of query hotspots and applying the data migration concept to decompose such hotspot without affecting the k-d tree of the DIM scheme. To locally detect and decompose mixed hotspots, both schemes are blended into the ZS/ZP/ZPR scheme. All three schemes, ZS, ZP/ZPR, and ZS/ZP/ZPR are experimentally evaluated for the different types of hotspots through extensive simulations.

Then, in Chapter 5, we present KDDCS, a globally load-balanced DCS scheme whose main goal is to avoid the formation of hotspots of different types and sizes. KDDCS continuously maintains the load balancing of the underlying k-d tree when mapping sensors to virtual addresses at the start of the network operation, as well as when mapping readings to storage sensors throughout the network lifetime. Due to the generality of its building components, we present different KDDCS versions to avoid storage hotspots, query hotspots, and mixed hotspots. We experimentally evaluate the performance of KDDCS to show its efficiency against different hotspot types and settings.

At the end, we conclude this dissertation and highlight possible future work, both related and complementary to our thesis work in Chapter 6.

2.0 RELATED WORK

In this chapter, we will present a brief quick review of the work related to our schemes and already presented in literature. We start by reviewing the Data-Centric Storage (DCS) literature. Then, we give an overview on the previously proposed load balancing schemes. Finally, as all our schemes use point-to-point routing for reading/query routing, we review the point-to-point routing protocols widely used in sensor networks.

2.1 DATA-CENTRIC STORAGE (DCS)

Many approaches have been presented in literature defining how to store the data generated by a sensor network. One line of work consisted in sending aggregated readings to be stored in base stations, lying within, or outside, the network. However, this approach is more appropriate to *continuous queries* running on servers and mostly processing events generated by all the sensor nodes over a large period of time [12, 38, 59, 52, 44, 43]. In order to improve the *QoD* of ad-hoc queries, In-Network Storage (INS) has been proposed. In INS, sensor caches are used as temporary storage devices to store sensor readings. All INS storage schemes already presented in literature were based on the DCS concept [53]. In general, DCS schemes differ from each other based on the readings-to-sensors mapping used. For example, this mapping was done using hash tables in the Geographic Hash Table (GHT) scheme [53,48] and using k-d trees in the Distributed Index for Multi-dimensional data (DIM) scheme [37].

The first DCS scheme presented in literature was GHT. The scheme basically hashes keys into geographic coordinates, and stores a key-value pair at the sensor node geographically

nearest the hash of its key [48]. The system replicates stored data locally (at surrounding sensor nodes) to ensure persistence when nodes fail. It uses an efficient consistency protocol to ensure that key-value pairs are stored at the appropriate nodes after topological changes (caused by node failures, both temporary or permanent, node movements, or any other external factors). By replicating values in different locations across the service area, GHT further distributes *regular* storage and query loads throughout the network. Data is routed to sensor nodes using the Greedy Perimeter Stateless Routing (GPSR) algorithm [32] (described in Section 2.3). In light of our problems, GHT does not provide any special handling for data skewness occurring in the forms of storage and query hotspots. Furthermore, the hashing technique of GHT does not implement any special support for multi-dimensional range queries. Processing such queries occurs through separate processing for each of the query dimensions followed by joining results of the different dimensions.

To efficiently process multi-dimensional range queries by leveraging the benefit of a locality-preserving hash in storing data, DIM presented the novel idea of using k-d trees [11] as the underlying DCS index structure. To map readings to sensors, DIM uses a k-d tree where the leaves $\mathcal{L}$ form a partition of the service area, and each element of $\mathcal{L}$ contains either zero or one sensor. The formation of the k-d tree consists of rounds. Initially, $\mathcal{L}$ is a one element set containing the whole service area. In each odd/even round r, each region $L \in \mathcal{L}$ that contains more than one sensor is bisected horizontally/vertically. Every time 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. Upon generating a reading, a sensor maps the reading to a binary code based on a repetitive fixed uniform splitting of the attribute ranges in a round robin fashion. Let the reading be consisting of only one attribute, say temperature. Then, the high order bits of a temperature reading 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 becomes the storage sensor of this reading. Due to the regularity of regions in this k-d tree, the reading is routed from the generating sensor to the storage sensor using GPSR. Experimental results have shown that DIM scheme exhibits better performance than the GHT scheme in processing ad-hoc range queries.

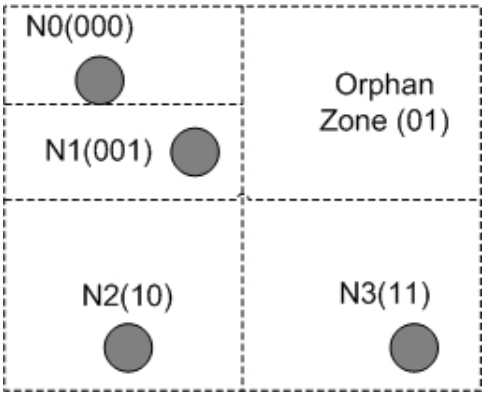


Figure 1: Sensor to Bit-code Mapping

Figures 1 and 2 show an example of a deployment of four sensors and the DIM k-d tree for that deployment, respectively. Figure 1 shows the initial sensor to bit-code mapping that assigns a bit-code to each sensor based on the sensor geographic locations. In Figure 2, X and Y represent temperature and pressure, respectively. The possible value range for temperature is $[30, 70]$ while that of pressure is $[0, 2]$. The figure shows the sensor storage responsibility assignments resulting from the value range to bit-code mapping. In the figure, each sensor is assigned two sub-ranges of values, one for temperature and one for pressure. Each sensor is supposed to store all readings whose values fall in the sensor's assigned sub-ranges. The orphan zone (01) is assumed to be delegated to node 001, which is the least loaded among its neighbors. Periodic messages are exchanged between sensor nodes to maintain the DIM k-d tree structure. As for GHT, DIM did not have load balancing as a major design goal. Thus, its performance was poor against skewed data sets [37].

In a follow-up work for DIM, Gummadi et al. [**?**] evaluated the effect of the data organization on DIM. The authors have shown that storing sensor readings in separate DIMs, one for each attribute, works better than storing the readings in one DIM based on all attributes. The authors then used this observation in order to build energy efficient query processing techniques by optimizing the joins involved in processing any of the queries and caching previous query results in query issuers (sensor nodes that issued those queries) and/or query processors (sensor node that were involved in processing queries) in order to avoid repeating

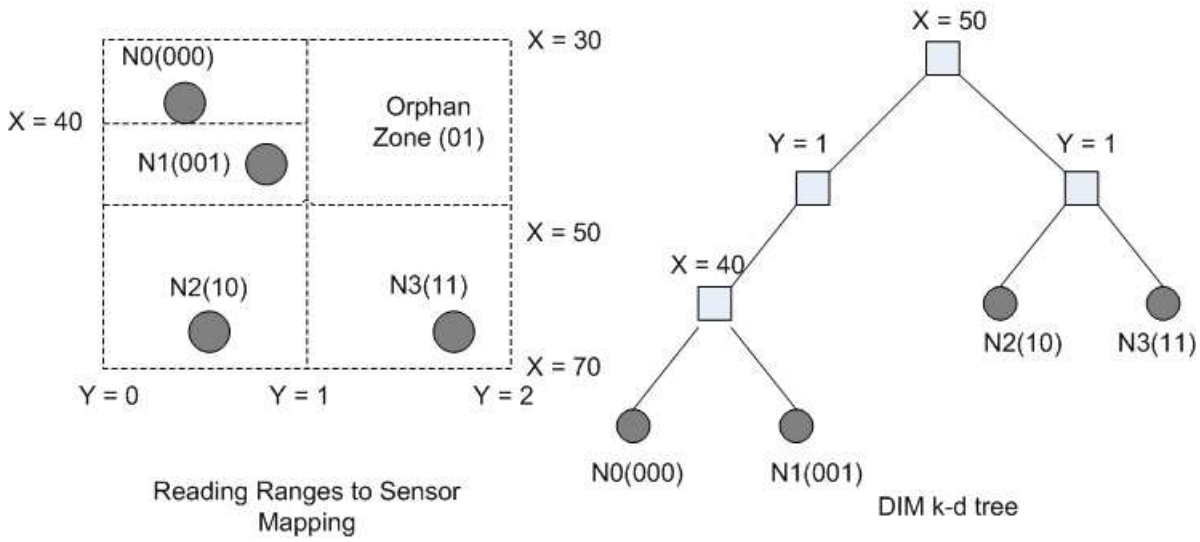


Figure 2: Assigning Value Ranges to Sensors

most of these joins for future queries.

Building on GHT and DIM, different DCS schemes have been presented in literature with different goals. None of these schemes considered load balancing as a major design goal. For instance, the practical data-centric storage scheme, pathDCS [18], had the main design goal of avoiding the dependence of DCS on point-to-point routing. pathDCS aimed at making DCS a basic primitive available to the different sensor network applications by using only tree-based communication primitives. This was based on the fact that standard tree construction algorithms are already available in many real-world deployments unlike point-to-point routing. The design of pathDCS relied on associating data names with paths instead of sensor nodes. These paths are derived from a collection of trees initially constructed at the beginning of the network operation starting from a small subset of sensors, namely *landmarks*. The scheme did not provide any special handling for data skewness or traffic skewness. Due to the static assignments of storage responsibilities and the static routing paths, the scheme is not suited for handling storage and/or query hotspots.

The Center Mapping Data Centric Storage (CMDCS) was developed to address the issues of data storage and query processing [25]. CMDCS supports the power efficient processing of event queries by preprocessing of the observed data (i.e., the generated readings) to check

whether these readings satisfy the occurrence conditions of events under concern. Once an event occurs, the event information is stored in the sensor network while the irrelevant readings are ignored. Though this approach has the benefit of solely focusing on storing events rather than raw sensor readings, it may limit the sensor network's ability to process queries for non-supported events once the raw data has been partially discarded. Despite the fact of storing less sensor readings, the scheme suffers from both storage and query hotspots. A hotspot scenario occurs in CMDCS when one event occurs (or the sensor network is queried for such event) much more frequently than the other ones belonging to the set of events under concern. This results in the formation of storage/query hotspots and CMDCS does not provide any special handling for such cases.

All previously discussed DCS schemes did not consider security issues resulting from the lack of tamper-resistance of the sensor nodes and the unattended nature of the sensor network. For example, an attacker may simply locate and compromise the node storing the event of his interest. To address these security problems, a privacy-enhanced DCS network, namely pDCS [51], has been recently developed to offer different levels of data privacy based on different cryptographic keys. pDCS proposed several query optimization techniques based on Euclidean Steiner Tree and Keyed Bloom Filter to minimize the query overhead while providing certain query privacy. These techniques do not target load balancing as a design goal. Thus, they are not suited for dealing with storage and query hotspots. In fact, the improved query optimization techniques presented in pDCS can be adopted by our KDDCS scheme as a way to improve query processing. This can be considered as a future work.

Finally, the advances in flash memory has lead to seriously consider the option of equipping sensors with high-capacity and energy-efficient local flash storage [61, 16]. Recently, the authors of [16] presented StonesDB, a sensor network database architecture, to enable this vision through a number of innovative features including energy efficient use of flash memory, multi-resolution storage and aging, query processing, and intelligent caching. In general, increasing the storage capacities of sensor nodes through the use of flash memories can reduce reading shedding resulting from storage hotspots but it does not necessarily eliminate it. This may decrease the negative impact that storage hotspots have on QoD. However, flash memories do not reduce the negative impacts that hotspots have on both QoD as a result

of increasing wireless collisions and node deaths. Furthermore, the use of flash memories does not improve the unbalanced energy consumption caused by hotspots and thus does not reduce the negative effects that hotspots impose on QoS.

2.2 LOAD BALANCING PARADIGMS

In general, load balancing was not the main design goal for previous DCS schemes although irregularities have been classified as a major issue in DCS schemes [23]. Irregularities arise because of irregular sensor deployments or skewed reading distribution. Ping et al. [56] suggested exploiting similarities when processing queries issued by neighboring sensors in a DCS scheme to improve the energy efficiency of query processing in DCS schemes, especially DIM. After presenting the DIM scheme [37], data migration has been suggested by [36] to load-balance storage hotspots arising in DIM because of the irregular events distribution problem. Due to its complexity and its heuristic nature, the solution was not completely successful in dealing with storage hotspots. Furthermore, it did not consider query hotspots at all. To our knowledge, no previous solutions have been presented in literature to cope with query hotspots targeting a DCS scheme.

In GHT, the programmer has no control on the level of redundancy of data. This may result in a great unbalance in the storage load imposed on each sensor, even when sensors are uniformly distributed. This problem consequently results in serious data losses, wasting energy, and shortening the overall lifetime of the sensor network. To cope with this problem, the Q-NiGHT protocol [4] (recently revised to be the DELIGHT protocol [3]) has been designed to run on top of GHT in order to provide a direct control on the level of QoS in the data dependability. The protocol uses a strategy similar to the rejection method [2] to build a hash function which scatters data approximately with the same distribution as that of sensors. As the scheme is mainly suited for dealing with the sensor deployment distribution, it does not take the reading and/or query load distribution into account. Thus, the effect of this scheme is limited when it comes to data skewness problems including both storage and query hotspots.

Recently, the Dynamic Data centric Storage mechanism (DDS) has been proposed with the goal of increasing robustness [34]. DDS is based on centrally keeping track of the distribution of the data generated by the network and dynamically adjusting the mapping from sensor readings to storage points through linear programming to reduce the storage cost and balance the storage and query workloads in the network. The main problems with this scheme are that it is centralized and that it assumes the existence of a base station in the sensor network. This cannot be assumed in a disaster management scenario. Additionally, this has the effect of incurring a large communication cost when relaying data from and to the base station. Additionally, this results in a lack of scalability. The scheme takes advantage of the GPSR routing protocol to store multiple copies of readings to improve the robustness of the network with little overhead.

Independently from DCS, load balancing represented a goal for some sensor network applications. Almost all previously presented load balancing paradigms performed on the level of packets as a mean for dealing with traffic skewness problems. Consequently, most of these schemes were embedded in routing protocols. Many of these protocols were based on multi-path routing, where a set of paths are determined for each packet type (defined by one or more sources and one or more destinations) prior to the network operation and paths are interchangeably used afterwards [50, 17]. Directed diffusion [28] presented the idea of finding multiple routes from multiple sources to a single destination while applying in-network data aggregation. Many multi-paths routing schemes were later presented based on Directed Diffusion, e.g., [22, 46, 10]. For example, Ganesan et al. [22] suggested the use of *braided multi-paths* to achieve high resilience and fault-tolerance. It is clear that all these techniques try to address the effects of traffic skewness. However, they have no effect on data skewness.

2.3 POINT-TO-POINT ROUTING

The need for point-to-point routing has recently increased as many current sensor network applications assume its usage, e.g., data-centric storage [53, 9] and multi-dimensional range

```
GPSR (Packet P, Current Node C, Final Destination D)
Begin
    1.  If (C == D)
    2.      End
    3.  If (Node N is closer to D than current node and N is neighbor to current node)
    4.      Forward P to N.
    5.  Else
    6.      Apply perimeter mode to determine the next node N'
    7.      Forward P to N'
    8.  End
End
```

Figure 3: The GPSR Algorithm

queries [37, 8, 5]. In point-to-point routing, a packet contains a destination address. When the sender sends the packet through its wireless network device, the packet is picked by the node whose address matches the packet destination address. Other nodes falling in the communication range of the sender ignore the packet. Early point-to-point routing schemes were based on *geographic routing*. The best known of these schemes is the Greedy Perimeter Stateless Routing (GPSR) protocol [32]. GPSR is a responsive and efficient routing protocol for mobile, wireless networks. Unlike routing algorithms established before GPSR, which mainly use graph-theoretic notions of shortest paths and transitive reachability to find routes, GPSR exploits the correspondence between geographic position and connectivity in a wireless network, by using the positions of nodes to make packet forwarding decisions. GPSR uses greedy forwarding to forward packets to nodes progressively closer to the destination. In regions of the network where such a greedy path does not exist (i.e., the only path requires that one temporarily moves further away from the destination), GPSR recovers by forwarding in perimeter mode (in which a packet traverses successively closer faces of a planar subgraph of the full radio network connectivity graph) until reaching a node closer to the destination, where greedy forwarding resumes. Figure 3 presents a high-level code of the GPSR algorithm applied on every node in the sensor network.

Several modifications have been presented to the basic GPSR algorithm to deal with some of the general challenges of geographic routing such as traffic skewness and obsta-

cles to radio propagation [31]. One of these modifications is the geographic provisioning technique. This modification lets GPSR use geographic forwarding via a *way-point* (not on the path found by basic GPSR) to distribute load in the network. This approach may be promising because on a wireless network, position and capacity are generally correlated; distributing load geographically leverages spatial reuse, and cuts the average load in regions where traffic is concentrated. Another modification is to extend GPSR to deal with obstacles to radio propagation. Obstacles introduce the risk that the planar subgraph used by GPSR's perimeter mode may not be connected. The modification uses both deterministic and randomized algorithms for recovering from such disconnections when they occur. Again, these modifications mainly target solving the traffic skewness problems. They are orthogonal the data skewness problems of both storage and query hotspots. As our main focus in this thesis is on developing comprehensive solutions for both the data and traffic skewness caused by hotspots, we decided to use the basic GPSR rather than its modifications throughout our experimental evaluations in order not to mask the negative traffic skewness effects hotspots.

Later, it was pointed that geographic schemes suffer from various limitations, e.g., the unrealistic requirement of GPS-equipped sensors [21, 33]. Driven by this problem, schemes like NoGeo [47] and GEM [42] suggested the use of synthetic *virtual* coordinates assigned by iteratively embedding nodes in a Cartesian plane. Two recent schemes, BVR [21] and Logical Coordinates [13], used a collection of ideas from both geographic and virtual coordinates schemes. The basic idea of both schemes is to let nodes obtain coordinates from a set of landmarks. Routing then minimizes a distance function on these coordinates.

2.4 SUMMARY

In this chapter, we presented the state-of-the-art of DCS, load-balancing, and point-to-point routing. Our local hotspot detection and decomposition schemes use geographic routing as they are built on top of DIM that routes packets using the basic GPSR algorithm. As for global load balancing, our proposed KDDCS scheme adopts the virtual coordinates routing paradigm in its underlying Logical Stateless Routing (LSR) algorithm.

3.0 SYSTEM MODEL AND EXPERIMENTAL PLATFORM

Before presenting our schemes in the following chapters, we need to highlight our underlying system model and discuss the experimental platform that will be used to evaluate the performance of our proposed schemes. Our system model and our experimental platform are presented in the next two sections, Section 3.1 and Section 3.2, respectively.

3.1 SYSTEM MODEL

In this section, we describe our system model. We start by describing our sensor network model and our motivating disaster management application. Then, we discuss the types of sensor devices that can be adopted by our sensor network to best serve the requirements of our motivating application.

3.1.1 Network Model and Motivating Application

Sensor network applications can be characterized into two types, namely *search/discovery* and *investigation*. Search/discovery sensor network applications consist of continuously monitoring a service area for the sake of studying one or more phenomena, e.g., habitat monitoring applications [54],. These applications are generally characterized by the relatively long expected network lifetime. Due to this fact, the major challenge for these networks is to improve Quality of Service (QoS) as much as possible, even at the cost of providing a lower Quality of Data (QoD). One the other hand, investigation applications consist of sensor networks running for relatively short time periods in order to investigate one or more

events of concern, e.g., disaster management applications [7]. The expected network lifetime in these cases is fairly short, e.g., ranging from few hours to few days. Due to the importance and the urgency of the event being investigated by the sensor network, QoD is a major goal in these applications even at the cost of decreasing QoS.

We mainly focus on a sensor network for a disaster/emergency management application. We consider a cluster of sensor devices spanning a limited geographic area, namely the *service area*, in the range of hundreds of meters. Sensor devices are assumed to be stationary and capable of wirelessly communicating among themselves (through the IEEE 802.5 standard communication protocol), as well as with other mobile computing devices such as cell phones, laptops, palms, and PDAs (possibly through one of the standard IEEE 802.11 protocols). At the start of the network operation, sensor devices organize themselves to start performing their given task, which is mainly monitoring and storing different types of information, e.g., temperature, pressure, etc. Sensors are battery-operated devices equipped with limited storage and communication capabilities (detailed device specifications presented in the next subsection). The existence of base stations, i.e., servers, is not mandatory in our sensor network.

The query load imposed to the sensor network is formed as follows. First responders to the disaster roam in the service area with their handheld computing devices. Using these devices, they issue ad-hoc queries which are mainly *range* queries. A range query usually queries the sensor network for readings whose values belong to a given attribute range $[a, b]$ during a specific time period $[t_1, t_2]$. As most sensor nodes nowadays have the capacity to generate and store multi-dimensional readings, the query may be asking for readings satisfying a given attribute range for each of the reading dimensions, e.g., $[a_1, b_1]$ for attribute 1 and $[a_2, b_2]$ for attribute 2, etc. Furthermore, the query may not necessarily be asking for actual readings, it can ask for the sensor(s) that sensed these readings, the number of these readings, or any other aggregate function on these readings, such as their average value, their median value, their minimum value, their maximum value, etc. A simple example for such a query would be querying all sensors mounted in the service area for the IDs of sensors that sensed a temperature of between 50 and 60 degrees Fahrenheit and a pressure between 1000 and 1010 millibars during the last 10 minutes.

First responders may be using a variety of mobile computing devices to interact with our sensor network. Cell phones are currently the most widespread among these devices, especially after the current advances in the mobile phones technology. In addition to cell phones, other mobile devices that may possibly be querying the sensor network could be PDAs, laptops, car computers, etc. Mobile devices may also be operating independently from any human user such as the case in robots. The number of mobile users/devices is time-varying, and hardly predicted at any time of the network operation. Consequently, the query load imposed on the sensor network, represented by both the number and the distribution of the queries accessing the sensor devices, is also variable and hard to predict.

The sensor network is responsible for efficiently processing all queries it receives. As discussed in the previous chapters, Data-Centric Storage (DCS) is adopted as the storage paradigm for our sensor network model. In a DCS scheme, a sensor reading generated by any sensor is forwarded for storage to a possibly different sensor node, namely the *storage sensor*, determined by mapping the reading attribute values to the space of sensor addresses (or geographical coordinates). Specifically, whenever a sensor s_1 measures a reading, it sends the reading tuple to be stored in another sensor s_2 which is determined based on the mapping function of the specific DCS scheme applied. s_2 only keeps this tuple for a limited amount of time due to memory limitations. A sensor can use a First-In-First-Out scheme (FIFO) to always keep the most recent readings in its cache. In the case that large reading histories need to be kept in the sensor caches, each cache can be split into two parts, one for storing the recent readings and the other to aggregate the elder readings in terms of window increments [20].

Before determining the appropriate device type to be used in our sensor network model, we first characterize disaster areas into two types, namely *permanent* and *temporary*. Permanent disaster areas can be forests permanently under fire risks, volcano locations, rivers frequently experiencing floods, and buildings located in areas under continuous risks of tornados or earthquakes. Possible temporary service areas are mainly areas experiencing a non-frequent emergency such as the world trade center location immediately after 9/11, the location of a fire or that of a plane crash, etc. Throughout our experimental evaluation, we mainly focus on temporary disaster areas.

3.1.2 Sensor Devices

Currently, wireless sensor network devices fall into two main categories; *mote-class* devices and *microservers*. Mote-class devices are tiny, inexpensive, resource-constrained devices that operate for long periods of time on battery power (originally designed at UC Berkeley in 2000 [27, 26]). Mote-class devices are mainly characterized by their low power consumption achieved primarily through using a micro-controller (MCU) and a low-power radio (e.g., CC1000 or CC2420). The current generation of motes includes the Mica2 and MicaZ [1] as well as the Telos [45]. Mica motes include an Atmel ATMega128L 8-bit micro-controller running at 8MHz that contains 4 KBytes of SRAM and 128 KBytes of Flash ROM. The Mica2 contains the CC1000 FM-based radio while the MicaZ has the 802.15.4-compliant CC2420 radio. On the other hand, the Telos uses a 16-bit MSP430 microcontroller with 10 KBytes of RAM and 60 KBytes of Flash ROM as well as a CC2420 802.15.4 radio. Some other mote-class devices include the Eyes node [55] and the Blue Mote [41].

The second category, microservers, is comprised of devices that mostly resemble an embedded PC platform. Microservers include a 32-bit CPU, megabytes of DRAM and up to several gigabytes of stable storage as well as a high-bandwidth 802.11 radio. Since microservers are similar to a PC platform, they include support for sophisticated peripherals as well as high data rate sensors. One of the first microserver platforms specifically designed to support wireless sensor networks was the Intel Stargate [29]. The Stargate has a 400MHz 32-bit PXA55 XScale processor, 64 MB of SDRAM, 32 MB of internal Flash and a Compact-Flash slot that can extend its storage to 1 GB or more. An Ethernet port as well as an 802.11 PCMCIA card provide its networking support. The Stargate operates on rechargeable lithium battery. Since the Stargate is fairly powerful in terms of computational resources, it is able to run a conventional operating system. The current generation of Stargates runs a Linux distribution (Familiar [24], based on a 2.4.19 kernel) and supports several file system types.

The choice of a specific category of sensor devices to be used in our disaster management application depends on different parameters. The first is the expected lifetime of the sensor network. In case the sensor network is intended to be deployed once and expected to

be operating for long times (such as the case for permanent disaster areas), the mote-class sensors will be more fitting for this scenario. However, this would mean that the DCS model that we adopt should be acting as a temporary storage for sensor readings rather than a permanent one due to the limited storage capability of the mote devices. Synopses of sensor readings may then be forwarded to base stations for permanent storage and processing. A second parameter determining the sensor device category can be the mobile devices designated to be used with the sensor network and their networking capabilities. Assuming that the main mobile devices used to communicate with sensor devices are cell phones, the specific networking protocol used by these cell phones, e.g., 802.11g or 802.15.4, will a factor determining the sensor types adopted by the sensor network.

A third parameter determining the actual device category adopted by the sensor network would be the sophistication level of the individual tasks of sensor nodes in the network. Generally speaking, mote-class devices are more suited for monitoring applications than microservers. This is mainly due to the simplicity of the task of each individual sensor in sensor networks used for monitoring applications. In our disaster management application, the main task of each sensor lies in monitoring and relaying information about the disaster area. In such a case, the capabilities of a sensor mote are sufficient for performing the required tasks. A fourth parameter is the expected cost for building and maintaining the sensor network. Again, the cost of microservers may be a constraint for using them in sensor network deployments, especially large-scale ones. A fifth parameter is the nature of the service area and whether the operation will be unattended or no. This implicitly affects the possibility of existence of a recharging mechanism for microservers or would totally avoid their usage in case of the unavailability of such a mechanism.

A sixth parameter would be the type of queries forming the query load and whether they contain predicates or joins that would require intensive computing to be processed. Complex queries may lead to choosing to add one or more microservers to the sensor network to be responsible for processing queries and delivering final results to the user. A final parameter can be the disaster type and the subsequent sensing rate. In general, emergencies would need much higher sensing rates than those needed for sensor networks deployed in permanent disaster areas where the sensor network can be considered as a long-term monitoring sensor

network. Therefore, microservers would be more fitting for these emergencies. The actual selection of the device category to be used by the sensor network would be a design choice taken by the network designers based on all the previously discussed parameters.

In general, the schemes and algorithms that we design in this dissertation are independent from the actual device category adopted by the sensor network. Our schemes continue to work well even for a sensor network composed of devices from both categories. For this scenario, storage loads may not be initially homogeneous among all sensors. Microservers may be the ones responsible for storing sensor readings. Other motes may temporarily store readings then relay them to microservers, or may simply not be responsible for storing any readings. A possibly different set, or a subset, of microservers, may act as gateways for communication with the mobile devices. Our schemes continue to be fitting for all these possible design options. Throughout the experimental evaluation of this thesis, we consider our network to be solely composed of motes.

3.2 EXPERIMENTAL PLATFORM

In order to measure the performance of our proposed schemes and compare their behavior to the corresponding state-of-the-art, we built a sensor network simulation testbed in which we implemented all our schemes as well as other state-of the-art schemes. We formulated our experimental evaluation to simulate an *investigation* sensor network deployed in a temporary disaster area for search and saving/recovery purposes. In this section, we describe the details of our experimental platform and the settings of our experiments.

3.2.1 Sensor Network

We simulated a sensor network spanning a disaster area (or service area) R. Sensors are assumed to be static, with limited amount of energy and storage resources. The exact sensor characteristics like the sensing and communication ranges, the storage capacity, and the starting energy budget are parameters to be set by the user when launching a new

experiment. To accommodate all types of sensor deployments, the sensor locations are either carefully picked, randomly generated, or generated based on an input distribution. Typically, we use a uniform distribution. The network is accessible by a variable number of mobile users. The queries generated by these users and mainly forming the querying load of the network represent an important parameter of the simulator. A detailed discussion about how we form our query load is presented in Section 3.2.5.

We adopted point-to-point routing to be the main routing paradigm used in our simulator. We implemented both physical geographic routing and logical routing. In geographic routing, each node is addressed by a relative address. Due to the dependence of many of our simulated schemes on GPSR, we implemented it as the main geographic routing scheme in our simulator. Note that we implemented the basic GPSR scheme, rather its modifications (already discussed in Chapter 2). This is primarily due to the fact that DIM was implemented on top of the basic GPSR and our goal was to achieve a comparable DIM's performance as that reported in [37]. Furthermore, our experiments mainly focused on networks with uniformly distributed sensor node locations. In such a case, the majority of the readings are routed through the greedy mode (in our simulations, more than 90% of the packets were routed through the greedy mode rather than through the perimeter mode). In addition to GPSR, we implemented logical routing schemes as the Logical Stateless Routing (LSR) protocol, which is part of the KDDCS (Chapter 5). The appropriate routing scheme is used in any given simulation experiment.

3.2.2 Policies and Schemes

We implemented different DCS schemes in our simulator. We selected the Local Storage (LS) scheme to be our base case. In LS, each sensor stores its own readings and any query is flooded to all sensors. As state-of-the-art DCS schemes, we adopted the GHT and the DIM schemes. We implemented our ZS, ZP/ZPR, and ZS/ZP/ZPR schemes (presented in Chapter 4) on top of the DIM scheme. Also, we implemented our KDDCS scheme running on top of the Logical Stateless Routing (LSR) scheme (both are presented in Chapter 5).

3.2.3 Experimental Setup

Now that we have presented a high-level overview on our simulator, we move on to present the details of our experiments. Table 1 shows the parameter values and/or ranges used in our experimental setup. In selecting our experimental parameter values, we tried to be as close as possible to the DIM's original simulation environment [37] to guarantee a similar performance for the DIM scheme. We also considered ranges of values for each parameter rather than a single value to test the effect on each of the parameters on the performance of our schemes. As for the network size, the network radius, and the communication range, our goal was to always maintain a dense network where each sensor has in average 20 nodes within its nominal radio range. This guarantees the application of the greedy routing module of the GPSR algorithm for the majority of the time. As a simplification, we assumed that a reading size is equal to the packet size. We also assumed that the energy unit is the amount of energy needed to send or receive one packet (thus, one reading). As for the node storage capacity, we tested values ranging from 20 to 100 readings. To understand the meaning of these values in real-world scenarios, let us assume a packet size of 2 bytes (which is typically considered as a small packet size [30]). This means a reading size of 2 bytes and a node storage capacity ranging from 400 bytes to 2KB.

We tested our schemes with different parameter combinations and achieved similar performances in most of the cases. For all experimental results presented in this thesis, we set the node storage and energy capacity to be 30 readings and 70 units, respectively (with the goal of setting a common reference environment for comparing all schemes).

3.2.4 Storage Workload

We now describe our general procedure to form our storage loads. In a real-life sensor network implementing a DCS storage scheme, the storage load is composed of sensor readings sensed by the different sensor nodes. Once each sensor node senses a reading, the sensor forwards this reading, according to the underlying DCS index structure, to the sensor node responsible for storing it. In generating our storage load, we follow the exact same procedure. Each sensor node continuously generates readings and sends each reading to its storage sensor.

Table 1: Experimental Setup

Parameter	Value Range
Network size	50 to 500 sensors
Network radius	40m to 200m
Communication range	2m
Topology	uniformly distributed sensor node locations
Node storage	20 to 100 readings
Packet size	1 reading
Energy unit	send (or receive) 1 reading
Initial node energy	50 to 200 units
Reference DCS schemes	LS, GHT, DIM
Storage workload size	10 * network size
Query workload size	10 * network size
Frequency of reading generation	1 reading per simulation time unit
Attribute Ranges	Normalized to [0, 1]

The frequency of reading generation and/or the maximum number of readings per node for each simulation are parameters to be set in our simulator. This storage load generation technique can be used to either generate uniform storage distributions or skewed ones as will be discussed below. In our simulations, we tested different values for the number of readings generated by each sensor, ranging from $[50\%, 90\%]$ of the node storage capacity.

The reading value ranges represent an important factor that determines the degree of data skewness. As we model sensors sensing multiple phenomena, a reading is a tuple of two or more attributes. For each of the attributes, we set a network wide attribute range, e.g., the attribute range of temperature can be $[30, 120]$ degree Fahrenheit. We use ranges reflecting possible real-world values for each of our attributes. Forming readings can be done in several ways in our simulator. An actual list of reading values can be manually passed to our simulator. In such a case, when any sensor is about to generate a new reading, it picks a reading tuple from the list. Another way to generate readings is that each sensor can randomly generate the reading by following a given distribution.

Based on this storage generation technique, imposing a uniform storage load on the sensor network should be straightforward. We define uniform storage load to be a one where each sensor node receives an amount of sensor readings that is, with high probability, less than its storage capacity. Assuming that the sensor locations are drawn, uniformly at random, from a uniform distribution, generating a uniform storage load can be done by using a uniform distribution for the reading values. Thus, when generating any reading, each sensor draws the reading values at random from a uniform distribution. This technique should uniformly partition storage responsibility across the different sensor nodes in the network.

3.2.5 Query Workload

Concerning the query loads, mobile users are the main sources of ad-hoc queries which can be submitted to any sensor. In our simulators, sensors issue queries. A sensor s issuing a query q means that s received q from a mobile user U, issues it to the sensor network, receives its result, and relays this result back to U. An ad-hoc query can be of the following form:

```
select NodeID, timestamp
from sensors
where temperature > t1 and temperature < t2
and pressure > p1 and pressure < p2
```

To model such a query in our simulator, a querying sensor would either select the ranges from a given list or generate them based on a given distribution. The total number of queries is a parameter to be set in our simulator. To generate a uniform query load in our network, the querying sensor is picked uniformly at random from all the sensor nodes (once a query is to be sent to the network). Then, the starting points of the ranges $[t1, t2]$ and $[p1, p2]$ are randomly picked from the temperature and pressure attribute ranges, respectively. As for the size of the attribute range targeted by each query, it is a parameter for our simulator. In our simulation, we generally generate multiple groups of queries, each with a given attribute range size, such as 10% or 20% of the attribute ranges.

3.2.6 Hotspot Generation

To study the performance of our schemes, we run the above simulator against different types of hotspot settings. Specifically, we synthetically generate storage and query loads to carefully form different hotspot scenarios. When designing our simulations, we model real-world hotspot scenarios possibly arising in our disaster management applications as much as possible. We mainly concentrate on three hotspot types, namely *storage hotspots*, *query hotspots*, and *mixed hotspots* (consisting of simultaneous storage and query hotspots). For each hotspot type, we study a variety of settings, including *single hotspots*, *multiple hotspots*, and *moving hotspots*. Recall that a single hotspot is a hotspot occurring in one sub-range of the possible attribute ranges. To understand these hotspot types and setting in light of our disaster scenario, let us assume a typical scenario for a fire that is about to develop in the disaster area. In such a case, a single storage hotspot can arise when a fire starts to take place in the disaster area and most of the temperature readings fall in the $[500, 600]$ sub-range. Similarly, a single query hotspot may occur when many responders simultaneously check for the existence of a fire, thus target the $[500, 600]$ temperature sub-range. When both hotspot

types occur simultaneously, i.e., the fire takes place and a lot of responders are checking for it, this forms a mixed hotspot in the $[500, 600]$ temperature sub-range. Multiple hotspots simultaneously occur in two or more different sub-ranges of the attribute ranges. An example of such a scenario is the existence of two fires in the disaster area, an already developed fire and another one in the process of being developed. This can create simultaneous storage (or mixed hotspots) in two different temperature sub-ranges, e.g. $[400, 450]$ and $[550, 600]$. Moving hotspots are hotspots initially occurring in a given sub-range of the attribute ranges then moving to one or more neighboring sub-ranges (one at a time) as time progresses. An example of such a hotspot is a fire throughout its different development stages. The hotspot may start at $[300, 350]$, then move on to $[350, 400]$, then to $[400, 450]$, etc. For each hotspot setting, we simulated different hotspot sizes. This corresponds to the different possible sizes of fires that can possibly arise in our network. We discuss the technique to generate storage and query loads for the different hotspot types/settings in the following paragraphs.

Using the storage load generation technique discussed in Section 3.2.4, forming storage hotspots can be either done in a controlled manner or through a skewed reading distribution. In the first case, we determine the exact reading ranges of the hotspot. This subsequently determines the sensors falling in the hotspot, which are the sensors responsible for storing the pre-specified hot ranges according to the underlying DCS scheme. We also determine the percentage of readings (among all storage load readings) falling in the hotspot range. An example for that is to say that a hotspot will be arising in a range equal to 10% of the overall attribute range. As for temperature with attribute range $[50, 90]$, this range can be $[60, 64]$ degrees. The size of this hotspot can be 60% of all readings to be generated. This means that each sensor would randomly pick a reading falling in the hotspot range with a probability of 0.6. This controlled hotspot generation technique has many advantages. The first is that it allows us to accurately determine the effect of the hotspot including its location, sensors that it affects, its size, and its duration. This gives us the opportunity to test different hotspot settings which has the great effect of carefully studying the behavior of our schemes against each of these settings. A second advantage for this hotspot generation scheme is that it allows us to generate hotspot data in a way that closely simulates real-world scenarios. Note that the actual percentage of readings falling in the hotspot range can be

slightly larger than the predetermined hotspot size in case readings supposedly not falling within the hotspot range (i.e., with probability 0.4 in our example) are generated through a distribution over the whole attribute range, e.g., a uniform distribution.

A second technique of hotspot generation in our simulator is by adopting a given skewed distribution for the readings, e.g., a normal distribution for readings over each of the attribute value ranges (i.e., a multivariate Gaussian distribution). In this case, the generating sensor would be picking the reading values uniformly at random from all the attribute ranges. It is important to note that both techniques can considered almost identical. This is simply because it is usually possible to fit any data distribution generated from the first technique with a specific skewed distribution generated from the second one. In our simulations, we mainly use the controlled technique to generate storage hotspots.

The techniques for forming query hotspots are very similar to the storage hotspot formation techniques. Traffic can be generated either based on a given skewed distribution or through carefully selected distribution (i.e., a controlled manner). In the first case, the ranges of the queries arising in any sensor follow a specific skewed distribution over the attribute ranges. In the second case, queries originate at random sensors and ask for data falling in a given attribute value range with a large probability, otherwise, ask for data falling in any range picked uniformly at random from the attribute value ranges. Both techniques are valid ones for generating query hotspots in the sensor network. In our simulations, we mainly use the controlled technique for generating query hotspots.

The third type of hotspots that we are interested to model are mixed hotspots. A network facing a mixed hotspot actually faces both storage and query hotspots occurring simultaneously. We can classify mixed hotspots into two types: *correlated* and *uncorrelated*. We define mixed correlated hotspots to be those where storage and query hotspots coincide, i.e., fall in the same attribute ranges. In contrast, uncorrelated mixed hotspots are those where the storage and query hotspots fall in different attribute ranges. We implement both of them in our simulator.

When modeling hotspots, we will be interested in both *static* hotspots and *dynamic* hotspots. Static hotspots are those that occur in a given location at some point in time without moving to another location. Dealing with the hotspot once is enough for such a

case. We will be interested in modeling *single* and *multiple* static hotspots arising in the sensor network (simultaneously or back-to-back). On the other hand, *moving* hotspots can be represented by a series of two or more hotspots that are logically dependent and that occur one after the other. The dependence can be in terms of occurring in neighboring range values, such as a 10% hotspot that starts at the $[60, 64]$ temperature range, then moves to $[64, 68]$ temperature range. This example represents a very realistic scenario that can happen in a network mounted in any open-air location as the temperature increases and decreases equivalently in the whole service area. Thus, the sensor readings tend to fall in the same value range during any given time period. This range may be slightly shifting across time, which forms a moving hotspot. It is important to test the behavior of our schemes against all these hotspot scenarios for completeness.

It is important to note that the main goal of our experimental evaluation is to study the QoD that any scheme achieves against different hotspots. Recall that hotspots can have a secondary effect on the QoS by increasing the energy consumption of the nodes falling in the hotspot area and eventually increasing node deaths. As this is considered a long-term effect for hotspots, we did not concentrate on building test cases that simulate such a case. All the hotspot scales that we generate are not intense to the extent of generate a traffic skewness that actually increases node deaths. Thus, we test the performance of our schemes in terms of the QoD improvement that they achieve and the energy consumption overhead that they impose as compared to the case where no load balancing mechanisms are implemented.

3.2.7 Experimental Evaluation Scenario

The experimental evaluation of our proposed schemes will be conducted as follows. We first compare our schemes, both local and global, against uniform loads to test their performance in the regular cases and study the overhead of implementing each of them. Then, we compare the performance of our schemes, against storage, query, and mixed hotspots. For each of the hotspot types, we test the following settings.

1. *Single Static Hotspots:* We will test our schemes against storage and query hotspots, respectively. The high level scenario of these experiments will consist of an initialization

phase of reading insertions where a skewed followed by a querying phase where a query load is imposed on the network. The aim of this set of experiments is to test the basic functionality of our schemes and their ability to individually cope with single storage, query, or mixed hotspots.

2. *Multiple Static Hotspots:* This experiment will consist of an initialization phase of reading insertions followed by a querying phase in a way as to form two or more hotspots of the same type. This experiment aims at studying the performance of our schemes against hotspots simultaneously arising in the sensor network.
3. *Moving Hotspots:* This experiment will consist of multiple pairs of insertion and querying phases. A hotspot is initially formed in a given attribute range, then, it moves to a neighboring attribute range. This moving process is repeated for several times. This experiment studies the performance of our schemes against hotspots that keep sliding across the different locations of the network. It also checks the ability of any scheme to deal with the fading of a given hotspot and the creation of a hotspot in another location.

For each hotspot type, we study the effect of changing the hotspot size, in terms of the attribute range of the hotspot, on the performance of our schemes. To achieve statistically significant results, we run each of our experiments for multiple runs (5 to 10) and average the results over all the runs. We were aware of the standard deviation in all simulation runs and we did not encounter a relatively large variance in any of the simulations.

3.2.8 Experimental Metrics

For each of the above hotspot types, we compare our schemes based on the metrics presented below.

1. The Number of Dropped Events (Readings/Queries): measures the number of readings (or queries) dropped either due to storage overflow and/or collisions. This metric is important as it measures the Quality of Data (QoD) of any scheme against the different storage/query loads. In the case of storage hotspots, events are mainly readings dropped due to storage overflow. Wireless collisions represent a secondary source of dropping readings. Thus, this metric measures the data persistence achieved by any scheme against

different storage loads. In the case of query hotspots, it measures the number of dropped queries mainly due to collisions. The metric then can be viewed as the number of unanswered queries by the sensor network (Note that we do not retry queries once they are dropped). This is also considered as an important QoD measure at it gives an indication about the ability of the sensor network to answer the different queries and thus the QoD achieved by the sensor network.

2. The Average Results of an x% Query: measures the average number of readings returned for queries of a given attribute range size, e.g., the average number of readings returned for queries asking for 10% of the attribute range. This metric gives an indication about the QoD of the sensor network and its ability to successfully answer queries of different sizes.

3. The Average Storage Load: measures the QoD achieved by any scheme by measuring its ability to improve the average storage load across sensors. It also measures how well any scheme load balances the hotspot data across the sensor network rather than assigning it to sensor nodes falling in the hotspot area.

4. The Number of Full Nodes: measures the number of nodes whose storage reaches the maximum storage capacity. This is also another important load balancing metric. As a hotspot causes the storage overflow of sensor nodes falling in the hotspot area, the reduction of the number of full nodes gives an indication about increasing the number of nodes responsible for storing the hotspot data. Eventually increasing the number of full nodes, after this reduction, gives an indication about the extreme load balancing ability of any scheme by load balancing the hotspot data early enough so that the larger set of nodes responsible for storing the hotspot data reach their full capacity.

5. The Average Energy Consumption: compares the QoS of the different schemes by measuring the energy consumption overhead they impose on the sensor network.

6. The Number of Dead Nodes: is another important QoS metric as it indicates the coverage QoS of the sensor network. Additionally, it has an important implication on the load balancing performance of the schemes under concern.

3.3 SUMMARY

In this chapter, we presented our system model and the details our experimental platform. One shortcoming of our experimental evaluation is that it was not based on the standard sensor network simulators, e.g., TOSSIM [35]. The main reason for this fact is that the currently available sensor network simulators were not fully developed by the time we pursued this research. We intend to port our implementations to one or more of these simulators in the near future. Another important note is that, although our data sets are considered to be synthetic, we tried to make their distributions and value ranges as realistic as possible to be similar to real-world data sets that could be recorded from any already-deployed disaster management sensor network. Unfortunately, we were not able to find any real-world data set for a disaster management sensor network with sensors deployed in a service area (of any type, urban, open-air, etc) and queries are issued by any type of mobile devices. Experimentally evaluating our schemes on real sensor network testbeds will be left as one of our future extensions.

4.0 LOCAL HOTSPOT DETECTION AND DECOMPOSITION

Current DCS schemes fail to effectively cope with both storage and query hotspots. In this chapter, we propose two local solutions to detect and decompose storage and query hotspots, namely the *Zone Sharing (ZS)* and the *Zone Partitioning/Zone Partial Replication (ZP/ZPR)* schemes, respectively, . We combine the two schemes to form the ZS/ZP/ZPR scheme to detect and decompose mixed hotspots. Our schemes work on top of the DIM DCS scheme [37]. We present our schemes in the following sections.

4.1 LOCAL DETECTION AND DECOMPOSITION OF STORAGE HOTSPOTS

In this section, we propose a solution to the storage hotspots problem arising mainly due to *irregular data distribution* in DCS schemes. We will present our solution in the context of the DIM scheme, which has been shown to exhibit a performance superior to other DCS techniques, e.g. GHT, for the case of multi-dimensional range queries. In DIM, a sensor node is defined as a leaf of a k-d tree [11] and is assigned a binary address (*zone*) based on its geographic location. The address size (in bits) of each sensor is equal to its depth (level number) in the tree. Readings are mapped to binary codes based on their attribute values and they are routed to their storage sensors using the Greedy Perimeter Stateless Routing (GPSR) algorithm [32].

The intuitive idea of our proposed *Zone Sharing (ZS)* solution is that a node experiencing high load compared to its neighbors is a good indication of a storage hotspot and it is reasonable to share its load with one of its less-loaded neighbors. In the specific terms of

DIM, a high-loaded sensor node attempts to split its owned zone with one of its less-loaded neighbors. Based on the maximum number of times a zone could be shared, we present two ZS flavors: *Single-Hop Zone Sharing (SHZS)* and *Multi-Hop Zone Sharing (MHZS)*.

Experimental results show that the main advantages of ZS are:

- Improving *QoD* by distributing the hotspot readings among a larger number of sensors. Improvements ranged from 50% over DIM for single storage hotspots to 20% over DIM for multiple storage hotspots.
- Increasing the *energy savings* by balancing energy consumption among sensor nodes. Energy consumption overhead additionally imposed by ZS ranged from 5% (per node) over DIM for single hotspots to 3% for multiple hotspots.

This was valid for hotspots of sizes ranging from 40% to 80%.

In the next subsections, we describe and analyze the ZS scheme.

4.1.1 Basic Idea

Figure 4 shows a typical scenario for the zone sharing process. In the k-d tree on the LHS, N0 (address = 0) is experiencing high storage load compared to its neighbors, N1 (address = 11) and N2 (address = 10). N2 has a smaller storage load than N1. The difference of load between the two subtrees is considered as a potential hotspot indication in the left subtree. Therefore, in order to cope with this hotspot that is about to be formed in N0, node N2 passes the responsibility of its original zone to N1 and takes responsibility of a portion (around half) of N0's zone. The k-d tree takes the form presented on the RHS of Figure 4.

Note that the move of N2 is only *logical*, i.e., N2 still keeps in its original geographic location (assuming nodes are stable), but, it takes responsibility of another zone whose code value is different from its binary address value. Also, it is assumed that each node has enough energy to send and receive events during this logical move.

The above procedure is used to decompose storage hotspots as follows: In case a hotspot arises in a set of sensor nodes, the *border nodes*, i.e., those nodes falling on the border of the hotspot, will trade (pass) responsibility of portions of their zones to some of their less-loaded

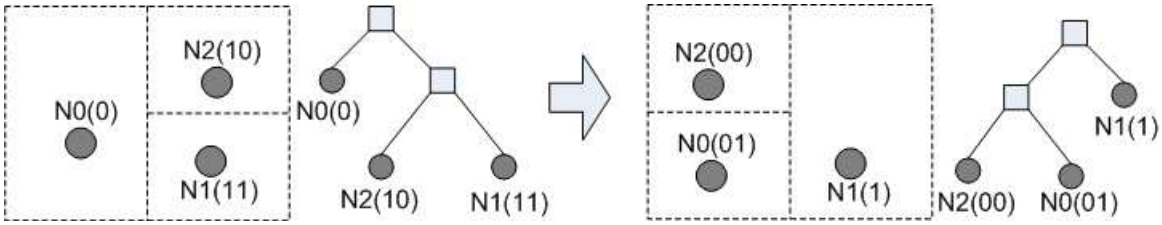


Figure 4: Zone Sharing Illustrative Example

neighbors, not falling in the hotspot area, as described above. This will lead to decreasing the size of the hotspot by removing these border nodes outside from the hotspot. Then, the nodes that were previously on the hotspot borders will now act as the less-loaded neighbors for the new border nodes, and consequently, receive part of the load of these border nodes. This process continues till it totally decomposes the hotspot and distributes its load on a larger number of sensor nodes.

But many questions need to be answered to efficiently implement zone sharing, such as: When can a node tell that it is falling in a hotspot? How does a node know the neighbors information? With which neighbor to trade the excess capacity? How much load to trade? How many times a given zone can be traded? What changes need to be made to GPSR to account for the zone migration? What will be the implementation overhead of this scheme? In the following subsections, we answer these questions in turn.

4.1.2 Distributed Migration Criterion (DMC)

A node experiencing an unusual load can split its owned zone with one of its neighbors. In such a case, the node's depth in the k-d tree is increased by 1 (i.e., its address size is increased by 1 bit). A neighbor of this node is logically moved to share its zone. As this neighbor was another leaf in the tree, it passes its original zone to one of its siblings, and the tree depth of this last decreases by 1 bit.

The above process can be described as a migration of a given node from its original level to a lower one in the k-d tree. Thus, we call the original node that splits its zone **the donor**

(N0 in Figure 4) as it donates almost half of its load, while its neighbor that takes part of that zone is called **the migrator** (N2 in Figure 4) as it migrates from a zone to another. The node that receives the original migrator's zone is called **the receiver** (N1 in Figure 4).

We define l_x to be the storage load of node x and e_x to be the energy level of node x. Therefore, in Figure 4, the original load of the migrator, which is passed to the receiver, is referred to as $l_{migrator}$, while the total load of the donor, before the migration process, is l_{donor}. The amount of load that the donor passes to the migrator after the migration of this last is defined as the *traded zone*, T. It is expressed in terms of the number of traded messages (Note that an event represents one message). The value of T is solely defined by the donor, based on the distribution of its storage load, in a way that balances the storage between itself and the migrator after the migration process. In case the load is uniformly distributed in the hotspot zone, T can be considered as half of the original load of the donor, l_{donor}.

In order to define which neighbor to split zone with, we need to relate the loads, as well as the energy levels, of the three nodes involved in the migration process in a reasonable way that maximizes the profit gained from migration. Therefore, we provide a set of inequalities to be *locally* applied by the three nodes. These equations represent *necessary* conditions that must be fulfilled to justify zone sharing. In these equations, an energy unit represents the amount needed to send one message. The fraction r_e is the amount of energy consumed in receiving one message (if less than one energy unit):

$$\frac{l_{donor}}{l_{migrator} + l_{receiver}} \geq C_1 \tag{4.1}$$

$$\frac{T}{l_{migrator}} \geq C_2 \tag{4.2}$$

$$\frac{T}{e_{donor}} \leq E_1 \tag{4.3}$$

$$\frac{l_{migrator} + r_e * T}{e_{migrator}} \leq E_2 \tag{4.4}$$

$$\frac{l_{migrator} * r_e}{e_{receiver}} \leq E_3 \tag{4.5}$$

where C_i's and E_j's are constants representing storage ratio and energy ratio thresholds, respectively.

The first two equations are concerned with relating the storage loads of the three nodes. Equation (1) states that the pre-migration load of the donor should be much bigger than the post-migration load of the receiver. Such constraint is needed in order to guarantee that no migration oscillation would occur. It should be applied by both the donor and the receiver. The value of the constant C_1 should be large enough in order to make sure that the donor is really falling within a storage hotspot. Equation (2) states that the post-migration load of the migrator is bigger than its pre-migration load. This is needed in order to gain some profit from the whole migration process. This equation should be applied by the migrator. It is obvious that C_2 should take large enough values (e.g., greater than or equal to 2) in order to avoid cyclic migrations.

Equations (3) to (5) relate the energy levels of the three nodes, before and after the migration process. Equation (3) states that the energy consumed in transferring the traded zone is much less than the donor's energy level prior to the migration process. It is applied by the donor. Equation (4) states that the energy consumed in sending the migrated zone, as well as receiving the shared zone, is less than the total energy amount owned by the migrator pre-migration. It is only applied by the migrator. The last equation, Equation (5), states that the energy consumed by the receiver in receiving the original load of the migrator is less than its available energy before migration. It is obvious that this last is applied by the receiver. These inequalities are needed to make sure that the energy amount consumed in the migration process will not cause the death, or the approach to death, of one or more of the nodes involved in the migration process. The values of the E_j thresholds should be small (e.g., less than 0.5).

It is clear that the values of different thresholds can differ based on the importance of energy as opposed to that of data in the sensor network. Recall that sensor network applications can be classified into *search/discovery* applications and *investigation* applications. In search/discovery applications, the sensor network is expected to run for a long period of time, which indicates the high value of energy in the network. Additionally, the application consists of monitoring the environment and using the sensor network data in offline studies. Thus, no urgent action will be taken based on a subset of the sensor readings. In such an application, the energy safety requirements should be strictly applied. Thus, the E thresh-

olds should take very small values (e.g., 0.3 or less), while the C thresholds should take large values (e.g., 3 or more), to make sure zone sharing is not applied except in really needed cases. On the other hand, investigation applications, such as our disaster management application, the sensor network data is highly urgent and is mostly used to trigger important actions, such as decisions taken during the disaster management process. In such a scenario, the C thresholds should take small values (that can be as low 1.5 or even lower), while the E thresholds should take large values (e.g., 0.6 or more), to facilitate the satisfaction of the migration process requirements. In general, the actual values of the E and C thresholds depend on the specific needs of the sensor network application.

One benefit of the DMC is that it has the ability of using the available sensor storage and energy capacities with no dependence on the capacity or the energy distributions in the network. As each node shares its storage and energy capacity once engaging in the zone sharing process, the ZS scheme can be efficiently applied in a sensor network with homogeneous or heterogeneous storage and/or energy capacities.

4.1.3 DMC Implementation Details

To be able to localize the evaluation of the above equations, each node is supposed to maintain load information, in terms of in terms of storage and energy, of its neighbors. This can easily be done by taking benefit from the list of direct neighbors that each node maintains in the DIM scheme and keeping track of the storage and the energy of each of these neighbors. Using this list, the node will be able to select the best candidate neighbor to split its owned zone with in case of falling in a storage hotspot. The periodic messages exchanged between neighbors to maintain the underlying DCS index structure of DIM, as well as insertion and query messages accessing the sensor network, can be piggybacked with such information. Each node periodically checks its storage level and applies the migration criterion in case this level exceeds a given threshold. The time window of checking for hotspots, and thus applying the migration criterion, depends on the generation rate of readings in the network and can be a network parameter that the network designer can carefully set based on the probability of a hotspot formation and the expected time the hotspot may take to start

```
ZS (Current Node C, List of Neighbors N, Threshold t)
Begin
  1. If (storage load (C) / storage load (n) > t) for all nodes in N
  2.       Sort list N ascendingly based on storage load
  3.       For every node n in N
  4.           Send RTM to n (containing storage load (C), energy (C), and size (T))
  5.           If (ATM received from n (this means n successfully applied DMC))
  6.                       m = n
  7.                       Break
  8.           End
  9.       Send traded zone T to m
 10. End
End
```

Figure 5: Zone Sharing Algorithm

causing problems in the sensor network.

To fully localize the evaluation of the DMC equations without the dependance on a central authority in the network, the migration process, as described above, needs some distributed decision making among sensors involved in migration process. For this purpose, a *three-way hand-shaking* procedure must be applied in case a node decides to share its zone with one of its neighbors. First, the donor should decide which neighbor to be the migrator and contact this last with a *Request to Migrate* message (RTM). In case the candidate migrator is able to migrate, it should select an appropriate receiver, inform it about its migration decision, and invites it to take its storage responsibility. In case that receiver accepts this invitation, the migrator replies to the donor with an *Accept to Migrate* message (ATM). Figure 5 shows the ZS algorithm periodically applied by each node in the network. Note that the RTM and ATM requests could either be piggybacked on other messages or sent explicitly. The overhead of such messages is negligible compared to that of the actual migration process.

It is important to note that the case of multiple RTM messages simultaneously accessing a single node does not cause any problems in the network. In such a case, the node simply processes RTMs on a first-come-first-serve basis. Once receiving the first RTM, the node

searches for a receiver. In the meanwhile, it rejects any further RTMs that it receives. As the ZS algorithm presented in Figure 5 shows, a donor searches for migrators by considering its neighboring nodes, one by one. Thus, in case a donor receives a rejection for an RTM, it moves on to consider the next prospective migrator from its list of neighbors.

4.1.4 Single Hop Zone Sharing (SHZS)

Now that we defined the DMC, we need to decide how will the hotspot decomposition process take place. This can be done by determining the number of times a single zone can be traded. In order to minimize the changes made to the tree, we limit the maximum depth change of any zone to 1. In other words, each zone can be shared only *once*. A receiver cannot share the original migrator's zone another time. Hence, we call this ZS version *Single-Hop Zone Sharing (SHZS)*.

As every zone will be at most one hop further from its original location, the original GPSR can be used to query the different zones and the original donor node will just forward the query, or the insertion, to the migrator in case the queried zone has been already donated. This will enable us to use the same DIM scheme with minimal changes made for the hotspot decomposition purpose.

Although the SHZS algorithm is quite simple, it has *three* drawbacks when dealing with large hotspots, i.e. those hotspots spanning more than one or two nodes. First, with large storage hotspots sizes, the neighbors of the overwhelmed node will most probably be falling in, or close to, the hotspot, thus suffering from the same symptom. Therefore, these node will hardly satisfy the DMC. As the hotspot size increases, border nodes will be extremely loaded as well as their neighbors, thus complicating the DMC satisfaction. Furthermore, when a border node passes a portion of its load to a neighbor, it will be still falling in the hotspot, thus, unable to receive any load from other nodes closer to the center of the hotspot. Hence, the DMC will not lead to the hotspot decomposition.

Second, the fact that a shared zone cannot be shared again complicates the scheme in the case of dealing with large hotspots. At some point, all the neighbors of the hotspot node will be responsible for zones that were already falling under this node's responsibility. Then,

this last receives further readings by dropping old ones, leading to decreasing the QoD.

A third complication is regarding the GPSR algorithm in the case of reading insertion. Assuming that a donor selects to share the zone that has more upcoming readings to be inserted in the future, all such insertions will first pass by the donor before going to the migrator. This will impose a large energy consumption burden on the donor node.

However, it should be noted that the situation changes when the storage hotspot spans a fairly small number of nodes. In fact, the important criteria about such small hotspots are that the number of nodes directly neighboring the hotspot nodes is equal or greater to the hotspot nodes and that the storage load imposed on those nodes is negligible. In such a case, the hotspot decomposition can start from the borders of the hotspot area and going inwards towards the center of the hotspot. This process would proceed in rounds. In each round, SHZS is applied to the nodes on the border of the hotspot in order to share their storage with their less-loaded neighbors. After each step, the size of the hotspot, in terms of the number of nodes falling into it, decreases. The process continues until reaching the center of the hotspot and completely decomposing it. In order for SHZS to work well in this scenario, the total number of rounds needed should be fairly small and the nodes falling around the hotspot borders, throughout the process, should be ready to receive part of the hotspot load. Unless these two conditions are satisfied, SHZS will fail to completely decompose the hotspot.

From the above points, it is obvious that the SHZS algorithm is best suited to decompose small hotspots, in terms of both the number of nodes falling in the hotspot and the storage load imposed on the hotspot nodes. The ability of the scheme to decompose larger hotspots becomes fairly limited due to the simplicity of the hotspot decomposition technique used. In order to extend the zone sharing idea to handle larger hotspots, we present the Multi-Hop Zone Sharing (MHZS) scheme in the next subsection.

4.1.5 Multi-Hop Zone Sharing (MHZS)

The main idea of the MHZS scheme is to relax the strict single hop sharing assumption adopted in the SHZS scheme. Based on this relaxation, we introduce another ZS version

where a zone can be shared more than once. This implies that a given zone can encounter a tree depth change, as well as a tree path change, of any size. To see how this can occur, consider a network with two main subtrees, $0x$ and $1y$, where x and y are zone bit-codes of different lengths. In this topology, a hotspot may arise in node $N0$ with address 0110, i.e., falling in the $0x$ subtree. To decompose this hotspot, MHZS applies the zone sharing process for 3 hops. Thus, at the end of the zone sharing process, the hotspot zone will be in a node which is 3 hops away from its original location. Among these hops, the first one is applied with node 0111 from the $0x$ subtree. However, the following hops are applied with nodes 100 and 101, respectively. Thus, the final location of the hotspot zone is node 101, which belongs has a different path and different path length in the k-d tree than the original node 0110.

4.1.5.1 GPSR Modifications This above description of the MHZS scheme introduces an important question, which is: *how would the routing work in the resulting k-d tree?* In the original k-d tree of the DIM scheme, routing worked in a systematic hop-by-hop purely localized manner without any need for keeping immediate information of the node actually responsible for storing each zone as the convergence to that node at the end of the routing process was guaranteed. However, due to the changes introduced by MHZS to the k-d tree structure, this is not the case anymore. As a zone can be moved several hops away from its original node, using the basic GPSR routing algorithm without modifying it will involve the original donor, as well as subsequent donors, in all insertions and queries concerning the shared zone. This would be an extreme overhead as it would involve the original hotspot nodes in all these operations. All insertions and queries will first go to the original hotspot node. Then, they would be forwarded to the subsequent donors, one after the other, until reaching the final node actually storing the zone. This overhead would be proportional to the number of times the zone sharing process is applied. Furthermore, this would defeat one of the purposes of applying the zone sharing process, which is reducing the energy consumption burden imposed on the hotspot nodes. Hence, GPSR must be augmented by some means to determine that a zone has been shared and moved away from its original location. We discuss the modifications that we introduce to GPSR in the following paragraphs.

```
GPSR_ZS (Current Node C, Shared Zone List SZL, Destination D, dest_changed flag)
Begin
        1. If (dest_changed == false)
        2.      If (D is found in SZL as original donor of SZL entry e)
        3.              dest_changed = true
        4.              D = final migrator (e)
        5.      End
        6. End
        7. Apply basic GPSR algorithm to D
End
```

Figure 6: Modified GPSR Algorithm for ZS

To implement the MHZS scheme, we assume that each node maintains a *Shared Zones List (SZL)* containing three entries: zone address, original donor, and final migrator. Upon zone sharing, the donor sends the shared zone address, its name, and the migrator's name, to all its neighbors. Thus, each node will be aware of zones traded by its neighbors.

In case of multiple sharing of the same zone, the old migrator becomes the new donor. It sends the zone address, the original donor, and the new migrator, to all of its neighbors. Thus, it has to check its SZL first and send the entry corresponding to the zone under concern after updating the final migrator entry with the new migrator. In case a node receives a shared zone entry that is already present in its SZL, it updates its list with the new entry. This means that a given zone has been re-shared. This update guarantees that a shared zone will have similar entries in all shared lists containing information about this shared zone. The node then forwards the shared zone entry to its neighbors.

When routing an event (an insertion or a query), GPSR first searches the SZL with the zone address. In case an entry for such zone is found, this means that the original destination of the zone has been changed to a new one. Therefore, the destination node found in the SZL entry of the zone is used to explicitly update the destination field of the message to be routed to its new value. GPSR then uses the new destination address to forward the message using the best path to the final migrator. A flag is updated in the message to indicate that its original destination has been already changed to its current one to avoid further lookups

in subsequent nodes. Figure 6 shows the modified version of the GPSR algorithm run by each node in the network.

As all the neighbors of each donor and migrator are forwarded with the new SZL entry of the shared zone, this results in a *cloud* of nodes with the same SZL entry. This cloud includes the original hotspot area as well as the sensor nodes surrounding it. Applying the SZL search at each of the routing steps results in changing the routing direction of any event as soon as an SZL entry for the event zone is encountered. This usually happens when the event starts entering the cloud. Thus, the cloud deviates the direction of any event to direct it to the new destination of the required zone. This has the effect of considerably reducing the energy consumption overhead imposed on the nodes falling in the hotspot area.

It is important to mention that queries are routed exactly as they used to be routed in the original DIM scheme. Any query initiating at any of the sensor nodes, and asking for a given range over the attributes stored in the network, is processed in a hop-by-hop manner. At each step, the query is forwarded to the next node towards its final destination in case the query can be completely answered by one node, i.e., in case the attribute range of the query is falling under responsibility of one sensor node. Otherwise, the query is split at one point into two or more queries each targeting a different node. Applying our GPSR modification results in the possibility of splitting the query at any time in case a subset of the queried zone has already been shared. On the other hand, the query is fully deviated in case the whole queried zone has been shared. Otherwise, the query processing proceeds exactly as dictated by the original DIM scheme.

4.1.5.2 Hotspot Decomposition Mechanism

After describing the routing process of any insertion or query in MHZS, we now illustrate the high-level hotspot decomposition technique that applying the above MHZS would result in. Let us consider a storage hotspot arising in a group of sensor nodes. Applying MHZS results in multiple-rounds of hotspot decomposition (with each round similar to the one we described for SHZS in Section 4.1.4). The hotspot decomposition starts from the hotspot borders and goes on towards the center of the hotspot area. However, the main difference between MHZS and SHZS is that the former will be able to scale more than the latter. In both schemes, applying the zone sharing

for the first time would result in increasing the number of nodes responsible for storing the hotspot load by involving the immediate neighbors of the border nodes in this responsibility. However, for the next round, MHZS tends to further send the zones shared in the first round away from their original locations. This results in more room in the border nodes and their surrounding nodes. This room would be use in further dissipations of the hotspot load away from the hotspot area. This has the effect of increasing the MHZS ability in decomposing large hotspots, unlike the case with SHZS.

4.1.6 Handling Dynamic Hotspots Through Zone Rejoining

The above description of the ZS scheme had an underlying assumption that a the load distribution does not change over time. In other words, hotspots were assumed to be static. Once a hotspot arises in some attribute range, it continues to be there for the rest of the network operation time. It is clear that this does not seem to be realistic. For example, a storage hotspot may arise in a given attribute range, e.g., $[x_1, y_1]$, at some point in time t_1. After some time period, the hotspot may vanish and the load may return to be normal, i.e., following a non-skewed distribution, over the whole possible range of attributes. Another possibility is that another hotspot may arise in another attribute range $[x_2, y_2]$ in a later point in time $t_2 > t_1$. To handle these scenarios, our ZS scheme needs to implement a *zone rejoining* functionality that is responsible for stopping the zone sharing process and retrieving the k-d tree structure dictated by the original DIM scheme once a storage hotspot dissipates. We describe this zone rejoining functionality in the following paragraph.

Our zone rejoining process relies on continuously monitoring the storage load of the hot zone and periodically comparing the change rate of that zone with the storage load change rates of its neighboring zones. This can be implemented as follows. Once the zone sharing process is applied for one or more times, the donor(s) and the migrator(s) of each shared zone keep track of the initial load of the shared and migrated zones, respectively. The initial loads of the shared (or migrated) zones denote the loads of these zones immediately after the zone sharing process. Additionally, the two nodes keep track of the rate of load change of these zones. Periodically, these rates are compared with the load change rates of the zones stored

by the neighbors of these two nodes. In case the change rates are similar, this indicates a strong possibility that the initially hot zone (composed of the union of both the shared and the migrated zones) became less overloaded. Subsequently, this shows the success of the zone sharing process in decomposing the hotspot. In case the change rates of the shared and/or migrated zones are much higher, this indicates that another zone sharing step may need to take place to fully decompose the original hotspot.

However, in case the change rates of the shared and migrated zones are much less than those of the neighboring zones, this can be an indication that the hot zone is no more popular. In such a case, the zone sharing process may be rolled back and the migrated zone may be returned to its original storage node (or to its previous donor, in case this zone has been shared more than once). This *rollback process* is performed in two steps. The first step is to send the current readings belonging to the migrated zone to the donor. This storage load transfer is accompanied by a notification from the migrator to the donor that the latter has resumed the storage responsibility of the migrated zone. The second step consists of notifying all the neighbors of the migrator and the donor, as well as their neighbors, to update their SZL entries of this zone and indicate that the previous donor became the final destination of the zone. It is important to mention that the first step may be omitted in case one of the two nodes (or the two of them) is (are) low in energy or in case the migrated zone readings are relatively old. This can be a design parameter for the zone rejoining process.

To conclude, ZS copes with storage hotspots of different sizes by decomposing the hotspot storage load across a larger number of sensor nodes. The decomposition process takes place from the hotspot borders and going all the way towards the hotspot center.

4.1.7 ZS Implementation Overhead

Now that we have fully described the ZS scheme with its two versions, we move on to discuss its implementation overhead on sensor motes. We concentrate on two types of overheads: the *processing overhead* and the *memory overhead*. We discuss each of these overheads in the following paragraphs.

We start by the processing overhead. Looking at the ZS scheme, we realize it introduces

two main types of processing overheads, the first is the DMC evaluation and the second is the SZL search. The DMC evaluation results from the application of the DMC five equations in a periodic manner. As none of these equations contains any mathematical operations other than the standard addition, subtraction and multiplication, the overhead of applying these equations once is negligible. Furthermore, as the DMC is periodically applied by each node a number of times in the order of the direct neighbors of that node, which is in the worst case $\theta(n)$. Thus, the overall overhead imposed by the DMC on each sensor node is negligible.

The second processing overhead to be considered is the SZL search overhead. As we are constraining each node to send the information about shared zones only to its neighbors, the size of a shared zone entry will be relatively small. It is easy to prove that an n-times shared zone will be at most present in the shared zone lists of nodes of $\theta(n)$ hops away from its final destination. Thus, the SZL search overhead that will be encountered by each sensor node will be at most in the order of $\theta(\log n)$. Also, for shared zones, search in shared lists occurs only once in the furthest node containing shared zone information (i.e., furthest from the final destination). Upon seeing the destination flag set in the message, GPSR in the following nodes uses such destination immediately without searching the list. Thus, this search does not add a considerable energy consumption overhead on individual sensor nodes as it results in a maximum of $\theta(n)$ energy units to be consumed by all nodes involved in routing any event.

We now move on to consider the memory overhead imposed by the ZS scheme on each sensor node. This overhead results from two sources: the DMC code and the SZL. Considering the DMC code, we realize that it mainly consists of a for loop spanning the neighbors of each node and having the five inequalities applied for each of these neighbors to decide whether a zone sharing is to take place or no. Additionally, the storage loads and energy status counters have to be kept for each of the neighbors of the node. Two additional variables representing the best migrator and the best receiver have to be maintained within the loop. The DMC implementation takes less than 30 lines of C++ code. Once run, the program requires less than 1 KBytes of main memory. Thus, the memory requirement of the DMC implementation is considered very miniature. Our experimental study showed that the SZL is small thus does not impose a considerable storage burden on the sensor nodes.

4.1.8 ZS Experimental Evaluation

In this section, we move on to study the performance of our ZS scheme when facing storage hotspots of different types and sizes. Our study is twofold. We first study the effect of the different parameters on the ZS performance. Towards this goal, Section 4.1.8.1 presents the high level overview on our sensitivity analysis results. Based on these results, we set default values for the different ZS parameters and compare the ZS performance (both SHZS and MHZS) against our reference schemes. Section 4.1.8.2, Section 4.1.8.3, and Section 4.1.8.4 compare the performance of our schemes against single, multiple, and moving storage hotspots, respectively. We then present the full results our sensitivity analysis in Sections 4.1.8.5 to 4.1.8.7. The results show the detailed effect of each of the ZS parameters on the overall performance of the scheme.

Throughout this section, the node storage capacity is equal to 30 readings and the node initial energy capacity is equal to 70 units. Therefore, a *full sensor node* is defined to be a sensor node having 30 readings in its cache. Similarly, a node is depleted (and consequently considered dead) as soon as it consumes 70 energy units. Once a node is dead, all readings stored in this node are considered lost. Based on the DIM scheme, the storage responsibility (a subset of the attribute range) of the dead node is assigned to one of its direct neighbors. Recall that we define an event to be either a reading or a query.

For each of our experiments, we study three aspects: QoD (R1), load balancing (R2), and energy consumption (R3). For the QoD, we study the number of dropped events (readings/queries) and the average node storage. We refer to the percentage of QoD improvement to be the percentage of decrease in event drops. For the load balancing, we study the number of full nodes. As for energy consumption, we study the average node energy and the number of dead nodes. The average node energy is the one that defines the improvement or the downgrading in the energy consumption performance. To be statistically significant, we conducted 5 simulation runs for each of the experiments and taken the average of values across all runs.

For each of the hotspot types, we conducted experiments on different hotspot sizes ranging from 20% to 80%. Unless otherwise stated, performing well on the large hotspot sizes,

i.e., [60%, 80%], implies a good performance on the moderate sized hotspots, i.e., [40%, 60%]. In most of the cases, the performance burden imposed to the network by small hotspots, i.e., hotspots less than 40%, does not justify the cost paid to detect and decompose the hotspots.

The main lessons that we learned from the experimental evaluation of ZS against our reference schemes can be summarized in the following points:

1. When facing single storage hotspots, MHZS achieves a QoD improvement of around 55% over DIM (as opposed to 20% improvement for SHZS). Towards this, MHZS imposes an energy consumption overhead of 5% per node over DIM (as opposed to 3% overhead for SHZS).
2. For multiple hotspots, SHZS slightly improves QoD by around 20% while MHZS performs worse than DIM in some cases due to collisions (occurring when decomposing multiple hotspots simultaneously). SHZS imposes an energy consumption overhead of around 2% over DIM.
3. For moving hotspots, MHZS scores a 20% QoD improvement (as opposed to 15% for SHZS) while introducing an energy consumption overhead of 3% over DIM (as opposed to 2% for SHZS). This comes with an increase in node deaths of around 5% of the network size.
4. For single hotspots, increasing the C value from 2 to 3 decreases the QoD of MHZS by around 20% (compared to DIM) due to the decrease in the number of times the ZS process is applied. This comes with decreasing the energy consumption overhead by around 3% (from 8% to 5%).
5. For single hotspots, decreasing E to 0.3 results in improving QoD by at least 20% for both single and multiple hotspots while reducing the energy consumption overhead by around 5% (compared to when $E = 0.5$ with DIM as the base case).
6. For single hotspots, Changing the SC value does not highly affect QoD. When $SC = 5$, energy consumption overhead is around 3% better than for higher SC values.
7. In general, SHZS is the better choice than MHZS as the former scores acceptable QoD improvements for all hotspot types while imposing small energy consumption overheads (compared to DIM).

We start by presenting the results of our sensitivity analysis study.

4.1.8.1 Sensitivity Analysis To pick the default parameter values for ZS, we studied the effect of changing the storage level threshold C, the energy level threshold E, and the maximum allowed share count SC over the ranges $[1.5, 3]$, $[0.3, 0.8]$, and $[5, 15]$, respectively. Figure 7 plots the performance of ZS (for single storage hotspots) in terms of both the QoD improvements and the QoS overheads encountered for the different parameter values. Before describing the selection process of the default values for the different ZS parameters, we first illustrate the reasons for selecting each of these ranges.

We study the effect of changing the C threshold between 1.5 and 3. The selection of this range is based on the role of C in the ZS process. Recall that the pre-migration load of donor to the collective pre-migration load of the receiver and the migrator should be more than, or at least equal to, C for the ZS to take place (Inequality 4.1). Furthermore, the size of the traded zone to the pre-migration load of the migrator should be larger than, or at least equal to, C as well (Inequality 4.2). Thus, selecting a value for C which is lager than 3 will result in highly reducing the probability of satisfying the DMC, and in a consequent high reduction in the number of zone shares taking place in the network. Similarly, setting C to a small value would highly increase the number of zone shares and may cause cyclic migrations. We study the performance for C values ranging from 1.5 to 3 with an 0.1 increment. We only present the results for a subset of these values that captures the main learned lessons from the study.

We study the effect of changing the E parameter between 0.3 and 0.8. The selection of this value range is based on the role of E in the ZS process. Recall that the energy needed to send the traded zone to the pre-migration energy of donor should not exceed E (Inequality 4.3). Thus, E should definitely be less than 1 so that the donor does not deplete all its energy. In fact, it should make sure to leave the donor with a good amount of energy. That's why we select a value of 0.8 as the maximum for this threshold as to make sure that the donor has a least a fraction of 0.2 of its energy remaining after the zone sharing process. Furthermore, The energy of needed by the migrator in the zone sharing process (which comprises the energy needed to send its pre-migration load plus the energy needed to receive the traded

zone) to the energy of the migrator should not exceed E (Inequality 4.4). Thus, E should guarantee that the migrator is left with energy after the zone sharing process, exactly as it was discussed for the donor. The same argument follows for the receiver (Inequality 4.5). As for the lower bound, selecting a very low fractional value for E will be too restrictive and would result in decreasing the probability of applying the ZS process, especially after the energy of the sensor nodes across the network is about to be fully consumed. In light of this, we selected a lower bound of 0.3 for E. Specifically, we study the performance for E values ranging from 0.3 to 0.8 with an 0.1 increment. We only present the results for a subset of these values that captures the main learned lessons from the study.

In addition to studying the ZS performance for $SC = 1$, i.e., testing the SHZS performance, we study the effect of changing the SC value between 5 and 15 hops (i.e., shares per zone). The reason for selecting 5 as the lower bound is to set a clear distinction between MHZS and SHZS. Based on our experiments, a smaller SC value, e.g., 2 or 3, is not sufficient to highly improve (or degrade) the performance of the SHZS scheme. In fact, our results show that pushing a hot zone only 2 or 3 hops away from its original storage sensor does not highly differ from pushing it for only one hop. In all these cases, the ZS ability to decompose the hotspot is limited. As for the upper bound for SC, setting SC to a very large value will result in completely disturbing the DCS index structure. This would result from the fact that hot zones will be sent very far from their original destination. Additionally, all mappings and routing decisions will be based on our ZS scheme rather than based on the underlined DIM scheme. To avoid this, we set the upper bound for SC to be 15. Specifically, we conducted experiments for SC values ranging from 5 to 15 with an increment of 1. We only present the results for a subset of these values that captures the main learned lessons from the study.

We now move on to discuss the results of our sensitivity analysis. We started our analysis by studying the effect of C on SHZS. To limit the effect of E, we set it to the maximum value of 0.8. This experiment resulted in a peak SHZS performance for $C = 2$, namely a 20% improvement on QoD with a 3.5% overhead on QoS. Thus, we select a default value of 2 for the C parameter.

Moving forward, we set $C = 2$ and study the effect of E on the SHZS performance. This

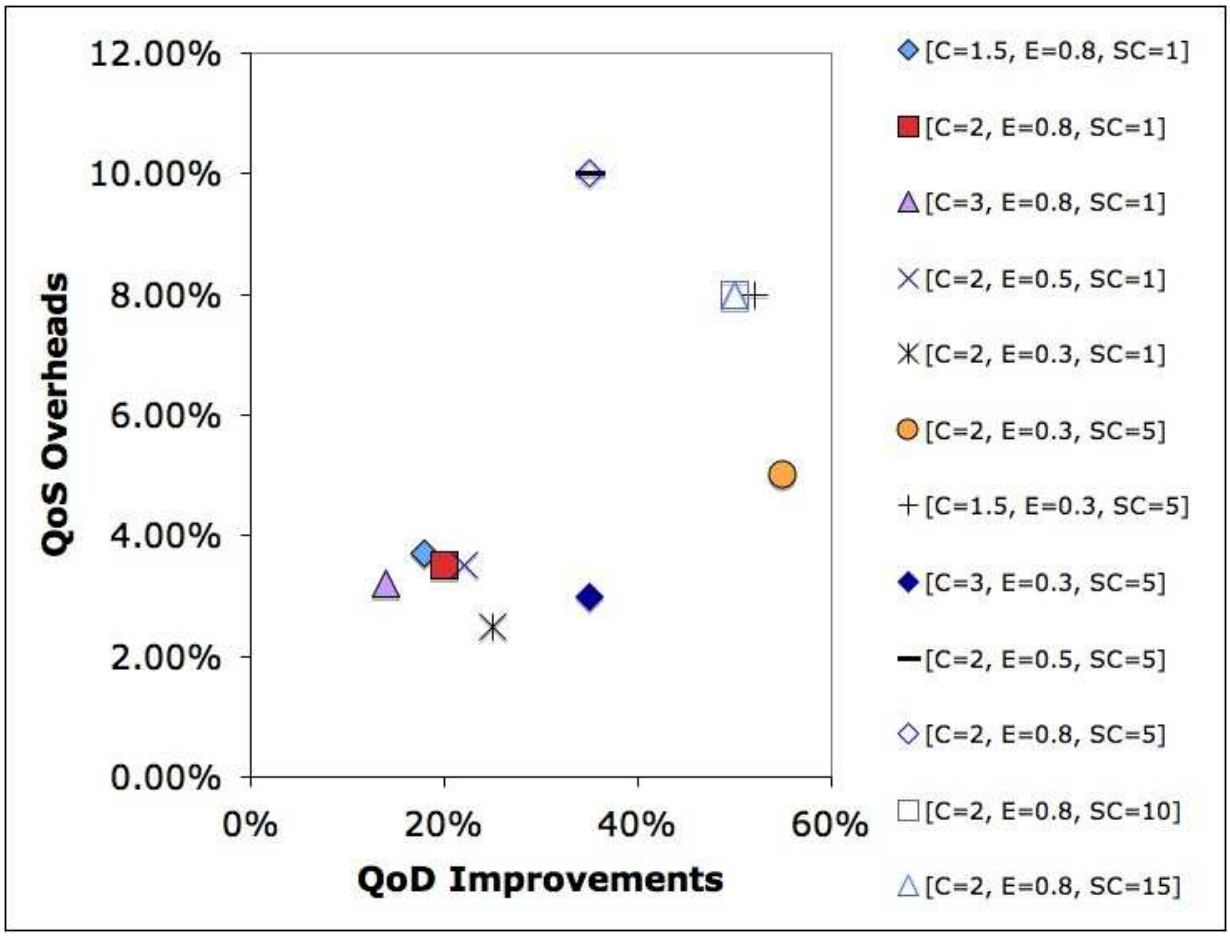


Figure 7: QoD Improvements vs QoS Overheads of the Different ZS Versions

resulted in a performance improvement that is inversely proportional to the value of E. The peak performance was achieved for $E = 0.3$ with a 25% QoD improvement and a 3% QoS overhead. Thus, we select a value of 0.3 to be the default value for E.

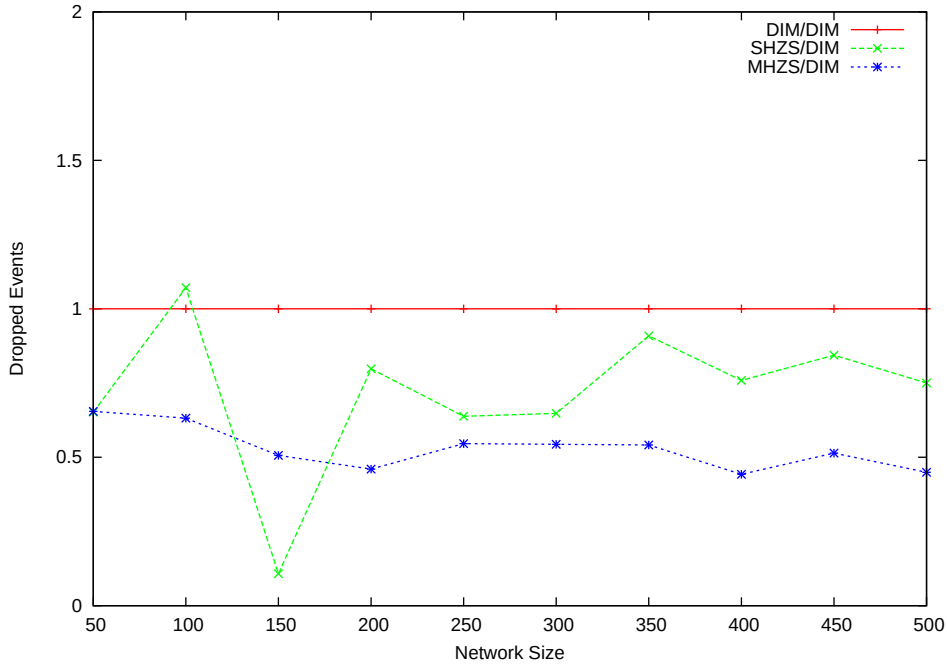
Using the default values for both C and E, we move on to study the performance of MHZS with $SC = 5$. As a first step, we verify that the default values already computed for SHZS achieve the best performance for MHZS as well. We start by studying the different C values while setting E to 0.3. For this study, the best performance is achieved when $C = 2$. QoD improvements are about 55% and QoS overheads are around 5%. Similarly, we test the MHZS performance for the different E values. The value of $E = 0.3$ continued to score the best MHZS performance. This verifies the selection of the default values for C and E.

Finally, we study the effect of SC on the MHZS performance. We present the results for SC values of 10 and 15 (while setting the C and E to their default values). The results show that SC has a limited effect on the MHZS performance. The MHZS performance was almost identical for the two SC values, 10 and 15, with a 50% QoD improvement and an 8% QoS overhead. Thus, we set the default value of 5 for the SC parameter.

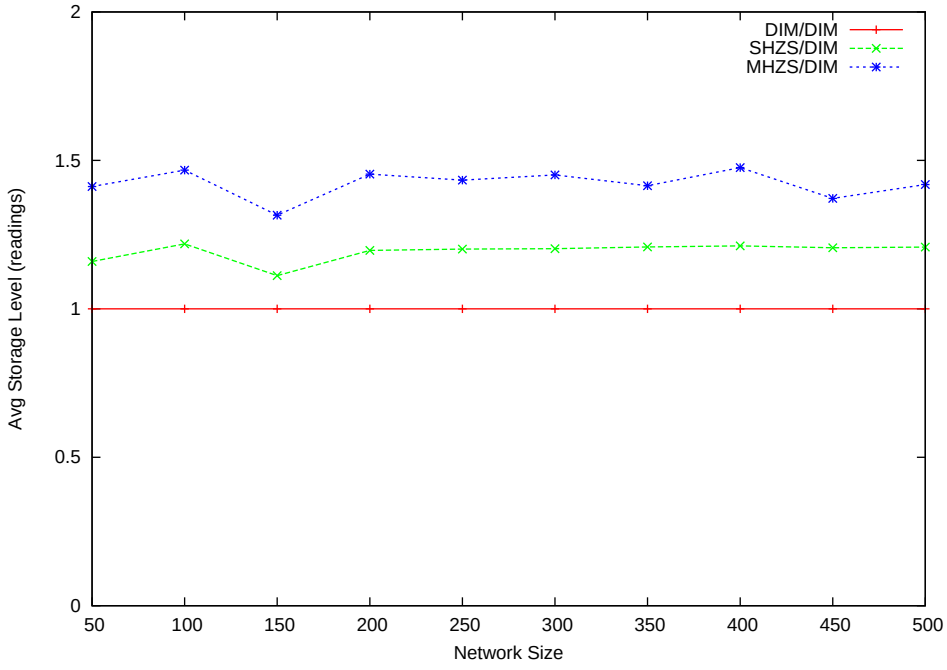
Using the default values for $C = 2$, $E = 0.3$, and $SC = 5$ (right most point on Figure 7), we test the ZS performance against the different hotspot types and settings. Our experimental results are shown in Figures 8 to 28. In these figures, we compare the performance of the basic DIM versus the performances of both SHZS and MHZS, with respect to our different performance measures. It is important to mention that we only included the DIM as the sole reference scheme as our simulations have shown that it completely outperform both the LS and the GHT schemes. Thus, in order to present more accurate and complete graphs, we only plot the results for our ZS scheme and those for DIM.

4.1.8.2 Single Static Storage Hotspots The following three results compare the ZS performance to that of the DIM when facing single static hotspots.

R1. QoD: Figure 8(a) presents the total number of events dropped by all network nodes for a 60% single storage hotspot. For the three simulated schemes, the number of dropped events is quite low and almost constant for networks of small sizes (less than 150 nodes), while it increases linearly for larger network sizes. Thus, the fluctuation of the ratios in the figure



(a) Dropped Events



(b) Average Node Storage

Figure 8: ZS: QoD Graphs for a 60% Single Storage Hotspot

(for network smaller than 200) does not actually have a big significance when comparing the performances of the three schemes. For networks of larger sizes, the main observation in this figure is that MHZS improves the performance by around 55% (over DIM) for all network sizes while SHZS improves the performance by at least 20% (over DIM).

Figure 8(b) shows the average node storage for networks with 60% storage hotspots. The figure shows that that MHZS improves DIM's performance by around 48% (around 5 readings per node) for all network sizes. Also, SHZS performs better than DIM by around 22% (an average of 2 readings per node). Note that we achieved similar results for hotspots of sizes up to 80%.

Based on the two figures, MHZS achieves a 50% QoD improvement over DIM while SHZS scores around 20% improvement. In general, a reduction in the number of dropped readings improves the average lifetime of a reading, which can be defined as the amount of time the reading will be stored in the network before being dropped. This consequently improves the network ability to efficiently provide answers for queries aiming any arbitrary set of readings throughout the network operation. Consequently, this results in improving the overall QoD.

R2. Load Balancing: Figure 9 compares the performance of the schemes in terms of the number of full nodes for networks experiencing 60% single storage hotspots. Recall that, by full nodes, we mean the nodes having caches full (i.e., storing 30 readings). This metric shows the ability of any scheme to decompose the hotspot and load-balance its readings across a larger number of network nodes. A successful load balancing strategy reduces the number of nodes reaching the edge of the storage capacity. In general, studying load balancing helps us in understanding the reasons behind the QoD improvements of our schemes.

The main observation from the figure is that MHZS decreases the number of full nodes by around 50% for small network sizes (less than 150) and the improvement increases to be around 70% for large networks. It is important to note how the performance of the scheme scales with the network size and how its load balancing effect becomes more obvious for larger networks. On the other hand, SHZS decreases the number of full nodes by around 35% compared to DIM. This result shows that ZS, especially MHZS, improves the network ability to maintain a higher portion of the hotspot readings by sending these readings for storage away from the hotspot area. This consequently increases the number of sensors responsible

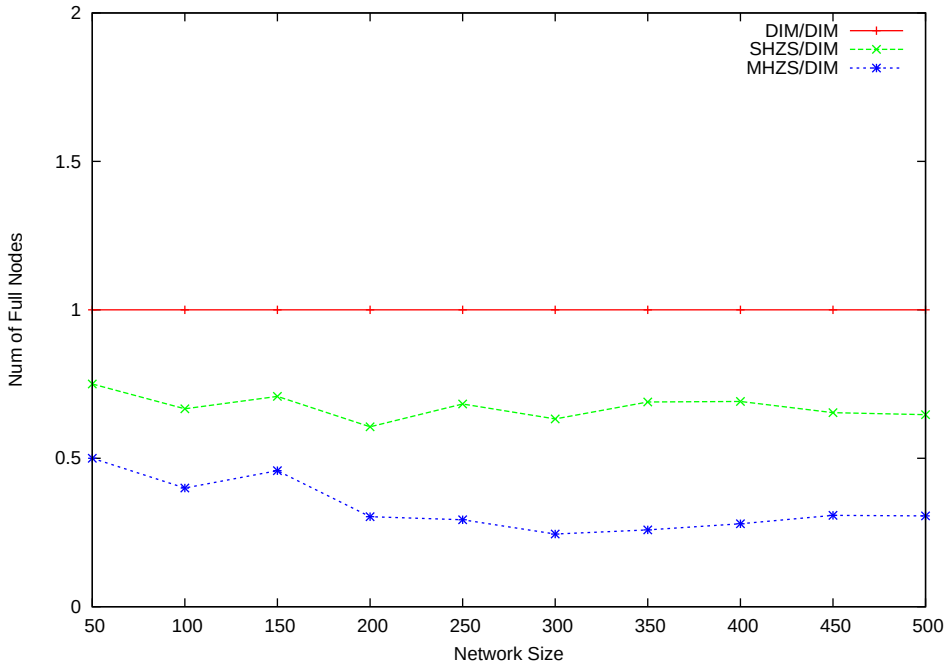
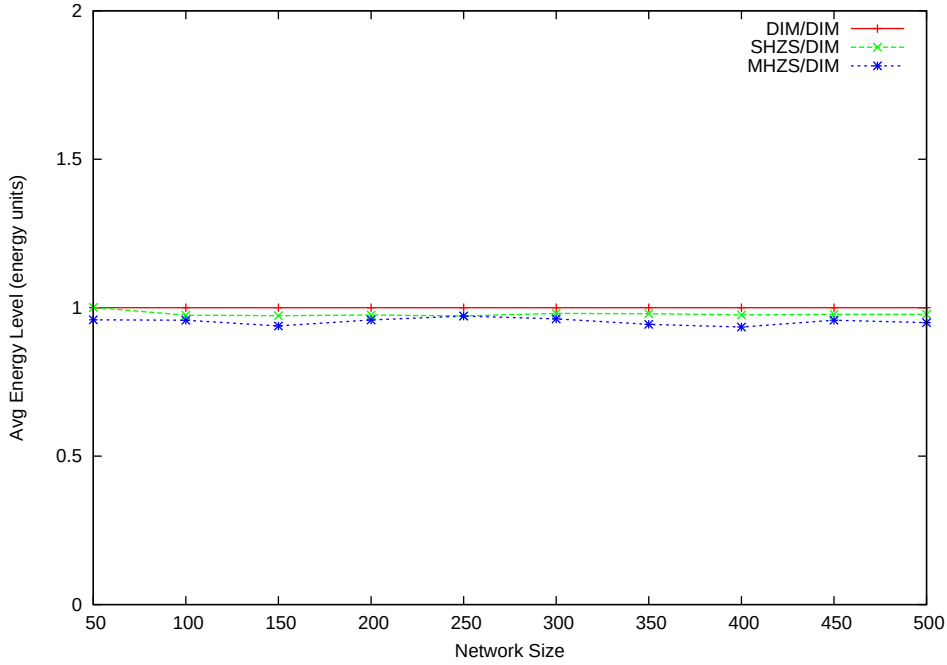


Figure 9: ZS: Number of Full Nodes for a 60% Single Storage Hotspot

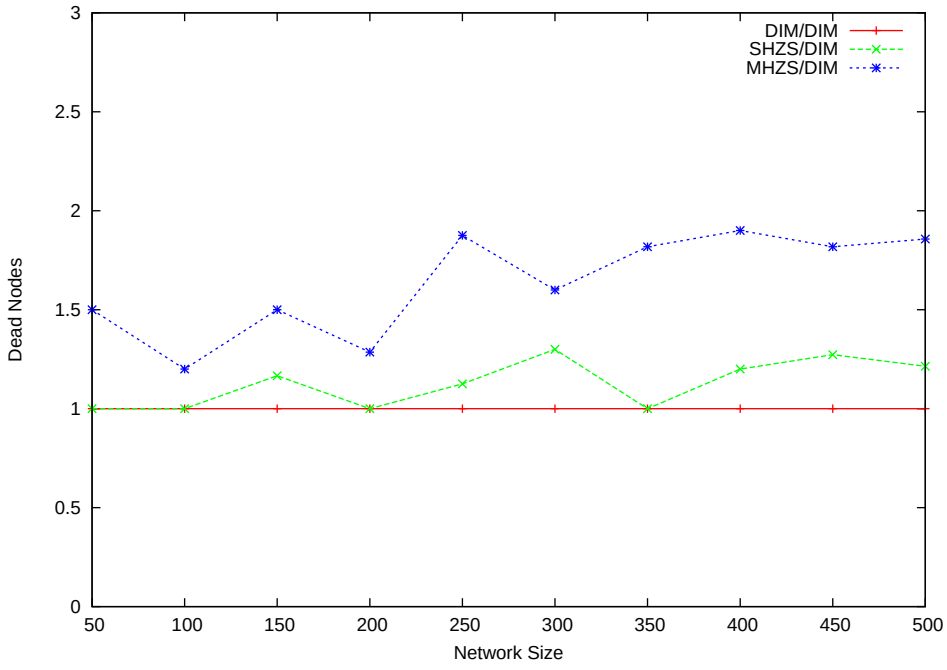
for storing the hotspot readings. Note that we achieved similar results for hotspots of sizes up to 80% hotspots.

Now that we have studied the ZS performance in terms of QoD, it is important to compare the energy consumption of ZS to that of the DIM to make sure that the QoD improvement does not come with a high energy overhead that would be imposed on the DIM scheme when applying ZS.

R3. Energy Consumption: Figure 10(a) presents the average node energy level for networks experiencing a 60% single storage hotspot. The figure shows that introducing ZS schemes slightly decreases the average sensor energy by around 3% for SHZS and 5% for MHZS when compared to DIM. Note that we achieved similar results for hotspots up to 80%. Figure 10(b) presents the number of dead nodes for networks with an 80% hotspot. The figure shows that MHZS increases node deaths (compared to DIM) by around 30% for networks less than 200 nodes and by around 100% for larger networks. This increase in node deaths is equivalent to 2% of the network size for small networks and by around 3% of the



(a) Average Node Energy for a 60% Single Storage Hotspot



(b) Dead Nodes for an 80% Single Storage Hotspot

Figure 10: ZS: Energy Consumption Graphs for Single Storage Hotspots

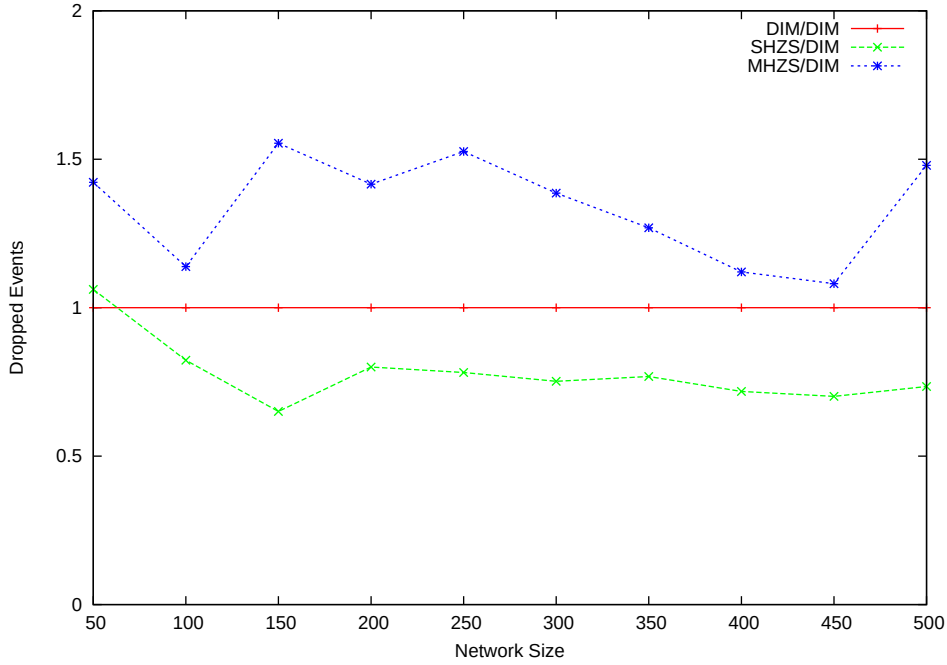
network size for larger networks. The important observation from the two figures is that the energy consumption overhead imposed by applying ZS on top of DIM is relatively small.

In conclusion, the above results show that applying ZS schemes on top of DIM improves DIM's QoD when facing single storage hotspots while imposing a slight energy consumption overhead. For SHZS, the QoD improvements are around 20% while the energy consumption overheads are around 3% per node. For MHZS, the QoD improvements are around 55% while the energy consumption overheads are around 5% per node. The results are valid for hotspot sizes up to 80%. In general, MHZS performs better than SHZS when facing single storage hotspots.

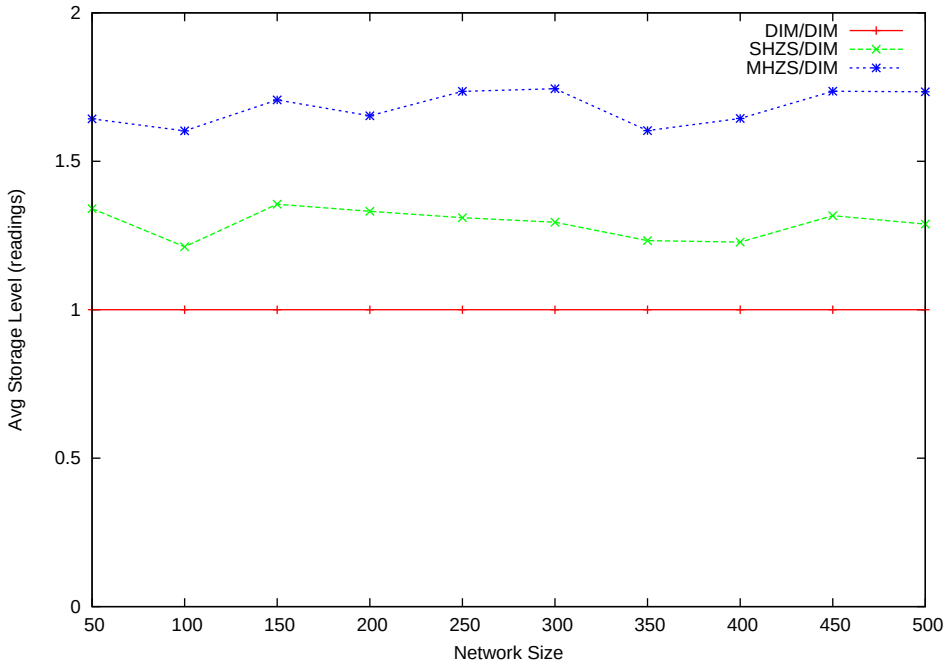
4.1.8.3 Multiple Simultaneous Static Storage Hotspots We now study the performance of our ZS schemes compared to DIM when facing multiple simultaneous and static storage hotspots. As for the single hotspots, we conduct our study in terms of QoD (R1), load balancing (R2), and energy consumption (R3). The results presented in this section are based on simulating networks with two simultaneous hotspots. Recall that an x% multiple hotspot means that at least x% of the readings fall in the two hotspots with each reading falling in any of the two hotspots with equal probability, i.e., at least $x/2$% of the readings are expected to fall in each of the two hotspots.

R1. QoD: Figure 11(a) presents the total number of events dropped by all network nodes in networks facing a 60% multiple hotspots. For the three simulated schemes, the number of dropped events is quite low and almost the same for networks of small sizes (less than 100 nodes). For larger network sizes, the number of dropped events increases linearly. Specifically, SHZS outperforms DIM by around 20% while MHZS performs worse than DIM for networks larger than 100 nodes (though the performance of all three schemes is quite comparable). Figure 11(b) shows the average node storage for networks with 60% multiple storage hotspots. The main observation in this figure is that SHZS improves performance by around 30%. Based on both figures, SHZS is able to score at least 20% QoD improvement over DIM for the case of multiple hotspots. We achieved similar results for hotspots of sizes up to 80%.

The bad MHZS performance results from its hotspot decomposition strategy which is



(a) Dropped Events for a 60% Multiple Storage Hotspot



(b) Average Node Storage for an 80% Multiple Storage Hotspot

Figure 11: ZS: QoD Graphs for Multiple Storage Hotspots

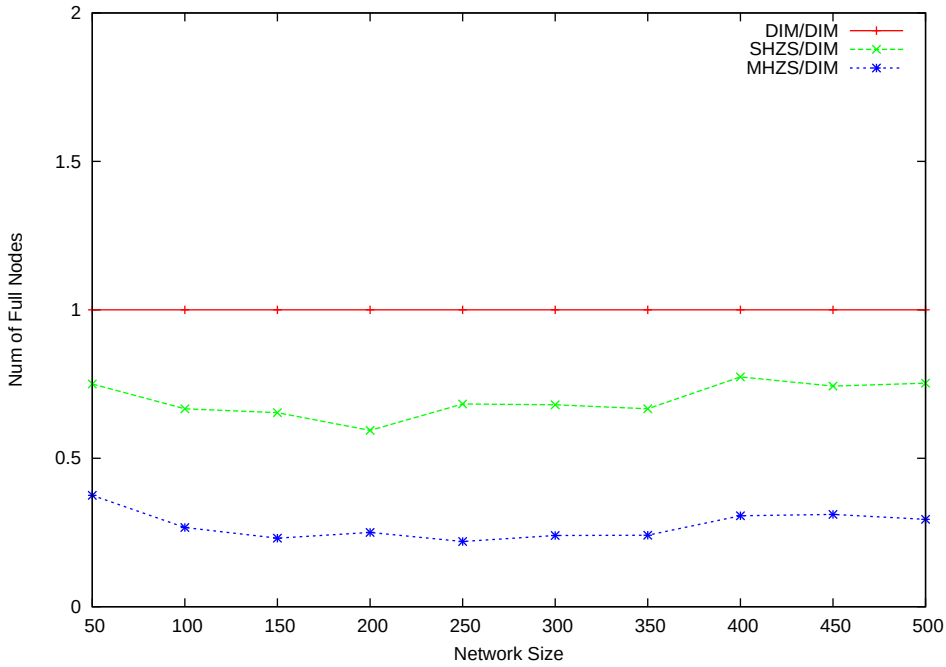
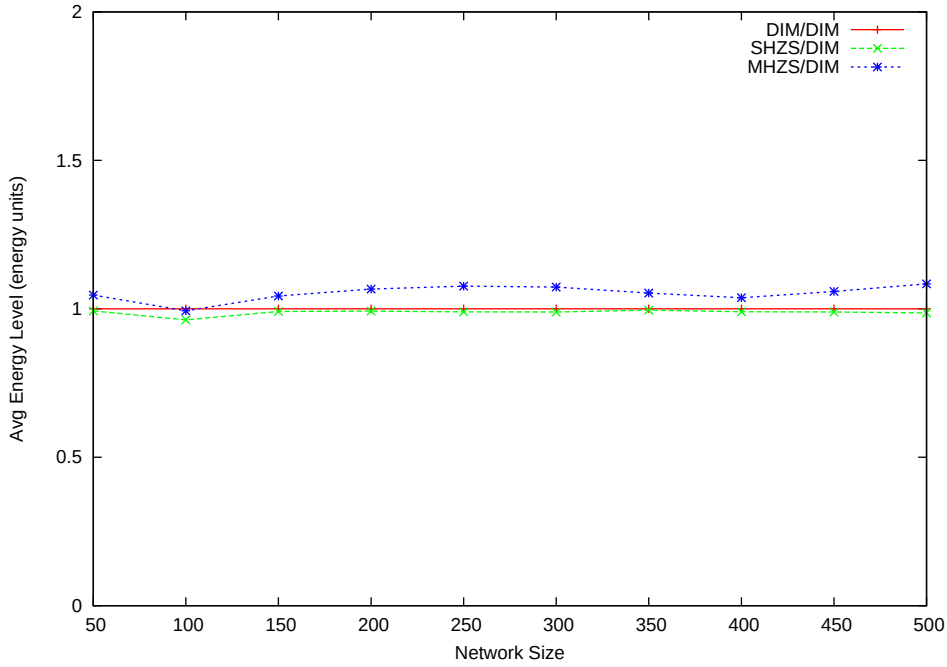


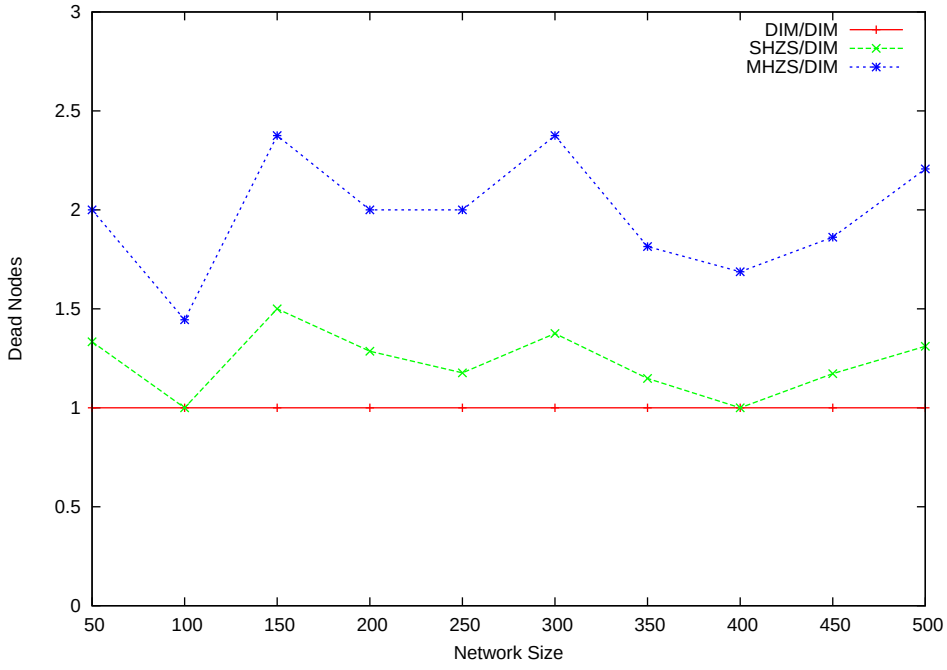
Figure 12: ZS: Number of Full Nodes for a 60% Multiple Storage Hotspot

based on decomposing the hot zone into sub-zones and sending part of these hot sub-zones away from the hotspot area. By carefully studying our experiments' input data and their results, we realized that this MHZS strategy causes further *collisions* among the hot sub-zones of the different hotspots. This can be viewed as the formation of additional hotspots which further trigger the MHZS scheme for their decomposition. This symptom results in slightly decreasing the dropped events of the MHZS scheme compared to those of DIM. It is worth mentioning that this collision effect does not take place in all multiple hotspot cases. In fact, depending on the subranges of the two hotspots, MHZS performs better than SHZS in some of the cases where no collisions take place.

R2. Load Balancing: Figure 12 presents the number of full nodes for networks with 60% multiple storage hotspots. The figure shows that SHZS decreases the number of full nodes by around 25%. This shows that SHZS achieves a better load balancing of the hotspot data than that of DIM. This consequently explains the QoD improvement achieved by SHZS. This was true for hotspots of sizes up to 80%.



(a) Average Node Energy



(b) Dead Nodes

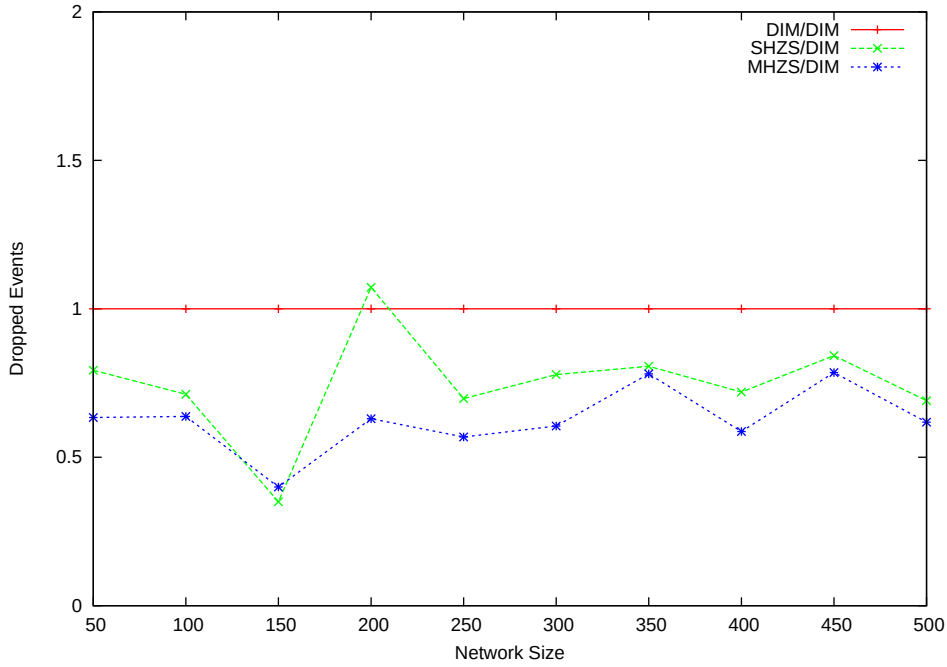
Figure 13: ZS: Energy Consumption Graphs for 60% Multiple Storage Hotspots

R3. Energy Consumption: Figure 13(a) shows the average node energy level for networks with 60% hotspots. The figure shows that SHZS does not impose except a 2% energy consumption overhead on DIM. This is mainly due to the fact that each zone can only be shared once. Figure 13(b) shows the number of node deaths for networks with 60% hotspots. The figure shows that SHZS increases the number of dead nodes by around 25% over DIM. As the number of dead nodes achieved by DIM is at most 5% of the network size (for all network sizes), the performance degradation caused by DIM is around 1.25% of the network size. In general, the two figures show that SHZS adds a small energy consumption on the DIM scheme in the case of multiple storage hotspots.

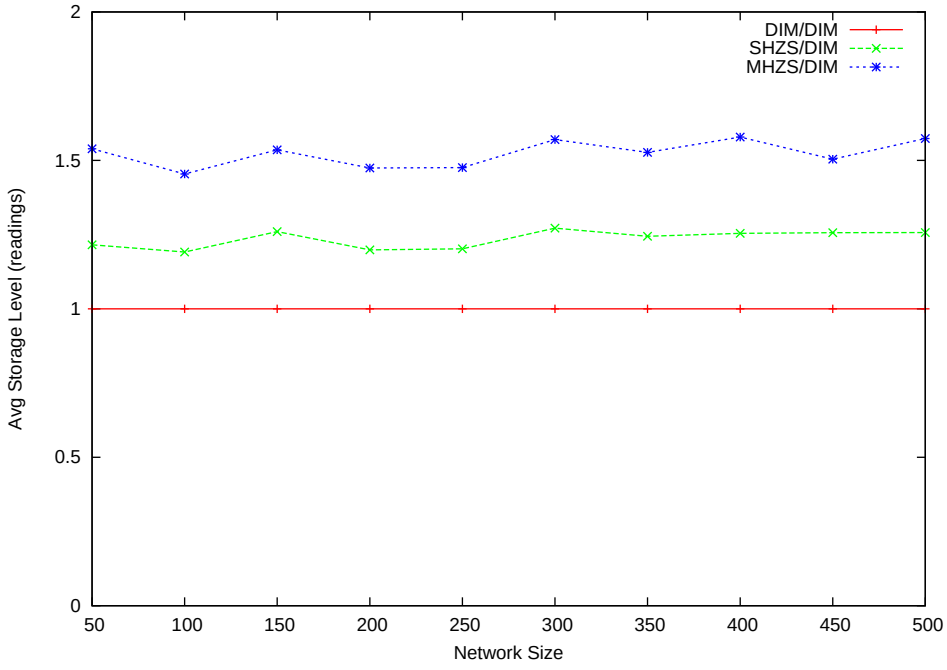
In conclusion, the implementation of the ZS schemes on top of DIM slightly copes with the problem of multiple storage hotspots. SHZS achieves at least 20% QoD improvement over DIM. This comes a 2% additional energy consumption overhead. As for MHZS, it may cause a QoD downgrading by creating further collisions across the hot sub-zones and subsequent formations of new hotspots. This is valid for hotspots of sizes up to 80%.

4.1.8.4 Moving Storage Hotspots We now study the ZS performance against the third type of hotspots we are interested in, which are moving storage hotspots. Our study continues to be in terms of the three dimensions: QoD (R1), load balancing (R2), and energy consumption (R3). We simulated an x% moving hotspot as follows. Each run has been divided into 5 steps. The hotspot starts in range $[t_1, t_1 + i]$ in the first step, then moves on to $[t_1 + i, t_1 + 2i]$ in the second step, etc. In each of the steps, x% of the generated readings fall in the step's hotspot range. We simulated hotspot sizes up to 50%.

R1. QoD: Figure 14(a) presents the number of events dropped by networks facing a 40% moving hotspot. The figure shows that the three schemes achieve a similar performance for networks less than 200 nodes. The number of dropped events is relatively small for these network sizes. For larger networks, SHZS improves DIM's performance by at least 18% while MHZS improves DIM's performance by at least 20%. This shows the ZS ability to improve the data persistence against moving hotspots. Figure 14(b) presents the average node storage for networks with a 40% moving hotspot. The figure shows that MHZS improves average node storage by around 50% while SHZS improves it by around 25%. Overall, the two figures



(a) Dropped Events



(b) Average Node Storage

Figure 14: ZS: QoD Graphs for a 40% Moving Storage Hotspot

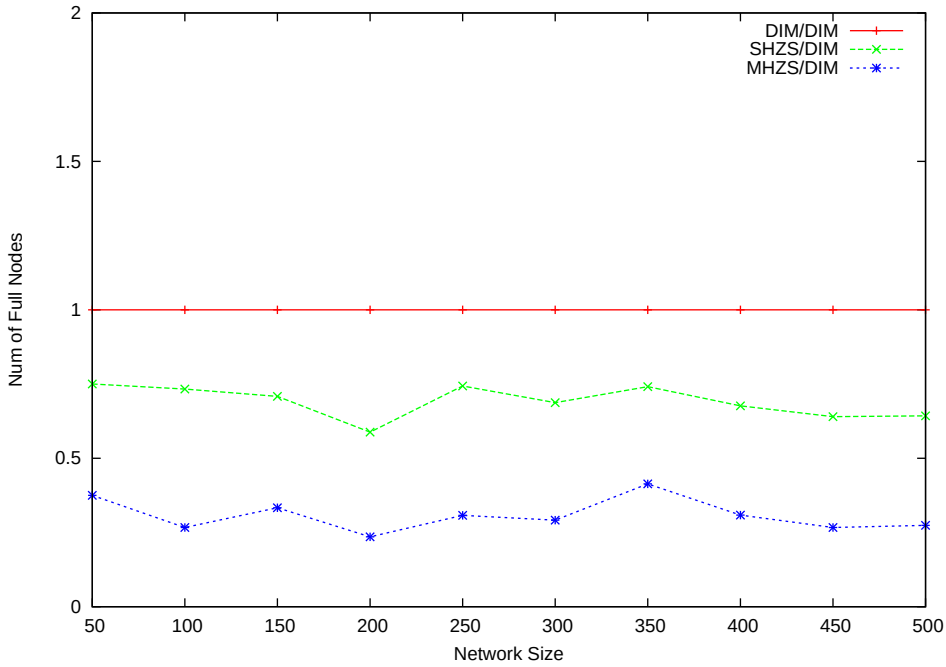
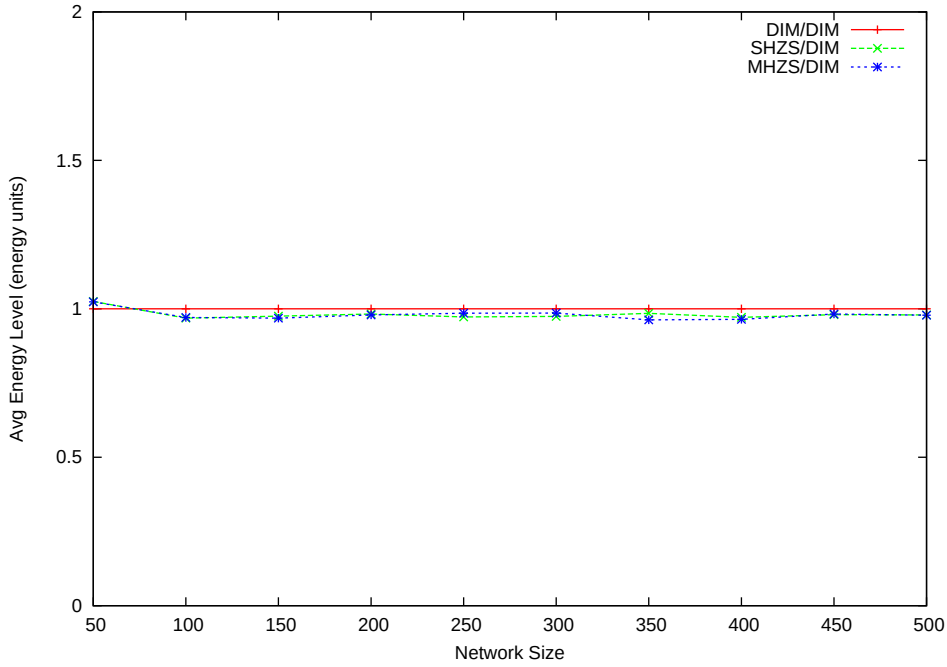


Figure 15: ZS: Full Nodes for a 40% Moving Storage Hotspot

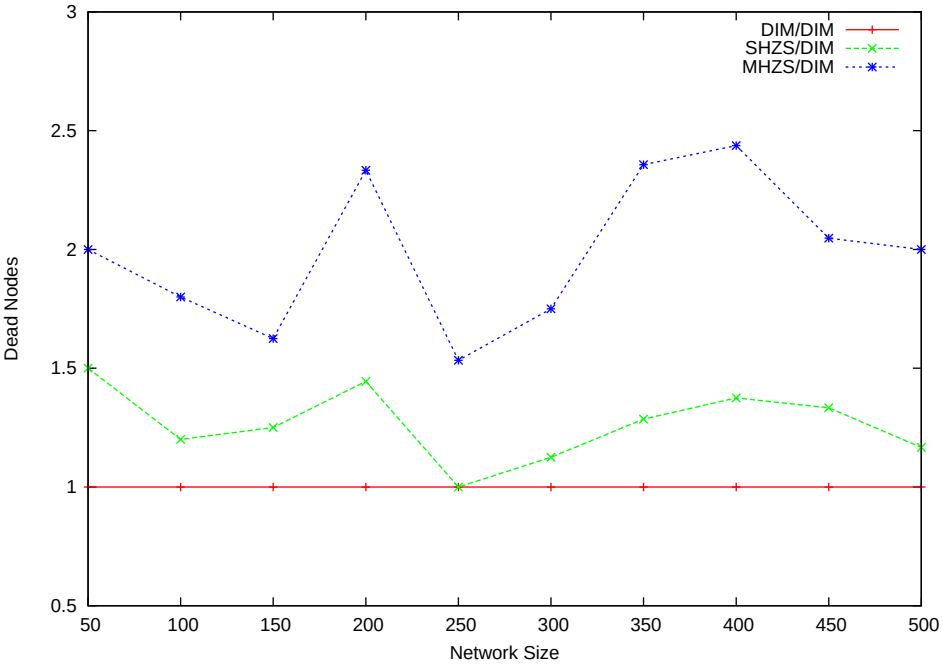
show the ability of ZS to decompose moving hotspots in a better way than DIM.

R2. Load Balancing: Figure 15 presents the number of full nodes for networks with a 40% moving hotspot. The figure clearly shows that MHZS decreases the number of full nodes by around 70% compared to DIM. SHZS also performs better than DIM by around 25%. This shows that MHZS considerably load balances the hotspot data throughout the network lifetime. This results in sending hot sub-zones away from the hotspot area(s). Consequently, this increases the number of sensor nodes responsible for storing the hotspot data. This results in decreasing the number of sensor nodes with saturated caches.

R3. Energy Consumption: Figure 16(a) presents the average node energy for networks with a 40% moving hotspot. The figure shows that the three schemes achieve almost the same performance, with the ZS schemes having around 1 or 2 readings less than DIM (specifically, 2% overhead for SHZS and 3% overhead for MHZS). Figure 16(b) presents the number of node deaths for networks with a 40% moving hotspot. In general, the number of dead nodes for DIM is at most 5% of the network size (for all network sizes). SHZS increases node deaths



(a) Average Node Energy



(b) Dead Nodes

Figure 16: ZS: Energy Consumption Graphs for a 40% Moving Storage Hotspot

by around 20% while qMHZS increases node deaths achieved by DIM by around 100% (or around 5% of the network size). In general, this shows that ZS imposes a moderate energy consumption overhead on DIM.

In conclusion, the ZS schemes are able to perform slightly better than the basic DIM against moving hotspots. The 20% QoD improvement, though it is not that big, comes with a 3% energy consumption overhead.

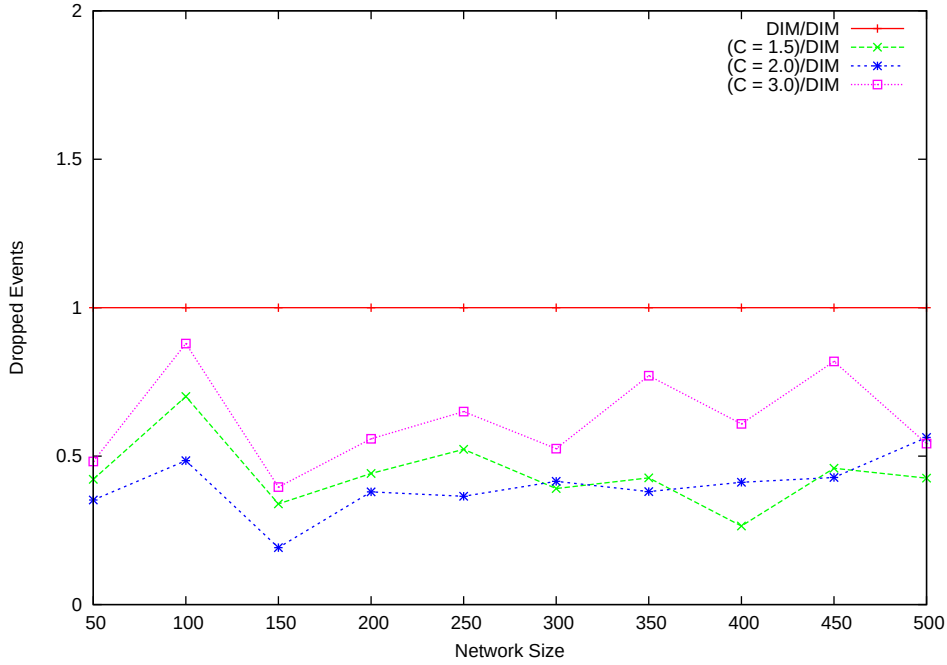
We now move on to the sensitivity analysis of ZS. We study the effect of the different ZS parameters on the ZS performance. We concentrate on the effect of three main parameters: the storage level threshold C , the energy level threshold E, and the zone share count SC. We present the results of our study in the following three subsections. For each of our studies, we concentrate on changing the value of the parameter under concern while keeping the values of the other parameters set to their default values (determined in Section 4.1.8.1). Note that although we studied the performance of ZS for many parameter values and combinations, we present a subset of results that captures the most important learned lessons.

To maximize the effect that any of the three parameters will have on ZS, we focus on studying MHZS rather than SHZS. This is simply due to the fact that MHZS applies the ZS process much more frequently than SHZS. Thus, the effect of any of these parameters on the MHZS performance for any network setting will be higher than that effect the same parameter has on SHZS. It is important to note that, throughout the following fine tuning results, the hot ranges for multiple hotspots were carefully selected to avoid the symptom of collision effects (and hotspot reformation) as much as possible.

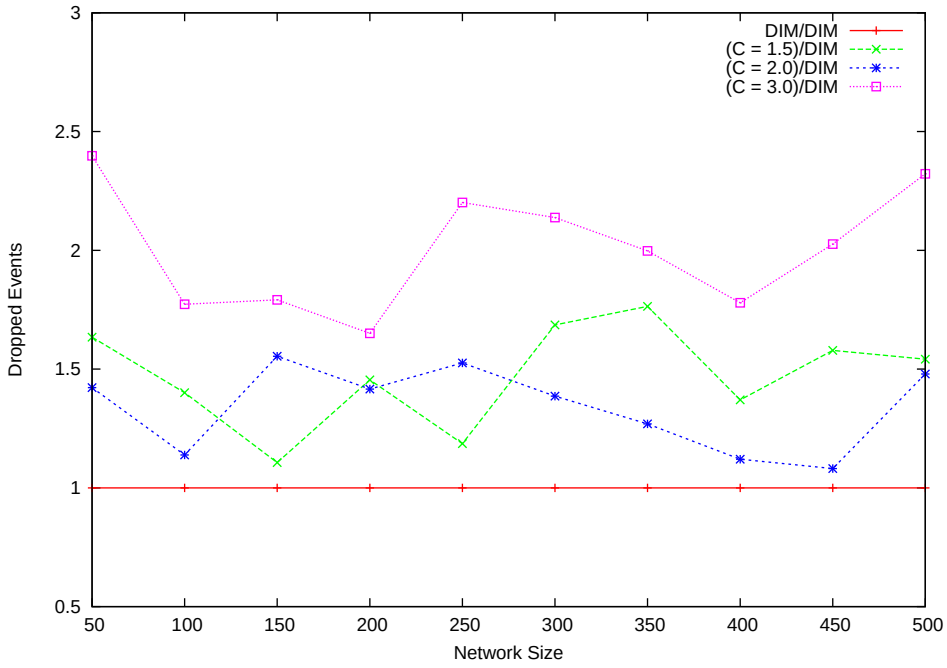
4.1.8.5 Effect of Storage Level Threshold on MHZS Performance

In this subsection, we study the effect of the storage level threshold, namely the C_1 and C_2 thresholds of the DMC, on the ZS performance. To maintain the same threshold among the three participants of the ZS process (the sender, the migrator, and the receiver), we mainly set the two parameters to a single value that we refer to by the variable C. Recall that our default value for C was 2. As discussed in Section 4.1.8.1, we study C values ranging from 1.5 to 3.

The following three results compare the MHZS performance for the different C values studying both single and multiple hotspots of sizes up to 80%.



(a) 80% Single Storage Hotspot



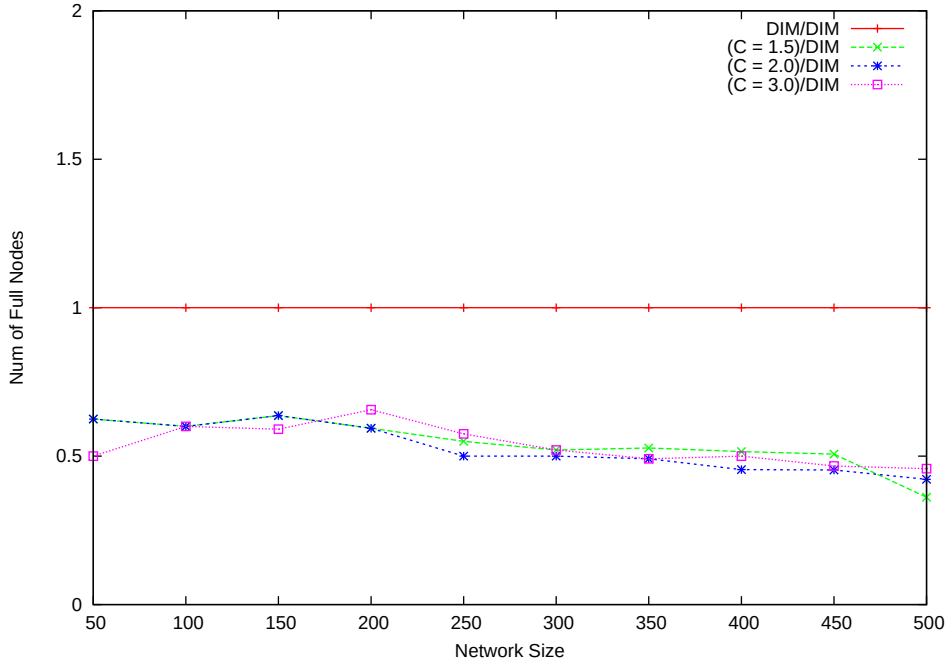
(b) 60% Multiple Storage Hotspots

Figure 17: ZS: Dropped Events for Various C Values

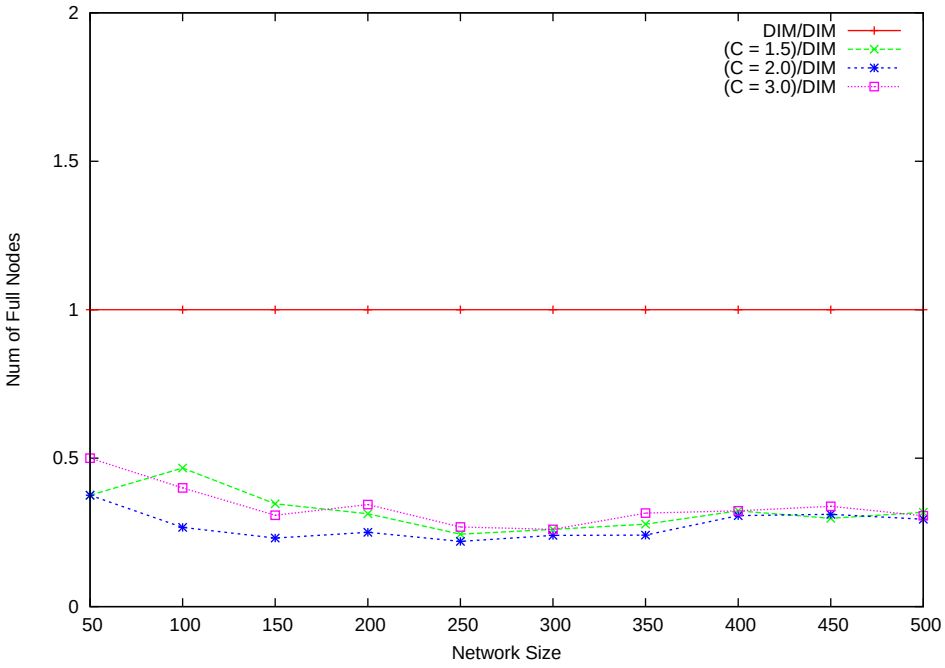
R1. QoD: Figures 17(a) and 17(b) compare the number of dropped events of the different MHZS versions for the 80% single hotspot case and the 60% multiple hotspot case, respectively. The figures show that increasing the storage level threshold generally increases the number of dropped events. For both single and multiple hotspots, the $C = 2$ MHZS version performs the best in terms of QoD. For single hotspots, the $C = 2$ MHZS version achieves around 55% QoD improvement over DIM, while $C = 1.5$ improves QoD by around 52% and $C = 3$ improves QoD by around 35%. For multiple hotspots, all schemes perform worse than DIM due to the collision effect (as discussed in Section 5.6.1.2). However, the $C = 2$ MHZS version continues to perform the best among the three versions by decreasing performance by around 35% as opposed to 50% for the $C = 1.5\%$ version and 100% for the $C = 3$ version. This result shows that the $C = 2$ version is the most appropriate version to deal with our hotspot scenarios as it applies the zone sharing process in a moderate way without over-reacting to or under-estimating the hotspot. Note that the $C = 2$ MHZS version continues to perform the best among all versions when the collision problem is avoided for the multiple hotspot. In such a case, QoD improvements can reach 65% as opposed to QoD improvements of around 60% and 45% for the $C = 1.5$ and $C = 3$ cases, respectively. We achieved similar results for hotspots of sizes up to 80%.

R2. Load Balancing: Figures 18(a) and 18(b) compare the number of full nodes of the different MHZS versions for the 80% single hotspot case and the 60% multiple hotspots case, respectively. The important observation is that $C = 2$ decreases the number of full nodes the most for both single and multiple hotspots (by around 55% for single hotspots and 75% for multiple hotspots). The number of full nodes is slightly less for the other two MHZS versions (around 50%. This shows that changing the C value above and below a threshold slightly downgrades the load balancing performance of MHZS. Thus, applying the ZS process for a very large or very low number of times decreases the load balancing ability of the ZS scheme. This is valid for single and multiple hotspots of sizes up to 80%.

R3. Energy Consumption: Figures 19(a) and 19(b) compare the number of dead nodes of the different MHZS versions for the 80% single hotspot case and the 40% multiple hotspots case, respectively. In both cases, the numbers of dead nodes for DIM are slightly small (3% of the network size for single hotspots and 4% of the network size for multiple hotspots).

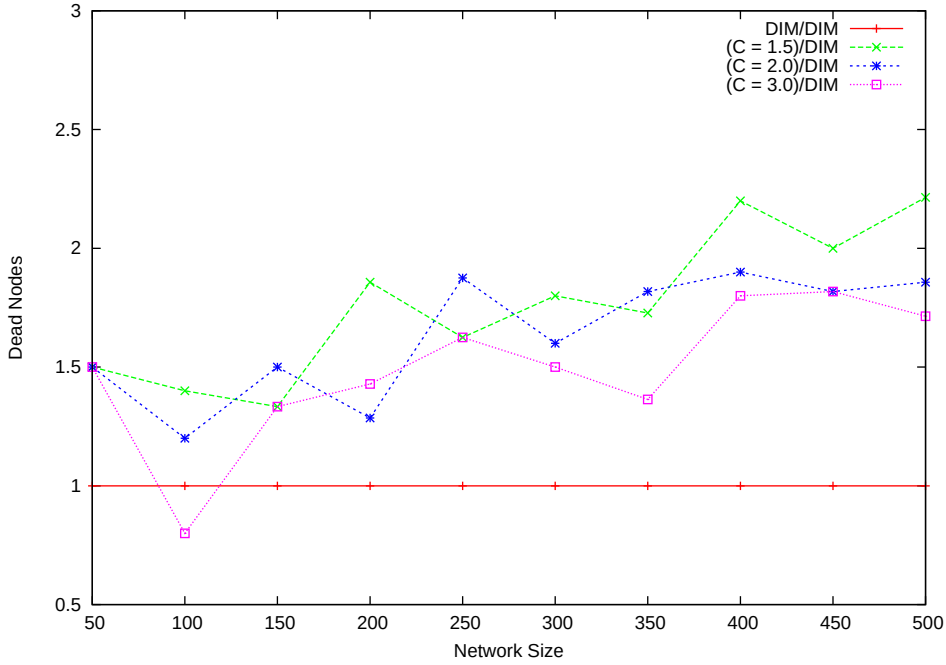


(a) 80% Single Storage Hotspot

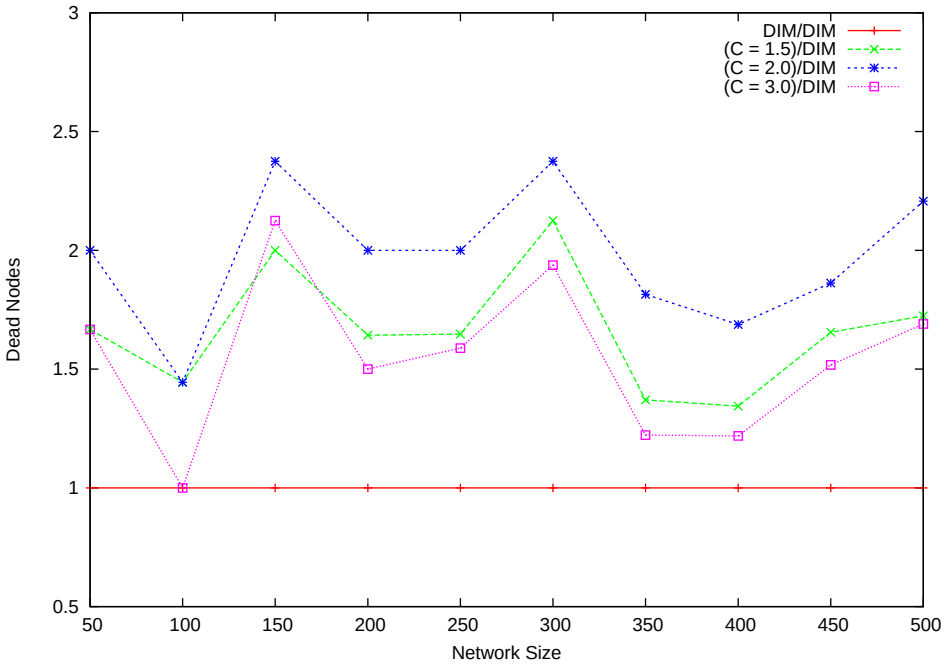


(b) 60% Multiple Storage Hotspots

Figure 18: ZS: Number of Full Nodes for Various C Values



(a) Dead Nodes for an 80% Single Storage Hotspot



(b) Dead Nodes for 40% Multiple Storage Hotspots

Figure 19: ZS: Dead Node Graphs for Various C Values

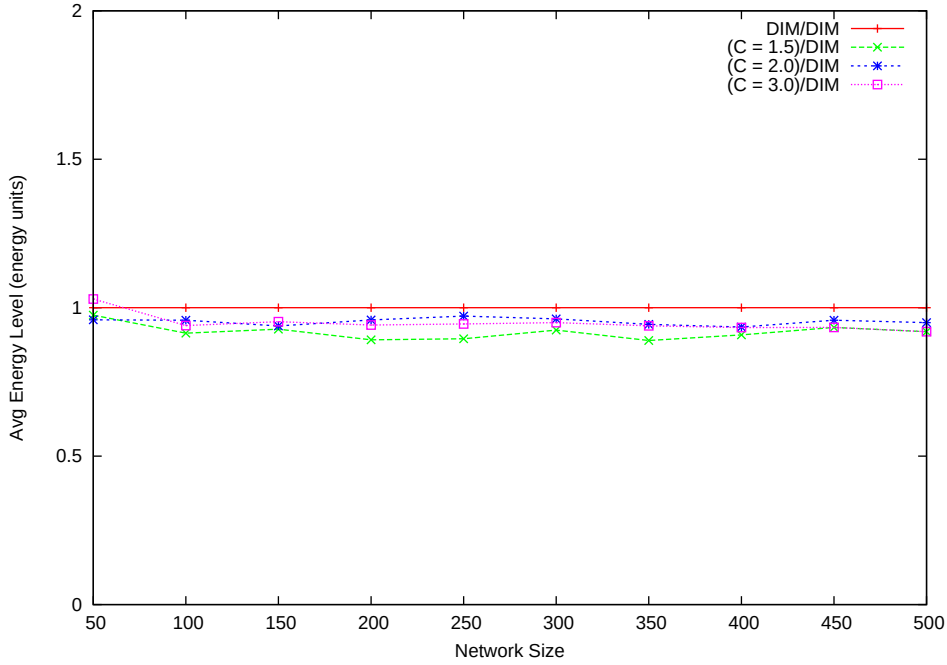
The main observation is both figures is that the performance downgrade imposed by all MHZS versions is quite small. For single hotspots, the downgrade ranges 75% to 100% over DIM, i.e., 2.25% to 3% of the network size. For multiple hotspots, the performance downgrade ranges from 75% to 125% over DIM, i.e., 3% to 5% of the network size. In general, increasing the C value decreases the number of dead nodes. This is due to the fact that increasing C results in applying the ZS process fewer times. This results in decreasing the energy consumption overhead imposed on each node.

This can also be observed in Figures 20(a) and 20(b) comparing the average node energy of the different MHZS versions for the 60% single hotspot case and the For 40% multiple hotspots case, respectively. The first figure shows that $C = 1.5$ downgrades energy consumption of DIM by around 10% for single hotspots as opposed to 8% and 5% downgrades for the $C = 2$ and $C = 3$ MHZS versions, respectively. The second figure shows that MHZS improves energy consumption overhead for multiple hotspots. This is due to increasing the event shedding as a result of the collision effect. With more events dropped (readings/queries), the energy consumption burden imposed on sensor nodes falling in the storage hotspot relatively decreases. For the cases where the collision effect was avoided, increasing the C value continued to slightly decrease the energy consumption overhead imposed on sensor nodes. Overheads ranges from 15% for $C = 1.5\%$ to 10% for $C = 3$. Overall, this shows that the average energy consumption of MHZS does not largely vary for small changes in the C value.

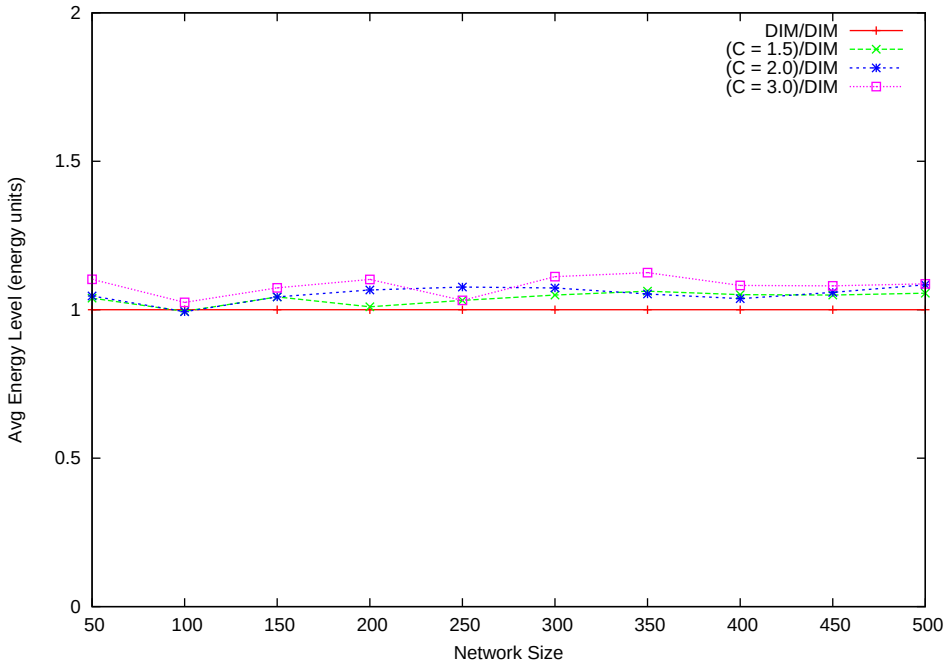
All the above three results show that the MHZS performance depends on the value of the C parameter. Increasing the C value beyond 2 decreases the QoD of MHZS by around 20% (compared to DIM's performance). This is mainly due to the decrease in the number of times the ZS process is applied. QoD of MHZS is almost identical for C values between 1.5 and 2. In terms of energy consumption, increasing the C value results in decreasing the energy consumption overhead of MHZS. For single hotspots, overhead increases by around 3% (compared to DIM) when changing the C value from 1.5 to 2 or from 2 to 3.

4.1.8.6 Effect of Energy Level Threshold on MHZS Performance

In this subsection, we move on to study the effect of the energy level threshold, namely the E_i thresholds of the DMC, on the MHZS performance. As in the previous experiment, we set the three



(a) Average Node Energy for a 60% Single Storage Hotspot



(b) Average Node Energy for 40% Multiple Storage Hotspots

Figure 20: ZS: Energy Consumption Graphs for Various C Values

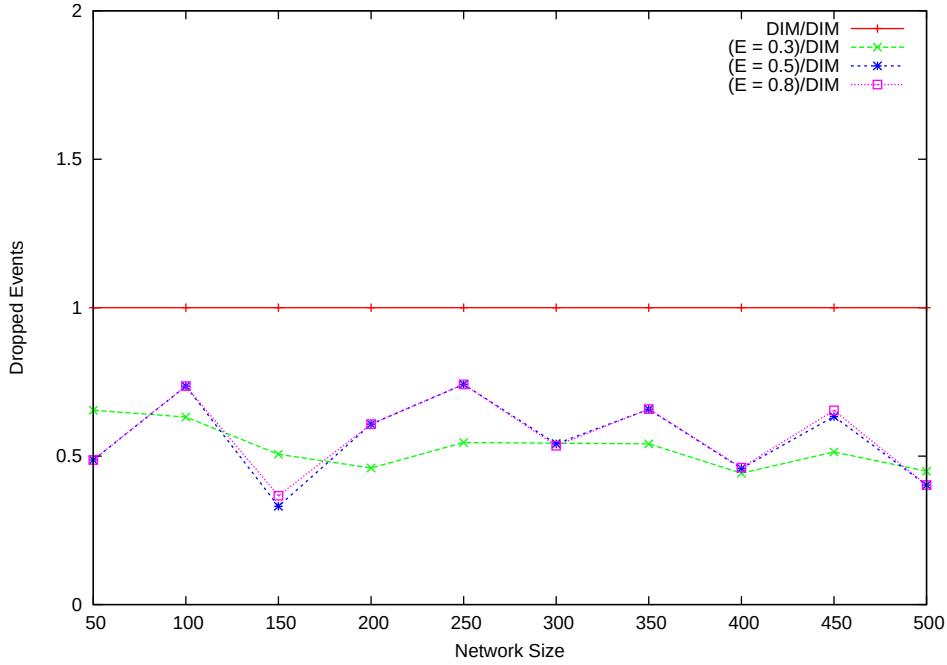
parameters to a single value that we refer to by the variable E to maintain the same performance among the nodes participating in the ZS process. Recall that our default value for E was 0.3. As discussed in Section 4.1.8.1, we study E values ranging from 0.3 to 0.8.

We concentrate on studying the MHZS rather than the SHZS as the MHZS is the one that applies the ZS process the most. The following three results compare the MHZS performance for the different values of E when facing both single and multiple hotspots. We conducted experiments for hotspots of sizes up to 80%.

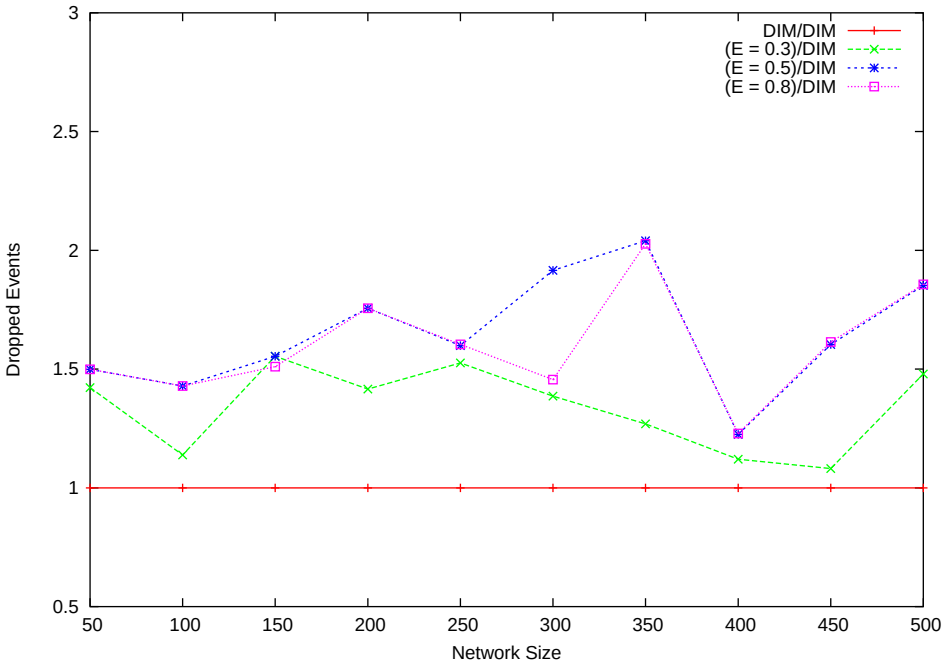
R1. QoD: Figures 21(a) and 21(b) compare the number of dropped events for MHZS, applying the three E values, for the 60% single hotspot case and 60% multiple hotspots case, respectively. The figures show changing the E value between 0.5 and 0.8 does not highly affect the ZS performance for both single and multiple hotspots. Decreasing E to 0.3 results in improving QoD by around 20% for single hotspots and 25% for multiple hotspots. For single hotspots, QoD improvements range from 35% from $E = 0.5$ (or more) to around 55% for $E = 0.3$. In general, this result shows that decreasing E results in increasing the number of times the ZS process is applied. This consequently results in a better decomposition of the hotspots and a better QoD for the network.

R2. Load Balancing: The same lesson learned from $R1$ is strengthened by Figures 22(a) and 22(b) comparing the number of full nodes for MHZS, applying the three E values, for the 80% single hotspot case and 60% multiple hotspots case, respectively. The two figures show that MHZS performance is almost identical for E values 0.5 and 0.8. As for $E = 0.3$, the number of full nodes drops by around 10% for single hotspots and 15% for multiple hotspots. This shows that important effect of the energy level threshold on the load balancing capability of the ZS scheme.

R3. Energy Consumption: Figures 23(a) and 23(b) compare the number of dead nodes for MHZS, applying the three E values, for the 80% single hotspot case and 60% multiple hotspots case, respectively. Both figures show that the MHZS versions with $E = 0.5$ and $E = 0.8$ perform almost identically. Decreasing the E value to 0.3 increases the number of dead nodes by around 10% to 30% (compared to DIM) for single hotspots and 50% to 70% (compared to DIM) for multiple hotspots. This is due to the fact that decreasing E results in fewer number of times of zone shares. This causes some nodes falling in the hotspot

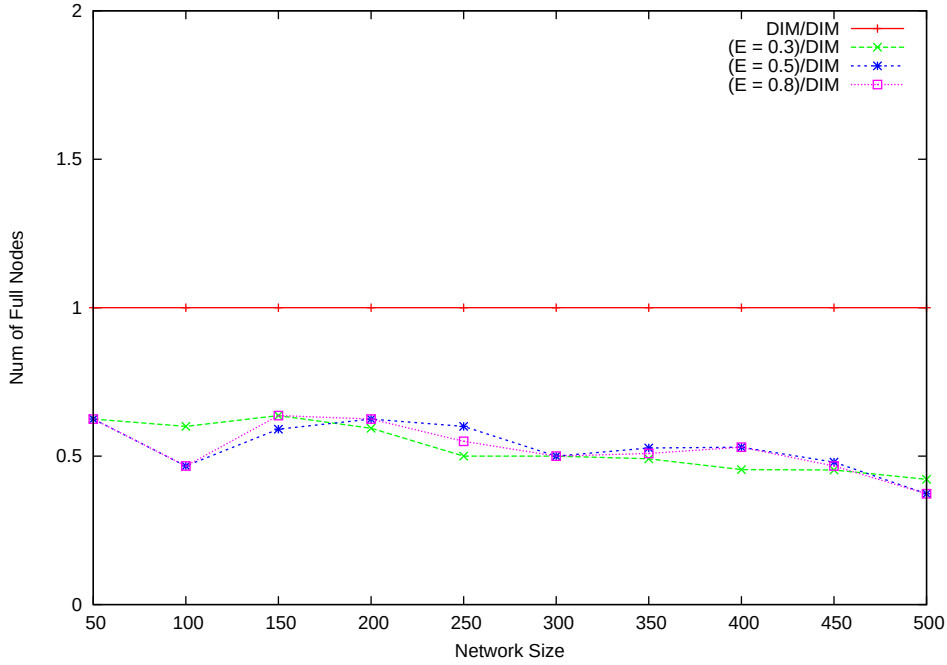


(a) 60% Single Storage Hotspot

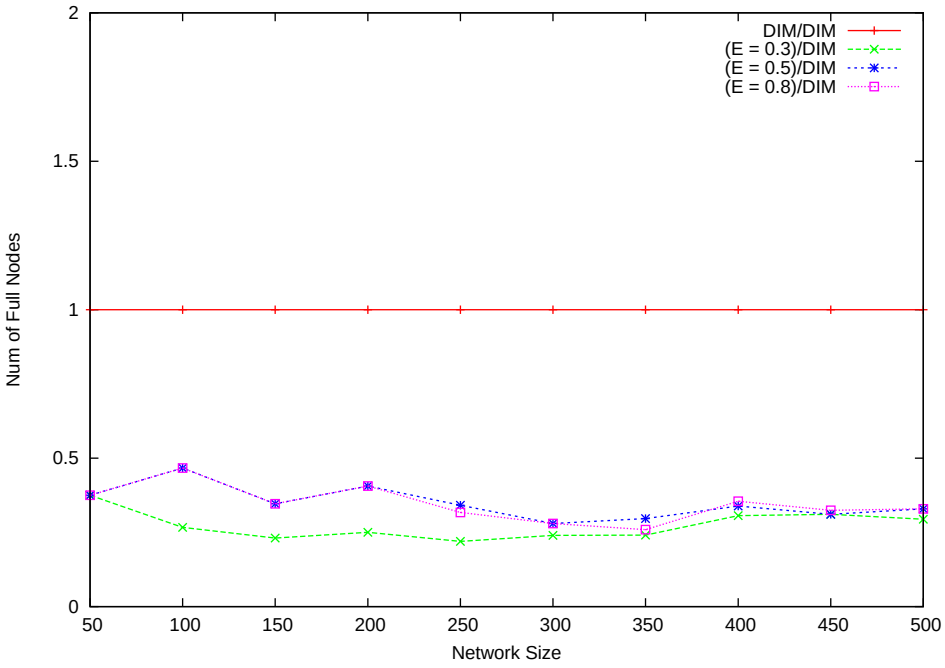


(b) 60% Multiple Storage Hotspots

Figure 21: ZS: Dropped Events for Various E Values

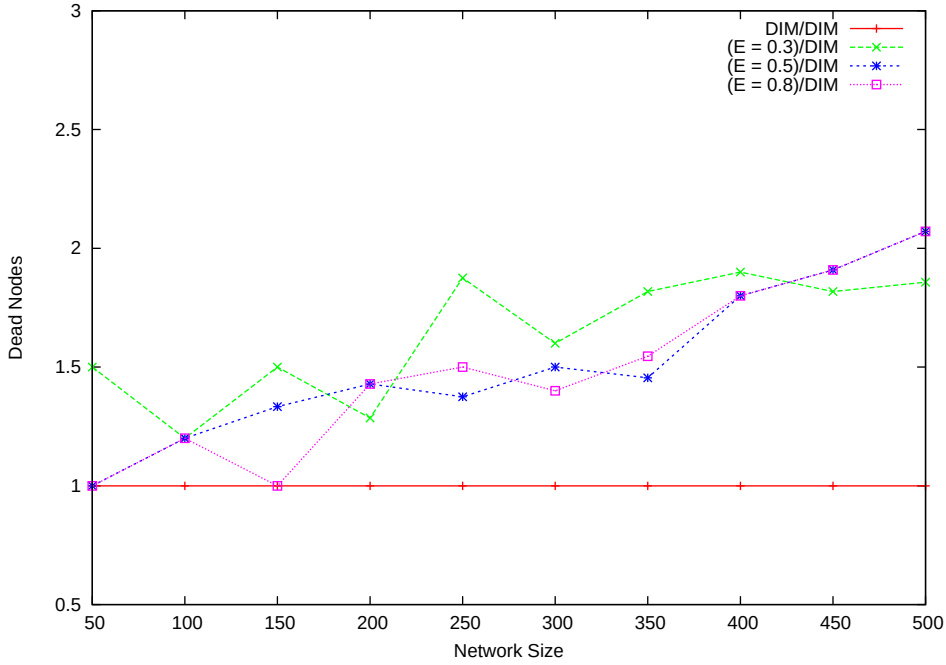


(a) 80% Single Storage Hotspot

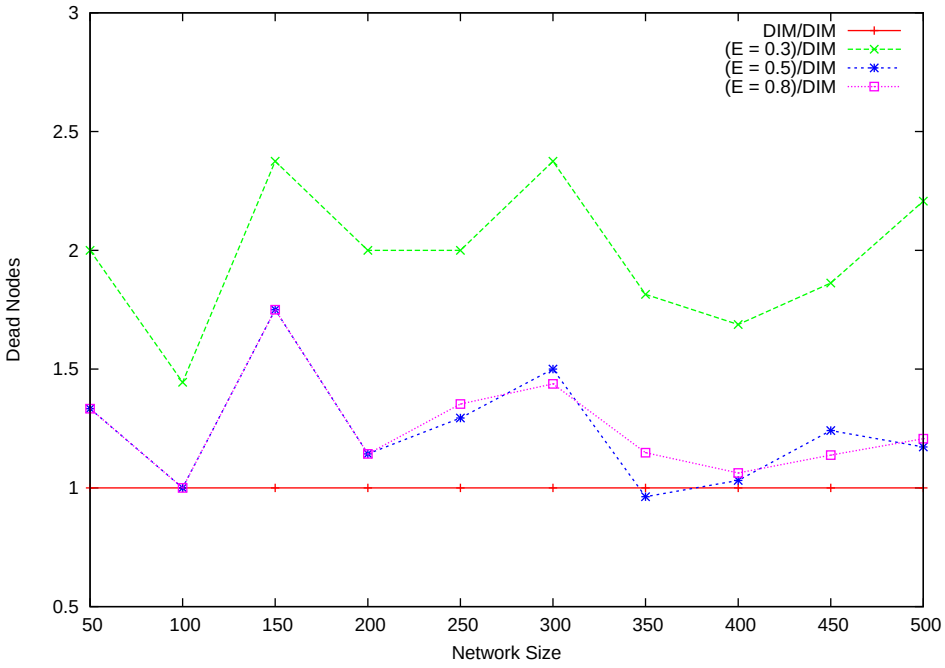


(b) 60% Multiple Storage Hotspots

Figure 22: ZS: Number of Full Nodes for Various E Values



(a) Dead Nodes for an 80% Single Storage Hotspot



(b) Dead Nodes for 60% Multiple Storage Hotspots

Figure 23: ZS: Dead Node Graphs for Various E Values

to deplete their energy. However, as node deaths for DIM are around 3% of the network size for single hotspots and 5% of the network size for multiple hotspots, the performance degradation imposed by MHZS (in terms of node deaths) is relatively small when compared the the network size.

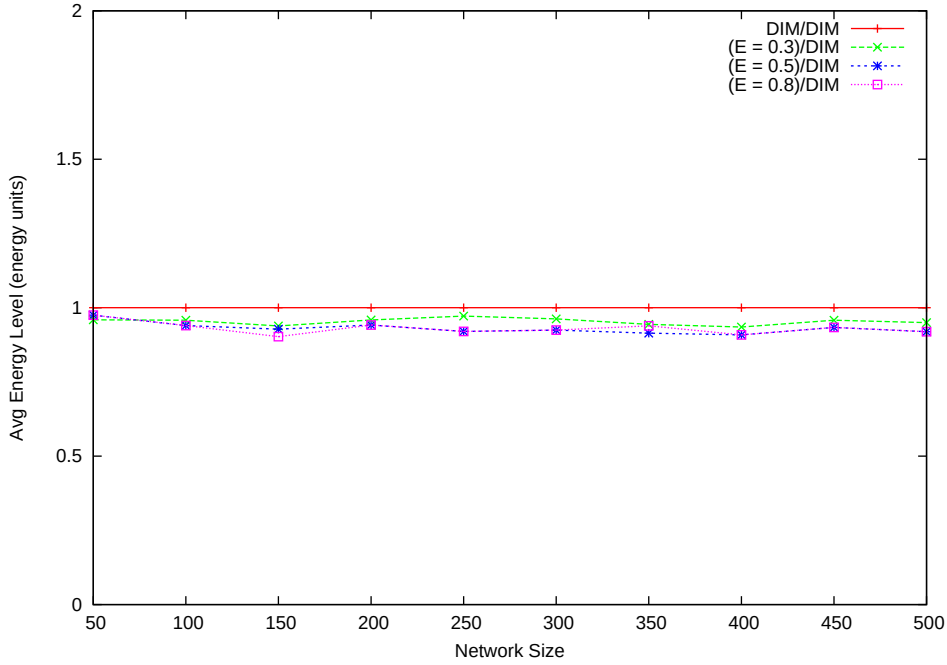
A similar observation can be seen in Figures 24(a) and 24(b) comparing the average node energy for MHZS, applying each of the three E values, for the cases 60% single and multiple hotspots, respectively. The figures show that the two MHZS versions with $E = 0.5$ and $E = 0.8$ exhibit almost the same performance. The first figure shows that the energy consumption overheads increase from 5% at the case of $E = 0.3$ to 10% at the other two cases. For multiple hotspots, the MHZS version with $E = 0.3$ reduces DIM's energy consumption overhead by around 9% per node as opposed to 15% for the other two versions (due to the collision effect). In general, increasing E to 0.3 slightly decreases the energy consumption load imposed on sensor nodes (by around 5% compared to DIM) for the case of single hotspots. This results from the fact that the number of times the ZS process is applied is higher than in the cases where E takes larger values.

In general, the above results show that the E parameter has an important effect on the ZS performance. The $E = 0.3$ version of MHZS performs better than the other two versions (that perform similarly). For single hotspots, the $E = 0.3$ MHZS version increases QoD by around 20% while improving the energy consumption by around 5% (compared to the other two MHZS versions with DIM as the base case).

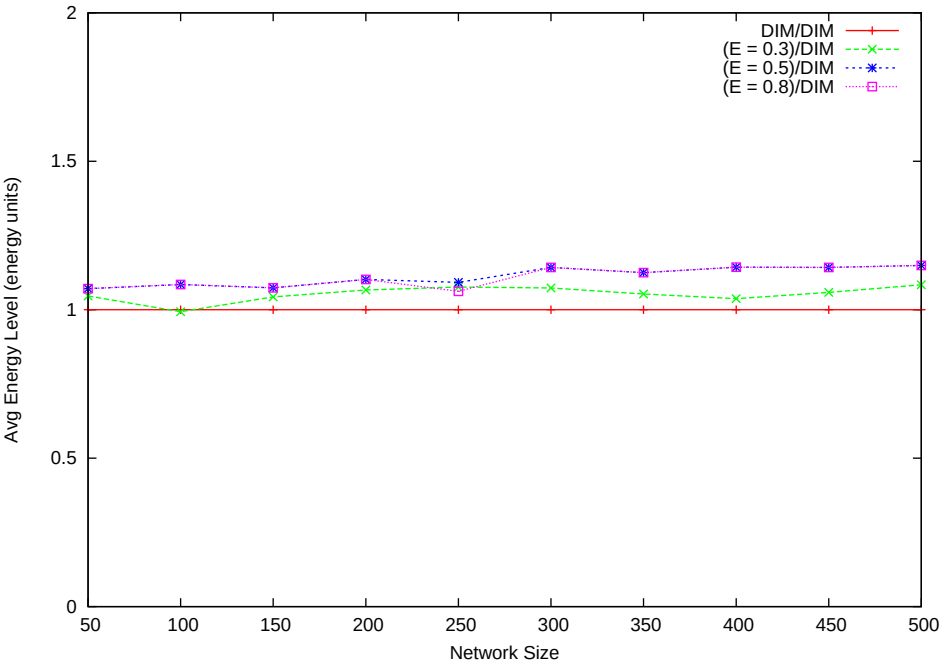
4.1.8.7 Effect of the Zone Share Count on MHZS Performance

In this subsection, we move on to study the effect of the zone share count on the MHZS performance. We will refer to this parameter by SC. This parameter controls the number of times a zone can be shared. Recall that our default value for the zone share count was 5. As discussed in Section 4.1.8.1, we study SC values ranging from 5 to 15.

We continue to concentrate on studying the effect of changing the threshold values on the MHZS scheme performance. The following three results compare the MHZS performance for the different values of the zone share count when facing both single and multiple hotspots. We conducted results for hotspots of sizes up to 80%.

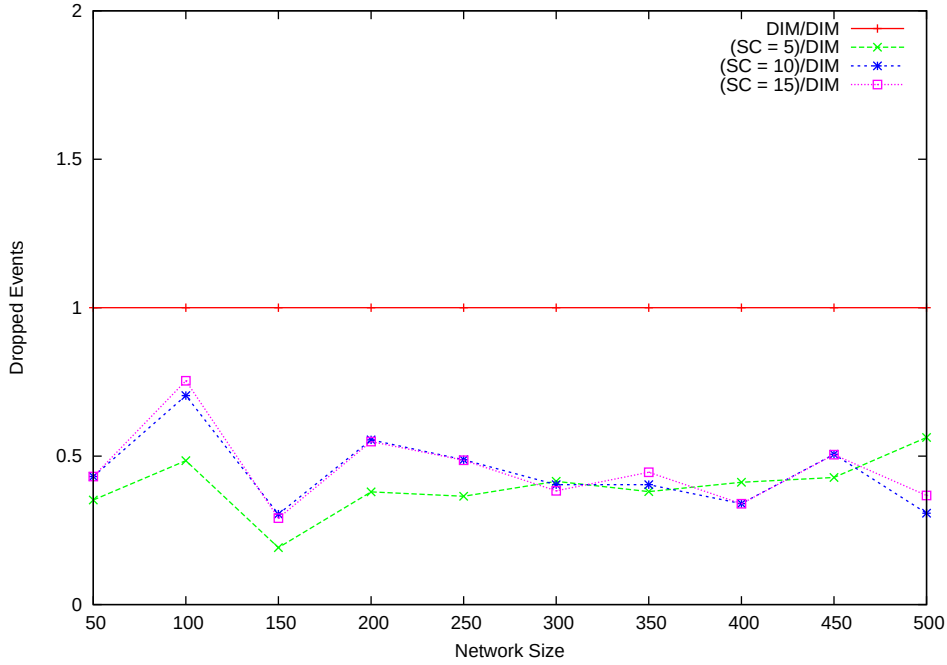


(a) Average Node Energy for a 60% Single Storage Hotspot

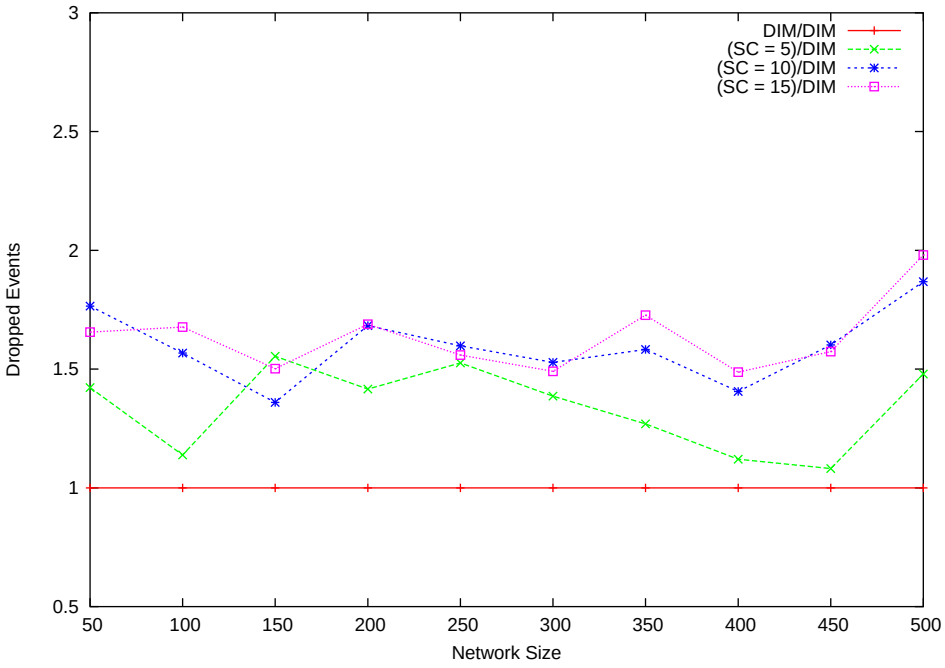


(b) Average Node Energy for 60% Multiple Storage Hotspots

Figure 24: ZS: Energy Consumption Graphs for Various E Values



(a) 80% Single Storage Hotspot



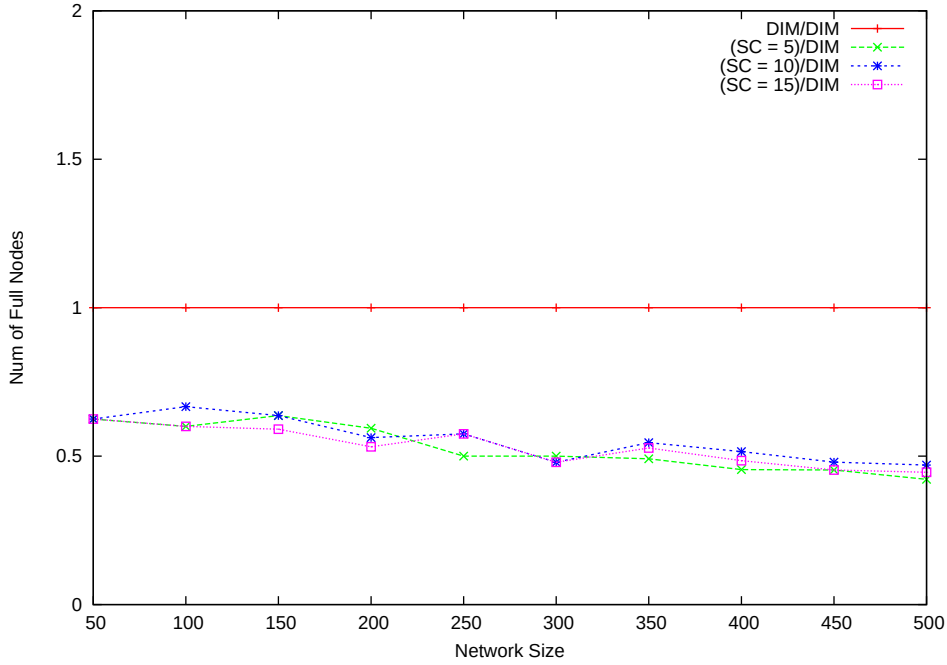
(b) 60% Multiple Storage Hotspots

Figure 25: ZS: Dropped Events for Various SC Values

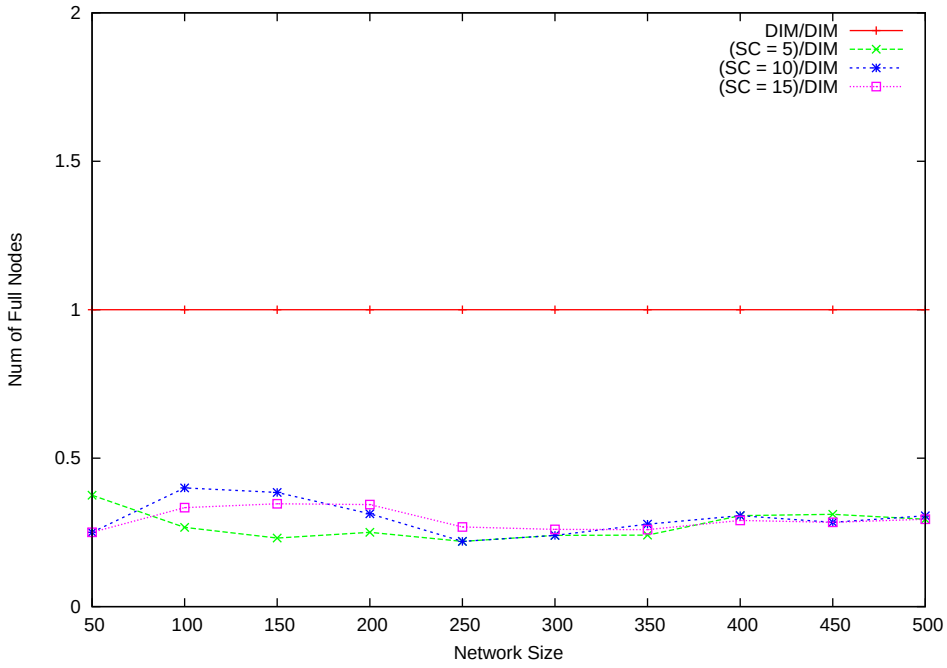
R1. QoD: Figures 25(a) and 25(b) compare the number of dropped events for MHZS, applying the three SC values, for the 80% single hotspot case and 60% multiple hotspots case, respectively. The first figure shows that, for the single hotspot case, the QoD performance of MHZS is very similar for all SC values ranging from 10 to 15. The QoD performance improvement scored by MHZS over DIM ranges from 50% for $SC = 10$ (or $SC = 15$) to 55% for $SC = 5$%. The important observation in this result is that the SC value does not highly affect the QoD achievements of the MHZS scheme. This result can be explained as follows. For the ZS process to be applied, all DMC inequalities need to be satisfied. Though increasing the SC value allows a hot zone to be traded a larger number of times, such a zone cannot be traded except when three sensor nodes satisfy the DMC. This purely depends on the loads of the sensor nodes, in terms of both storage and energy, throughout the network lifetime. Though we have changed the SC values in this experiment, each zone ends up by being traded an almost constant number of times. This mainly results in a similar performance for MHZS for the different SC values. We achieved similar performances for hotspots of sizes up to 80%.

R2. Load Balancing: Figures 26(a) and 26(b) compare the average node storage for the different MHZS versions for the 80% single hotspot case and 60% multiple hotspots case, respectively. The figures show that the two MHZS versions with $SC = 10$ and $SC = 15$ exhibit a similar performance, while setting $SC = 5$ decreases the number of full nodes by around 5% for both single and multiple hotspots. This justifies the QoD performances discussed in R1 by showing that the effect of SC on load balancing is limited. We achieved similar performances for hotspots of sizes up to 80%.

R3. Energy Consumption: Figures 27(a) and 27(b) compare the number of dead nodes for the different MHZS versions for the 80% single hotspot case and 60% multiple hotspots case, respectively. The first figure shows that increasing the SC value to 10 (or beyond) increases the number of dead nodes by around 15% compared to DIM (whose number of dead nodes is around 3% of the network size) for single hotspots. This is also strengthened by Figure 28(a) which shows that the same increase in the SC value decreases the average node energy by around 3%. This can be explained as follows. Increasing SC allows a hot zone to move for larger number of times. This results in involving a larger area (of sensors) in

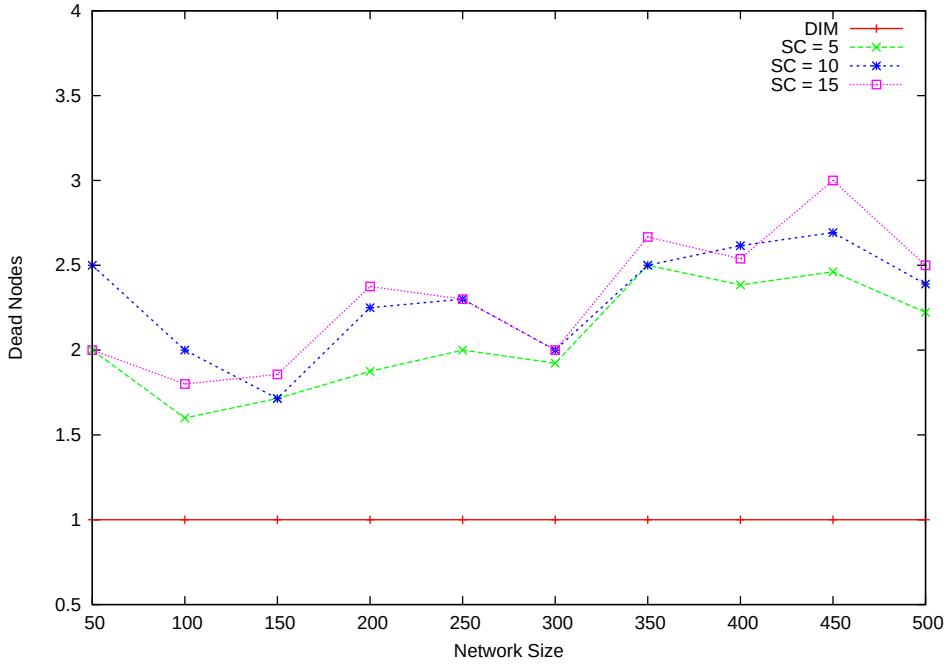


(a) 80% Single Storage Hotspot

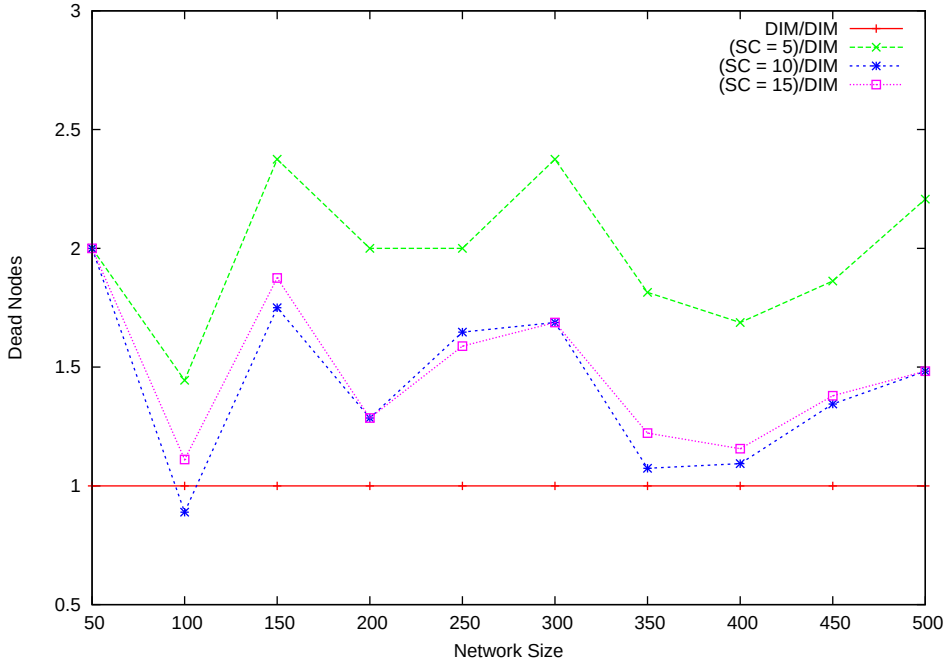


(b) 60% Multiple Storage Hotspot

Figure 26: ZS: Number of Full Nodes for Various SC Values



(a) Dead Nodes for a 60% Single Storage Hotspot



(b) Dead Nodes for 60% Multiple Storage Hotspots

Figure 27: ZS: Dead Node Graphs for Various SC Values

the hotspot storage. Thus, a larger number of sensors will be involved in sending/receiving packets. This results in a slight increase in the energy consumption overhead imposed on the individual sensor nodes in the network.

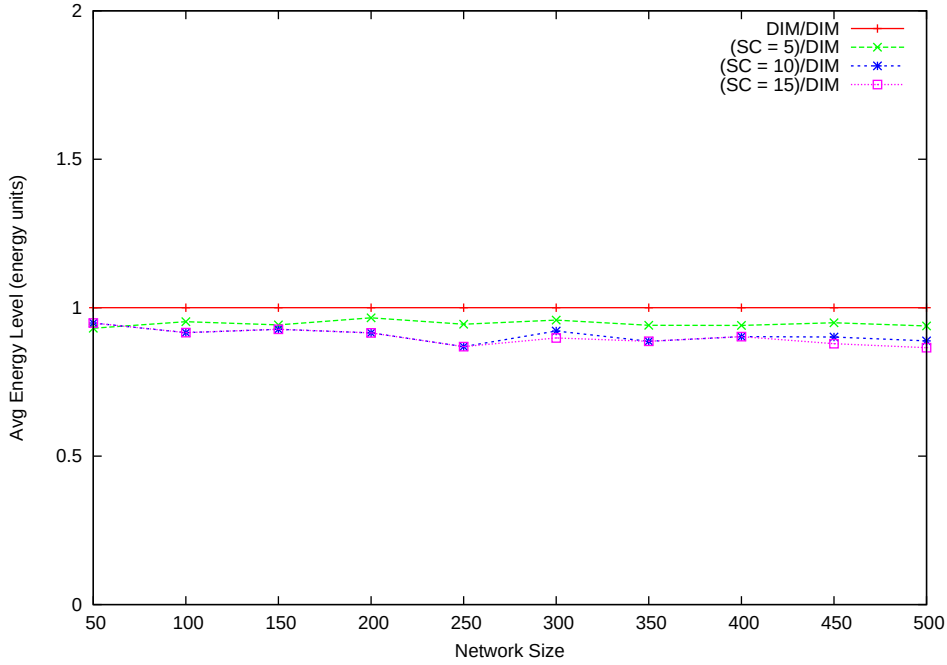
In conclusion, changing the value of the SC parameter does not highly affect neither QoD nor QoS. For $SC = 5$, QoD improvements are around 55% while energy consumption overhead is around 5%. When increasing the SC value to 10 or 15, QoD improvements are around 50% while energy consumption overheads are around 8% per node.

To summarize the sensitivity analysis results, Table 2 shows the effect of the different parameters on the MHZS performance for single storage hotspots (of sizes up to 80%).

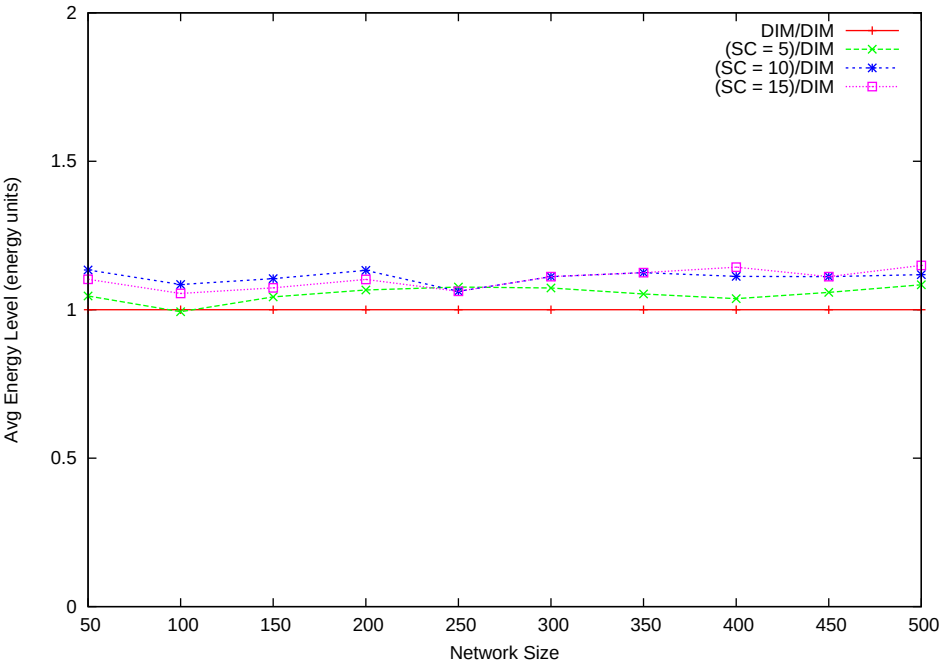
4.1.8.8 Discussion In this section, we conducted a complete study of the ZS scheme. We studied the ZS performance for single, multiple, and moving hotspots. Additionally, we studied the individual effect of the storage level threshold, the energy level threshold, and the zone share count on the ZS performance. Table 3 summarizes the ZS performance for the different storage hotspot types (of sizes up to 80%).

In general, SHZS can be considered as a better choice than MHZS. This is due to the fact that MHZS may cause further hotspot collisions in the case of multiple hotspots. This collision problem downgrades its QoD performance compared to DIM. This has the effect of limiting its usability for decomposing general storage hotspots. On the other hand, SHZS achieves moderate QoD improvements while imposing a small energy consumption overhead compared to DIM for all hotspot settings.

Now that we have completely presented and studied the performance of the ZS scheme, we move to present ZP/ZPR schemes aimed at locally detecting and decomposing query hotspots.



(a) Average Node Energy for an 80% Single Storage Hotspot



(b) Average Node Energy for 60% Multiple Storage Hotspots

Figure 28: ZS: Energy Consumption Graphs for Various SC Values

Table 2: Performance of the Different Versions of ZS (Relative to DIM) for Single Storage Hotspots

Hotspot Type	QoD Gains	QoS Losses
$[C = 1.5, E = 0.8, SC = 1]$	18%	3.7%
$[C = 2, E = 0.8, SC = 1]$	20%	3.5%
$[C = 3, E = 0.8, SC = 1]$	14%	3.2%
$[C = 2, E = 0.5, SC = 1]$	22%	3.5%
$[C = 2, E = 0.3, SC = 1]$	25%	3%
$[C = 2, E = 0.3, SC = 5]$	55%	5%
$[C = 1.5, E = 0.3, SC = 5]$	52%	8%
$[C = 3, E = 0.3, SC = 5]$	35%	3%
$[C = 2, E = 0.5, SC = 5]$	35%	10%
$[C = 2, E = 0.8, SC = 5]$	35%	10%
$[C = 2, E = 0.8, SC = 10]$	50%	8%
$[C = 2, E = 0.8, SC = 15]$	50%	8%

Table 3: ZS Performance (Relative to DIM) for Storage Hotspots

Hotspot Type	Scheme	QoD Gains	QoS Losses
Single Static Hotspots	SHZS	20%	3%
	MHZS	55%	5%
Multiple Static Hotspots	SHZS	20%	2%
	MHZS	−30%	−5%
Moving Hotspots	SHZS	15%	2%
	MHZS	20%	3%

4.2 LOCAL DETECTION AND DECOMPOSITION OF QUERY HOTSPOTS

Another major problem in DCS schemes, and specifically in DIM, is that of *query hotspots*. Query hotspots may occur in DIM if the sensors are not uniformly distributed. Query hotspots may also occur in the case of a skewed query workload. In both cases, relatively many queries are asking for data stored in a relatively small number of the sensors. Thus, a query hotspot is formed in the geographic area of these sensors. For example, if there was only one sensor on one side of the first bisection (i.e., in one of the two subtrees of the root of the k-d tree), then half of the query load would be accessing this sensor (assuming a uniformly distributed query workload). The presence of a query hotspot affects both the QoD and the QoS of the sensor network. Certainly, it is not desirable to have a storage scheme whose QoD and QoS guarantees rest on assuming a uniform distribution of both, sensor locations and query load.

In this section, we propose two schemes locally solving the query hotspots problem in the DIM scheme: *Zone Partitioning (ZP) and Zone Partial Replication (ZPR)*. ZP considers a node, having a frequently accessed zone, compared to its neighbors' zones, a good indication of a query hotspot. A reasonable solution would be to force this node to split its owned zone with one of its less-accessed neighbors. For the case where the access frequencies are not homogeneous among the subranges of the frequently accessed zone, we present a second algorithm, ZPR, to replicate the readings of the highly accessed subranges among a larger number of sensor nodes to reduce the total number of queries accessing the hotspot area.

Experimental evaluation shows that the main advantages of applying ZP/ZPR on top of DIM are:

- Improving *QoD* by distributing the hotspot events (readings/queries) among a larger number of sensors. Improvements ranged from 25% over DIM for single query hotspots to 15% over DIM for multiple static query hotspots.
- Increasing the *energy savings* by balancing energy consumption among sensor nodes. Energy consumption overhead additionally imposed by ZP/ZPR ranged from 7% (per node) over DIM for single hotspots to 5% for multiple static hotspots.

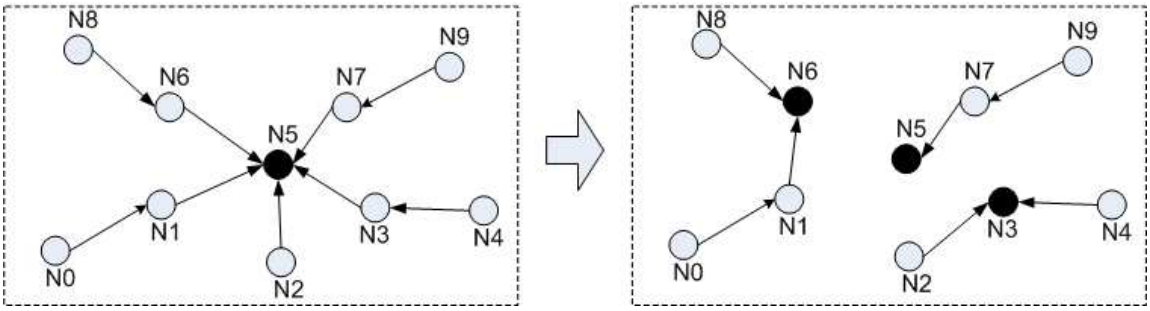


Figure 29: ZP Example

This was valid for hotspots of sizes ranging from 40% to 80%.

The following two subsections describe our ZP and ZPR schemes, respectively.

4.2.1 Zone Partitioning (ZP)

We now describe ZP, which is the first algorithm we present for decomposing query hotspots in DIM. As discussed above, a query hotspot represents a skewness of the query load toward a given path of the k-d tree. Thus, ZP is based on continuously checking for such skewness and trying to rebalance it when it is detected at any subset of sensors. We first illustrate the basic ZP idea using a simple example.

4.2.1.1 Illustrative Example Figure 29 shows a typical scenario explaining how ZP works. In the Figure, a circle represents a sensor node and an arrow represents a hop on a query path. A black node is one that is storing data accessed by queries, while a white node is accessed by no queries in the query load. In the sensor network on the left hand side, queries sent from sensor nodes N0, N2, N4, N8, and N9 require data falling in the storage range responsibility of N5. After knowing that none of its neighbors is accessed by queries, N5 determines that it is responsible for a hot range of attribute values, thus, falling in a query hotspot. Subsequently, node N5 partitions the responsibility of its original zone between itself, node N3, and node N6. After the zone partitioning, queries take the paths described in the right hand side of the Figure. Note that this zone partitioning is only logical,

hence, nodes N3 and N6 keep their original zones (i.e., addresses) and each of them takes responsibility of another zone whose code value is different from its binary address value. Note that Nodes N3 and N6 must have enough memory space to store the newly received readings belonging to the hot partition passed by N5.

In the above ZP process, we call the node that passes (donates) responsibility of part of its hot zone **the donor** (N5 in Figure 29). The neighbors that take parts of the partitioned zone are called **the receivers** (N3 and N6 in Figure 29).

Now that we have presented the basic ZP idea, we present the actual ZP components in the following subsections.

4.2.1.2 Local Detection of Query Hotspots

In order to locally detect hotspots, each node keeps track of the frequency of accesses of its readings. The tuple representing every stored event is appended by a counter called the $access\_frequency$. This counter represents the number of queries accessing such reading over a given time period (window), w . The node resets all counters at the start of w and increments the counter associated with a reading every time it receives a query requesting this reading during w.

Each sensor node continuously computes the Average Access Frequency, $AAF(Z_k)$ of each of its zones Z_k, which is the average of the access frequencies of readings belonging to zone Z_k. Note that, in general, a node can be responsible for more than one zone. At this point, let us assume that readings falling in each zone will be uniformly accessed. In case a node x realizes that

$$\frac{AAF(Z_i)}{AAF(Z_j)} > threshold_1 \ \forall Z_j \ stored \ in \ x$$

that is, zone Z_i is much more accessed than all other zones, x considers this as an indication of a query hotspot that it experiences. Thus, x decides to split the hot zone Z_i into two partitions: Z_{i1} and Z_{i2} in a way that keeps $AAF(Z_{i1})$ and $AAF(Z_{i1})$ almost equal. Note that each of the zone addresses of Z_{i1} and Z_{i2} should be one bit longer than the zone address of Z_i, as the former zones are children of the latter in the k-d tree. Then, x keeps one of the partitions and passes the other one to a selected node of its neighbors, namely the *receiver*. We call the partition that will be passed to a receiver, the traded zone T. Note that the ratio $threshold_1$ should be larger than 1 to guarantee the real existence of a hotspot. After

electing Z_i from its zones, x compares $AAF(Z_i)$ to the $AAFs$ of its neighbors as we discuss in the next subsection.

4.2.1.3 The Partitioning Criterion (PC)

We now present the Partitioning Criterion (PC), which is a set of inequalities to be *locally* applied by the donor to select the best candidate among its neighbors to be a receiver. The PC inequalities relate the loads, as well as the energy levels, of the donor with those of its neighbors. In these inequalities, we express the traded zone, T, in terms of the number of traded messages (Note that an event represents one message). An energy unit represents the amount needed to send one message. The fraction r_e is the amount of energy consumed in receiving one message (if less than one energy unit). We express the total storage capacity of a sensor node by S. By l_x, we mean the storage load of node x, while e_x is meant to be the energy level of node x:

$$T + l_{receiver} \leq S \tag{4.6}$$

$$\frac{T}{e_{donor}} \leq E_1 \tag{4.7}$$

$$\frac{T * r_e}{e_{receiver}} \leq E_2 \tag{4.8}$$

$$\frac{AAF(donor)}{AAF(receiver)} \geq Q_1 \tag{4.9}$$

where E_i's and Q_j are constants representing energy ratio and average access frequency thresholds, respectively.

Equation (4.6) represents a *Storage Safety Requirement*. It states that the sum of the pre-partitioning load of the receiver and the traded zone size should be less than the storage capacity of the receiver. Such constraint is needed in order to guarantee that the receiver will be able to store all the traded zone readings. Equations (4.7) and (4.8) represent *Energy Safety Requirements*. Equation (4.7) states that the ratio of the energy consumed by the donor in sending the traded zone to the available donor energy is less than a given threshold. Equation (4.8) states that the ratio of the energy consumed by the receiver in receiving the traded zone to the available receiver energy is less than a given threshold. These inequalities are needed to make sure that the energy amount consumed in the partitioning process will not cause the death, or the approach to death, of one or more of the nodes involved in the

```
ZP (Current Node C, List of Neighbors N, Threshold p)
Begin
 1. If (AAF (Zi) / AAF (Zj) > p for all Zj in C
 2     If (AAF (Zi) / AAF (n) > p) for all nodes in N
 3.          Sort list N ascendingly based on AAF
 4.          For every node n in N
 5.               Send RTP to n (containing AAF (C), energy (C), and size (T))
 6.               If (ATP received from n (this means n successfully applied PC))
 7.                    m = n
 8.                    Break
 9.               End
10.          Send traded zone T to m
11.    End
12. End
End
```

Figure 30: Zone Partitioning Algorithm

partitioning process. The values of the E_i thresholds should be relatively small, e.g., less than 0.5. Finally, equation (4.9) represents the *Access Frequency Requirement*. It relates the average access frequencies of both, the donor and the receiver, and makes sure that the ratio of the first to the second is greater than a threshold Q_1. This is needed in order to guarantee that the donor is really falling within a query hotspot, as well as to be able to choose the best receiver to partition the hot zone with. The value of Q_1 should be relatively larger, e.g., greater than 2.

Note that the values of the different PC thresholds depend on the actual sensor network application and the relative importance of energy to that of data in the network. In an investigation network, where the value of data is relatively more than that of energy, the Q thresholds should take small values while the E thresholds should be assigned large values. However, in a search/discovery network, where energy is more valuable than data, the reverse is true. That is, E thresholds should take small values while Q thresholds should take large values.

To be able to apply the *PC*, the donor is supposed to know load information, in terms of in terms of storage, energy and average query frequency, of its neighbors. The periodic

messages exchanged between neighbors to maintain the DIM structure, as well as insertion and query messages, can be piggybacked with such information. A sensor node experiencing a high frequency of accesses uses such neighbors' information to select the best receiver among them. This can be done by selecting the node that minimizes the left hand side of equation (3) while maximizing the left hand side of equation (4). Upon selecting a receiver, the donor sends a *Request to Partition (RTP)* to such receiver. In case the receiver accepts this request, it replies to the donor with an *Accept to Partition (ATP)*. Figure 30 shows the ZP algorithm periodically applied by each node in the network. Note that the RTP and ATP could either be piggybacked on other messages, or sent explicitly. The overhead of the load information exchange messages and that of the hand shaking messages are small compared to that of the traded zone passing messages as the formers take can be piggybacked on the structure maintaining messages that were presented in the original DIM scheme.

We now illustrate how the decomposition of large query hotspots, arising in more than one sensor node, takes place. Given the PC inequalities, nodes near the center of the hotspot cannot find any receiver to partition load with as all their neighbors are falling in the hotspot. Therefore, the hotspot decomposition starts from the *hotspot borders*. Each of nodes on the borders of the hotspot partitions its storage with one of its less loaded neighbors. These border nodes subsequently cease belonging to the hotspot. A new set of nodes now fall on the hotspot borders. These nodes start partitioning their storage with their less loaded neighbors. The process continues in an iterative fashion, decomposing the hotspot more and more at each iteration, until nodes in the center of the hotspot become normally loaded. This signifies the complete decomposition of the hotspot.

4.2.1.4 GPSR Modifications We now discuss the changes introduced to the GPSR algorithm to account for ZP. Based the above ZP algorithm, a receiver can re-apply the PC to partition a previously traded zone. The process can be applied more than once. At each time, a smaller hot sub-zone is moved further away from its original node. In case of k subsequent partitioning times, keeping GPSR with no changes will involve the original donor in all insertions and queries concerning any of the k traded zones. This overhead would be proportional to the total number of hops a zone is traded. To reduce this overhead, we

augmented GPSR to recognize that a zone has been traded and moved away from its original storage sensor.

For such purpose, each node maintains a *Traded Zones List (TZL)* containing three entries: zone address, original donor, and final receiver. Upon ZP, the donor sends the traded zone address, its ID and the receiver's ID to all its neighbors. Thus, each node will be aware of zones traded by its neighbors. In case a receiver x repartitions a traded zone z into z_1 and z_2, x becomes the new donor of z. Therefore, x sends the new sub-zone address (assume it z_2), the original donor of z, as well as the new receiver of z_2, to all of its neighbors. Note that x gets the original donor of z from the entry of z in its TZL. Each of the neighbors, upon receiving such entry, should check its TZL for a previous entry for z, or any of its parent zones. In case of finding z's entry, the node overwrites its zone address by z_1. Then, the node adds a new entry for z_2 in its TZL. At the end, it forwards the z_2 entry to all its neighbors.

As we are constraining each node to send traded zone information only to its neighbors, it is easy to prove that an entry for a p-times traded sub-zone will be present in the TZLs of nodes falling on a path of $\theta(p)$ hops away from its final destination. Based on the definition of a query hotspot, the number of zones that will be originally falling in the hotspot will be very small. Thus, keeping the TZL represents a small storage overhead on all sensor nodes. Additionally, the computation overhead imposed on any node for searching its TZL for an entry is $O(\log p)$, which is relatively small.

Using the TZL concept, GPSR is changed as follows. Each message sent by GPSR is appended by a *dest_changed* bit flag and a *dest*, which is a node address entry. Originally, *dest_changed* is set by the message sender to 0. In routing a message (an reading or a query), a node x first checks the *dest_changed* flag. In case it is 1, x uses the *dest* variable as an explicit destination for the message and uses the original GPSR to select the next hop for the message toward *dest*. Otherwise, x forms the message destination address z using the original DIM event to bit-code mapping and then checks its TZL for an entry whose zone address (or left most significant string of the zone address) is identical to z. In case z is found in x's TZL, x sets the *dest_changed* to 1 and *dest* to the new destination of z found in the z's TZL entry. The original GPSR is then used to send the message to *dest*. In case no

```
GPSR_ZP (Current Node C, Traded Zone List TZL, Destination D, dest_changed flag)
Begin
        1. If (dest_changed == false)
        2.      If (D is found in TZL as original donor of TZL entry e)
        3.              Dest_changed = true
        4.              D = final destination (e)
        5.      End
        6. End
        7. Apply basic GPSR algorithm to D
End
```

Figure 31: Modified GPSR Algorithm for ZP

entry is found for z, GPSR uses z as the message destination. Figure 31 shows the modified version of the GPSR algorithm run by each node in the network.

Using the modified GPSR algorithm, TZL search for any p-times traded zone T occurs only once by a node which is $\theta(p)$ hops away from the final destination of T. As soon as such node updates the *dest_changed* and *dest* fields in the message, *dest* is explicitly used by GPSR in the subsequent nodes without searching the list. This has the effect of minimizing the overhead of both, energy consumption and computational load, imposed on nodes falling in the query hotspot.

4.2.1.5 Coalescing Process We now discuss how to cope with query load distribution changes. Based on ZP, the receiver continues to keep track of the accesses of the traded zones. In case any of such zones is not accessed for a complete time window, d, this is considered as an indication that the hotspot has stopped to exist (or may have moved to another location). At such point, the receiver transfers the responsibility of the received zone back to its original owner. We call this the *coalescing process*. All neighboring nodes drop the traded zone entry from their TZLs. Further readings belonging to, as well as queries asking for, that zone are directed to the original donor based on the original DIM and GPSR schemes. In such process, the receiver only sends recent readings belonging to the previously

received zone to the original donor (readings inserted during d). Thus, the coalescing process is considered much cheaper than the partitioning process. In case d is large enough, we can guarantee that the hotspot would not be formed again in the future and that the coalescing process is not causing a loss in the QoD provided by the sensor network as the receiver drops the readings inserted before d. Using such coalescing process, partitioning *oscillations*, where the responsibility of storing the zone keeps going back and forth between its original owner in DIM and other neighboring sensor nodes due to hotspot changes, can be avoided.

From the above, ZP decomposes the query hotspot by continuously partitioning its storage load with its neighboring sensors starting from the hotspot borders and moving toward its center. However, in ZP, we assume that the access frequency is uniform among the subranges of the hot zone. How about in case it is skewed toward a narrow subrange? The following section addresses this question by presenting our second query hotspot decomposition algorithm.

4.2.2 Zone Partial Replication (ZPR)

We now present ZPR, which is our second query hotspot decomposition algorithm. ZPR mainly deals with the case where a sensor x falls in a query hotspot and a large percentage of the queries accessing x target a small subrange of x's attribute range responsibility. In such a case, a fairly limited number of readings are accessed. ZPR decomposes the query hotspot by replicating such readings in neighboring sensors. ZPR is meant to work in parallel with ZP such that the combination of ZP/ZPR can efficiently handle the decomposition task of query hotspots of different sizes.

4.2.2.1 Additional PC Requirements First, let us show how a node decides whether to apply ZP or ZPR. To do this, it is necessary to enhance the PC (the Partitioning Criterion already presented in Section 4.2.1.3) by additional conditions that help the node falling in a query hotspot in determining the algorithm to be used for hotspot decomposition.

Specifically, we add two more *Access Frequency Requirement* inequalities to the PC.

$$\frac{AAF(partialsubrange)}{AAF(totalzonerange)} \geq Q_2 \tag{4.10}$$

$$\frac{size(partialsubrange)}{size(totalzonerange)} \leq Q_3 \tag{4.11}$$

Equation (4.10) relates the AAF of given subrange of the attribute ranges of the hot zone to the AAF of the entire hot zone range. Such equation indicates that almost all queries targeting the hot zone are basically asking for readings falling within the hot subrange of attribute values. The threshold, Q_2 should take values close to 1, such as 0.7 to 0.9. On the other hand, equation (4.11) makes sure that the size of the hot sub-zone is fairly small compared to the total hot zone size. It is clear that the threshold, Q_3, should take values close to 0, such as 0.2 or less.

A node falling within the query hotspot first tries to satisfy all 6 PC inequalities. In case it succeeds in this, then the node chooses to apply ZPR. If the node is only able to satisfy the first 4 PC inequalities, it proceeds in applying ZP. In case ZPR is chosen to be applied, the donor sends the hot sub-zone to all its direct neighbors. Each of these neighbors, upon receiving the hot sub-zone, inserts an entry for such sub-zone in its TZL. The entry is represented by the tuple (sub-zone code, donor, self address). In other words, the node indicates itself as the receiver of such sub-zone. Note that we selected to follow the same TZL technique as the one used in ZP in order to reduce special cases.

Based on the above, no further changes need to be imposed to the modified GPSR to route queries in ZPR. Upon routing a query asking for a given zone, the node checks its TZL first. In case an entry is found for such zone with the receiver address is equal to the node's self address, the reading is simply looked up in the node's storage. When a query is answered by one of the receivers of the replicated hot zone, such receiver broadcasts the hot zone to all its neighbors. In other words, whenever a replicated hot zone is used to answer queries, we enforce such zone to be re-replicated one hop further from the original hot zone owner. Neighbors receiving such broadcast store the replicated hot zone in case of space availability. Note that, in case a query asks for a portion of the hot zone and the query is answered by replica node, such node only broadcasts that portion of the hot-zone. This should not happen frequently as the hot zone is small enough by definition.

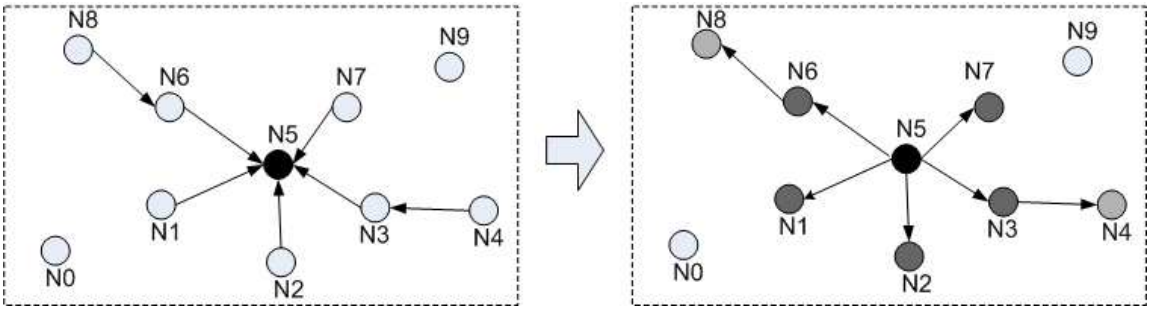


Figure 32: ZPR Example

Figure 32 shows a ZPR example. In the left hand side, node N5 (in black) is accessed by queries sent from nodes N1, N2, N4, N7, and N8. By applying ZPR, N5 sends the hot sub-zone readings to all its direct neighbors (nodes in dark gray, i.e., nodes N1, N2, N3, N6, and N7). Thus, queries initiated by N1, N2, and N7 are answered from the initiating nodes' caches. For queries initiated at N4 and N8, the results are first provided by N3 and N6, respectively. Each of the nodes N3 and N6, upon realizing that the replicated hot sub-zone was used in answering such queries, forward such sub-zone to N4 and N8 (filled with light gray), respectively.

4.2.2.2 ZPR Handling of Insertions An interesting question is how ZPR handles reading insertions. Recall that when any node x receives a message (in this case, a reading insertion), it applies the modified GPSR presented above. Thus, x first checks its TZL for the destination zone (i.e., node address) of such reading. In case ZPR was applied to the reading's zone z, x will find an entry for z with x's address as z's final destination. In such case, x inserts the new reading in its cache and proceeds with sending the reading to its original storage sensor as determined by the DIM scheme. Upon receiving a newly inserted reading, the original storage sensor re-initiates ZPR to propagate such reading to all neighbors having copies of z.

It is worth mentioning that ZPR may encounter some inconsistencies in the answers of some simultaneously posed queries. Consider in Figure 32, a new hotspot event generated by node N7. At the same time, two queries are generated by N4 and N8. By the time the

new reading arrives at N5, N8's query would have been already answered by N6. However, N5 updates N3 with the newly inserted event before N3 replies at N4 with the query result, therefore, the new reading will be included in such result. In order to decrease this effect, we bound the number of hops a zone can be replicated away from its original storage sensor to a limited number of hops.

4.2.3 ZP/ZPR Implementation Overhead

As the ZP and the ZPR schemes cope with different hotspot scenarios, they are intended to work in combination to efficiently decompose a wider range of query hotspots. In this subsection, we will discuss the implementation overhead imposed by the ZP/ZPR mix on sensor nodes. Similar to the ZS overhead, ZP/ZPR impose two types of overheads: the *processing overhead* and the *memory overhead*. We discuss these overheads below.

The processing overheads consist of the PC evaluation and the TZL search overheads. Due to the similarity between the PC and the ZS's DMC, the overhead of applying the PC inequalities is negligible exactly as that of applying the DMC inequalities. As the PC is periodically applied by each node a number of times in the order of the direct neighbors of that node, which is in the worst case $\theta(n)$, the overall overhead imposed by the PC on each sensor node is negligible. The second processing overhead, which is the TZL search overhead, is also similar to that incurred by the ZS's SZL search. As each node sends the information about traded zones only to its neighbors, a k-times traded zone will be at most present in the TZLs of nodes of $\theta(k)$ hops away from its final destination. Thus, the TZL search overhead that will be encountered by each sensor node will be at most in the order of $\theta(\log n)$. As the TZL search occurs only once in the furthest node from the destination (and containing the traded zone information), this search does not add a considerable energy consumption overhead on individual sensor nodes as it results in a maximum of $\theta(n)$ energy units to be consumed by all nodes involved in routing any event.

We now move on to consider the memory overhead imposed by the ZP/ZPR schemes on each sensor node. This overhead results from two sources: the PC code and the SZL. Considering the PC code, it mainly consists of a for loop spanning the neighbors of each

node and having the six inequalities applied for each of these neighbors to decide whether a zone partitioning or a zone partial replication is needed. Additionally, the AQF and energy status counters have to be kept for each of the neighbors of the node. Our PC implementation takes less than 50 lines of C++ code. Once run, the program requires around 1.2 KBytes of memory. This imposes a relatively small memory overhead on sensor nodes. Concerning the TZL size, our experimental study has showed that it is very small and does not impose a considerable storage burden on the sensor nodes.

4.2.4 ZP/ZPR Experimental Evaluation

In this section, we study the performance of the ZP/ZPR scheme when for the different types of query hotspots. We first conduct a sensitivity analysis to determine the effect of each of the parameters on the overall ZP/ZPR performance (Section 4.2.4.1). Based on the analysis results, we set default values for the different parameters. Using the default values, we compare the performance of the schemes against single static (Section 4.2.4.2), multiple static (Section 4.2.4.3), and moving query hotspots (Section 4.2.4.4).

Throughout this section, the node storage capacity is equal to 30 readings and the node initial energy capacity is equal to 70 units. Therefore, a *full sensor node* is defined to be a sensor node having 30 readings in its cache. Similarly, a node is depleted (and consequently considered dead) as soon as it consumes 70 energy units. Once a node is dead, all readings stored in this node are considered lost. Based on the DIM scheme, the storage responsibility (a subset of the attribute range) of the dead node is assigned to one of its direct neighbors. Recall that we define an event to be either a reading or a query.

For each of our experiments, we study three aspects: QoD (R1), load balancing (R2), and energy consumption (R3). For the QoD, we study the number of dropped events (readings/queries) and the average node storage. We refer to the percentage of QoD improvement to be the percentage of decrease in event drops. For the load balancing, we study the number of full nodes. As for energy consumption, we study the average node energy and the number of dead nodes. The average node energy is the one that defines the improvement or the downgrading in the energy consumption performance. To be statistically significant,

we conducted 5 simulation runs for each of the experiments and taken the average of values across all runs.

For each of the hotspot types, we conducted experiments on different hotspot sizes ranging from 20% to 80%. Unless otherwise stated, performing well on the large hotspot sizes, i.e., $[60\%, 80\%]$, implies a good performance on the moderate sized hotspots, i.e., $[40\%, 60\%]$. In most of the cases, the performance burden imposed to the network by small hotspots, i.e., hotspots less than 40%, does not justify the cost paid to detect and decompose the hotspots.

The main learned lessons from our experimental evaluation of ZP/ZPR can be summarized in the following points:

1. For single query hotspots, QoD improvements of ZP/ZPR are around 25% over DIM, while energy consumption overhead are around 7% over DIM.
2. For multiple hotspots, QoD improvements are around 20% over DIM, while energy consumption overhead is around 5% over DIM.
3. For moving hotspots, ZP/ZPR causes additional collisions in the network. It decreases DIM's QoD by around 300%. In terms of energy consumption, ZP/ZPR performs 25% better than DIM.
4. Setting the energy threshold E_i and the average access frequency threshold Q_1 to 0.3 and 5%, respectively, achieves boosts ZP/ZPR performance.
5. The actual values of Q_2, Q_3, and $threshold_1$ do not highly affect the ZP/ZPR performance.

4.2.4.1 Sensitivity Analysis We start by studying the effect of the different parameters on the ZP/ZPR performance, namely the energy thresholds (E_i) and the average access frequency thresholds (Q_i and $threshold_1$).

When tuning any of the parameters, we assigned the rest of the parameters to their default values. As for our default parameter values, we selected a value of 2 for $threshold_1$. Concerning the PC parameters, we chose a value of 0.3 for the E_1 and E_2 constants (inequalities 4.7 and 4.8, respectively). For the access frequency inequalities, we set Q_1 to 5 (inequality 4.9), Q_2 to 0.8 (inequality 4.10), and Q_3 to 0.2 (inequality 4.11). Note that the

initial selection of these values is a result of the role of each of them in the PC (as discussed in Section 4.2.1.3).

For the energy threshold E_i, we studied values between 0.3 and 0.8, exactly as we did with the E parameter of the DMC. The results of our study are highly similar to the parameter tuning results of the ZS scheme. This is a direct result of the high similarity between the hotspot detection and decomposition technique of the ZS and the ZP schemes. By having closer look at both the DMC and the PC, one can realize that the energy related constraints in both schemes have the same goal, which is to avoid the trade of any zone, and thus the movement of its storage responsibility, unless the nodes participating in the trade possess enough energy to fulfill this movement. Although one may think that the nature of storage and query hotspots are different, the underlying hotspot decomposition technique in both schemes is the same, as it is based on moving zones away from the hotspot area. For single hotspots, QoD improvements were around 25% for $E = 0.3$. This comes with an energy consumption overhead of 7%. QoD improvements dropped to 20% for $E = 0.5$ or $E = 0.8$. This comes with an energy consumption overhead of around 10%. Thus, the optimal E_i value for ZP should be around 0.3, exactly as that of ZS.

For the average access frequency threshold, our main focus was on the Q_1 parameter responsible of determining whether a zone is hot compared to its neighboring zones or no. We tested Q_1 values ranging from 2 to 10. QoD improvements were the most for $Q_1 = 5$ and dropped by around 2% to 3% for smaller and larger Q_1 values. As for energy consumption overheads, the $Q_1 = 5$ version continued to perform the best with an improvement of around 1% to 2%. The interesting observation was that the change of the Q_1 parameter values does not highly affect the performance of ZP, nor its hotspot load balancing capability, for the different query hotspot settings. This observation can be explained by the types of hotspots we tested. Recall that our hotspots were of the form of a large number of queries accessing a small range of attributes. This means that a lot of queries will be accessing a small number of nodes. Therefore, the nodes surrounding these hot nodes will mainly be of notably less access ratios. Consequently, the average access frequency criterion of the PC will quickly be triggered without highly depending on the value of the Q_1 parameter. Even if the increase in the Q_1 value results in some delay in applying the zone partitioning process,

this does not highly affect the performance of the scheme. This lets the PC energy criteria become the most dominating ones controlling whether the PC would be applied or no. For general hotspots, we believe a value of around 5 should be an optimal one for most of the hotspot cases. Individual hotspot types should be dealt with on a case by case basis by the administrators of the given sensor network.

For the rest of the average access frequency parameters, Q_2 and Q_3, their main responsibility is to trigger the replication functionality of the ZPR scheme. The importance of these parameters comes into play when many of the hotspots imposed to the sensor network are very narrow and spanning a very small subrange of the attribute ranges. As in most of our simulations, this was not the case, the change in the values of these parameters did not have a high effect on the ZP/ZPR performance. In real life, we believe that most of the hotspot setting will not meet the ZPR triggering requirements. The only query type for which ZPR would be very efficient are *point queries* which are queries asking for the number of (or a function on) readings having a single value. As this type of queries is not highly famous for sensor networks as compared to range queries, we believe that setting the Q_2 and Q_3 parameters to their default values would be a proper choice to achieve an acceptable ZP/ZPR performance.

As for the $threshold_1$ parameter, we encountered the same scenario as changing its value from 2 to 5 did not highly affect the ZP performance. We believe that the actual value of this parameter does not actually make a difference on the performance of the ZP except in the cases where a node is storing more than one zone with the average access frequencies of one of them is much higher than the average access frequencies rest of the zones. For our network settings, this case is very infrequent as the nodes are uniformly distributed in the network service area. This results in a uniform assignment of the DIM zones to nodes, which subsequently results in assigning one zone for each node for the majority of the cases. However, in cases where there exists many network nodes, each carrying the responsibility of multiple zones, e.g., due to temporary or permanent node deaths, the value of this parameter can have a higher effect on the performance of the ZP scheme.

To summarize, the results of our study show that the we selected a value of 0.3 would perform the best for the E_1 and E_2 constants. For the access frequency equations, results

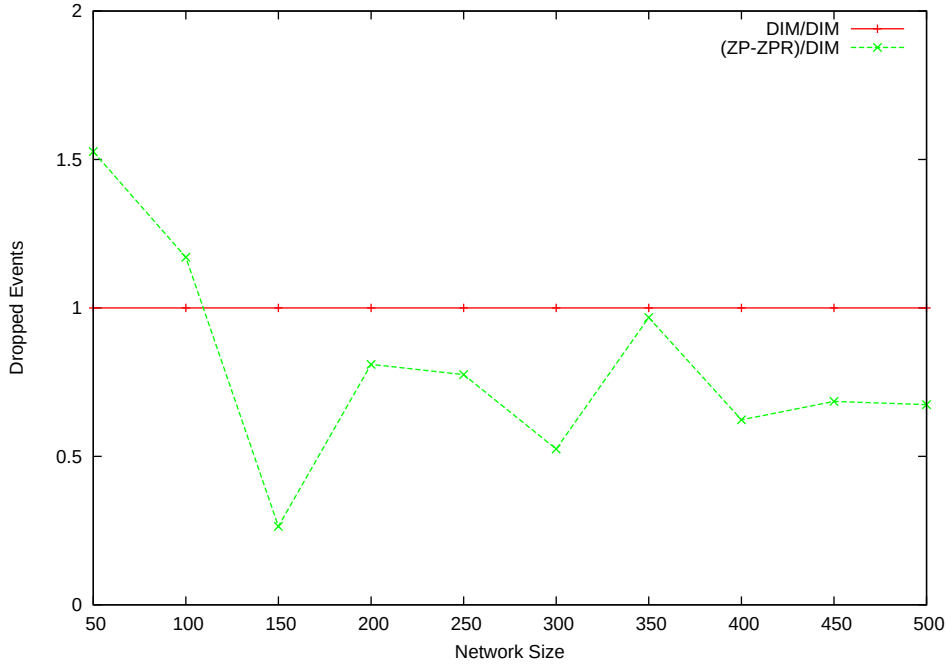
show that setting Q_1 to 5 would achieve the best results. As for Q_2, Q_3, and $threshold_!$, their effects are not tangible on the ZP/ZPR performance. In the rest of this section, we set these parameters to their default values.

We now move on to study the performance of ZP/ZPR for the different hotspot types. Our experimental results are shown in Figures 33 to 41. In these figures, we compare the performance of the basic DIM versus that of ZP/ZPR, with respect to QoD (R1), load balancing (R2), and energy consumption (R3). It is important to mention that we only included the DIM as the sole reference scheme as our simulations have shown that it completely outperform both the LS and the GHT schemes. Thus, in order to present more accurate and complete graphs, we only plot our ZP/ZPR scheme results and that of the DIM.

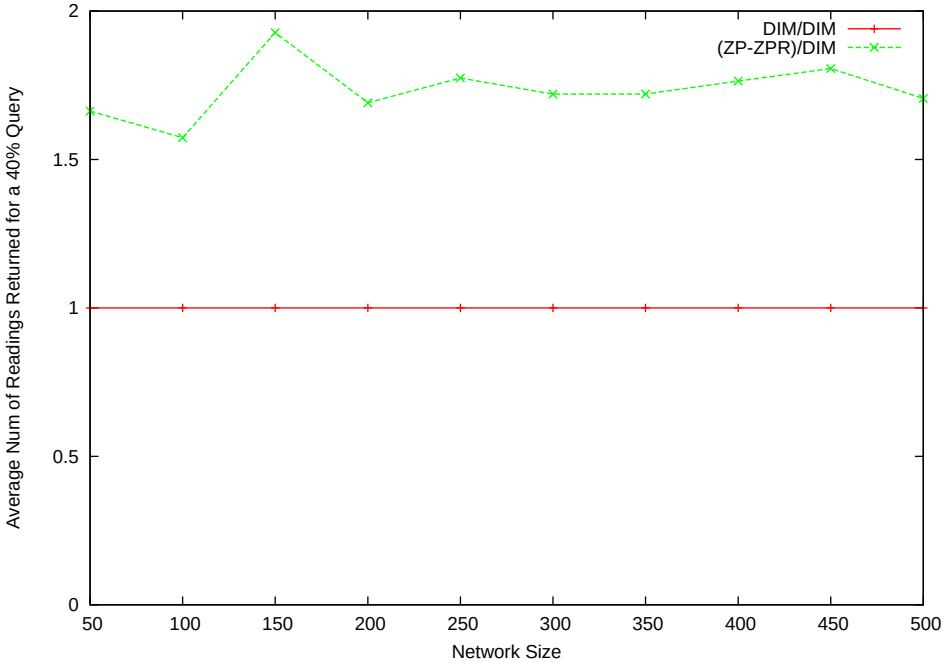
4.2.4.2 Single Static Query Hotspots In this subsection, we study the performance of ZP/ZPR compared to that of DIM when facing single query hotspots. We study the performance of the schemes in terms of QoD (R1), load balancing (R2), and energy consumption (R3).

R1. QoD: Figure 33(a) shows the number of events dropped by the different schemes when an 80% single query hotspot. Recall that the number of dropped events is directly proportional to the number of unanswered queries in the case of query hotspots. The figure shows that ZP/ZPR improves QoD by around 25% over DIM. Figure 33(b) shows the average number of readings returned for an arbitrary query asking for 40% of the attribute ranges for networks facing a 60% single query hotspot. The figure shows that the ZP/ZPR scheme performs around 75% better than the DIM in terms of improving the query result size for all networks sizes. Similar performance gains have been achieved by ZP/ZPR for queries of different range sizes. Also, we achieved similar results for hotspots of sizes up to 80%.

This result shows that ZP/ZPR decomposes the hotspot storage responsibility among a larger number of sensor nodes. This results in reducing collisions resulting from the query hotspot. Consequently, this helps in reducing the number of dropped events and increasing the expected result size of queries of different sizes in the sensor network. This has the great effect of increasing the data accuracy and the QoD of the different queries addressing the sensor network.



(a) Dropped Events for a 80% Single Query Hotspot



(b) 40% Query for a 60% Single Query Hotspot

Figure 33: ZP/ZPR: QoD Graphs for Single Query Hotspots

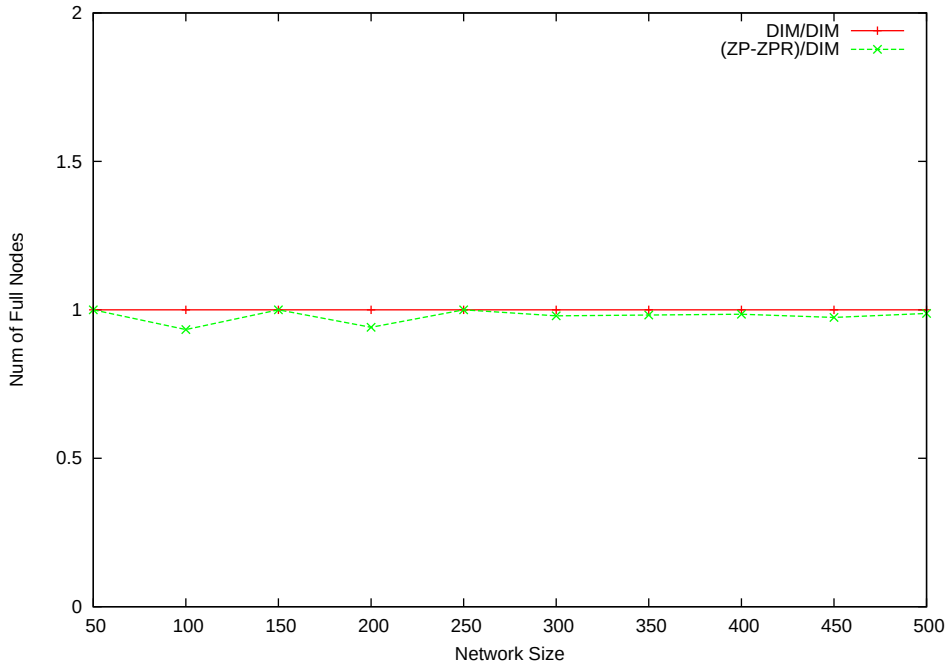
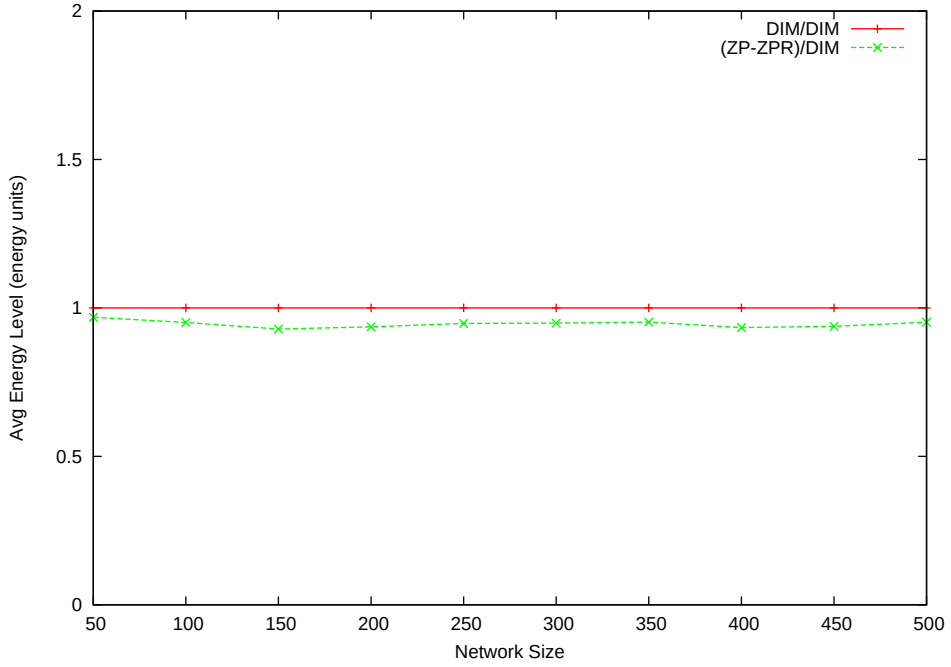


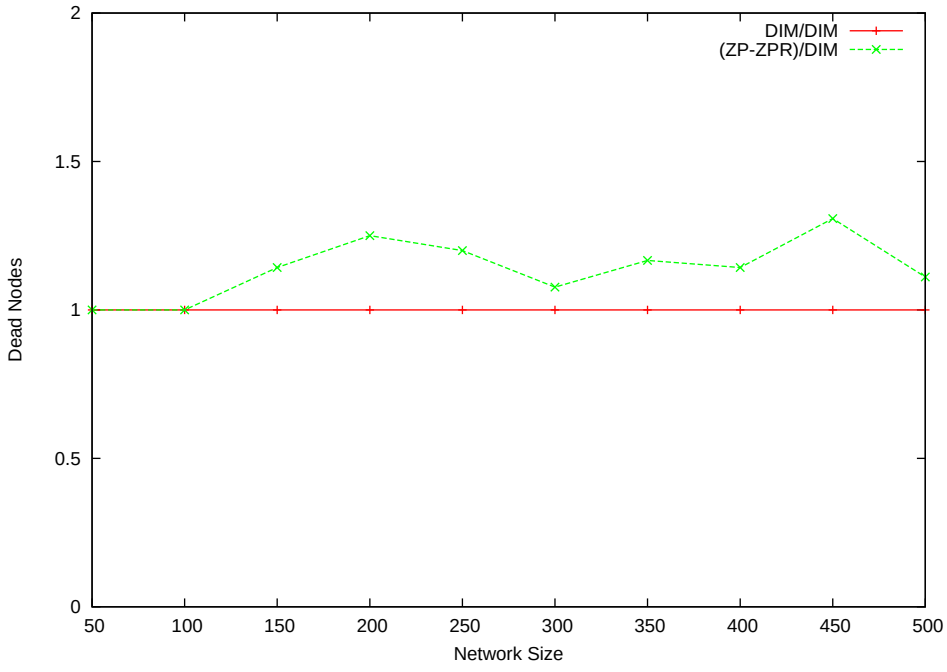
Figure 34: ZP/ZPR: Number of Full Nodes for a 60% Single Query Hotspot

R2. Load Balancing: Figure 34 presents the number of full nodes for networks facing a 60% single query hotspot. The Figure shows that the performance of networks applying ZP/ZPR on top of DIM is almost identical to those applying the basic DIM scheme for all network sizes. We achieved similar results for hotspots of sizes up to 80%. This result shows that the ZP/ZPR scheme does not cause the formation of storage hotspots by moving a large number of readings to few sensor nodes. Instead, applying the ZP/ZPR scheme has the benefit of load balancing the query loads on the network nodes without disturbing the initial balance of the storage load among the network sensors. Recall that the sensor reading values are drawn uniformly at random (from a uniform distribution) from the possible attribute range.

R3. Energy Consumption: Figures 35(a) and 35(b) show the average node energy and the number of dead nodes for networks facing an 80% and a 60% single query hotspot, respectively. The first figure shows that the average node energy drops by around 7% when applying ZP/ZPR. The second figure shows that the number of node deaths for ZP/ZPR is



(a) Average Node Energy for an 80% Single Query Hotspot



(b) Dead Nodes for a 60% Single Query Hotspot

Figure 35: ZP/ZPR: Energy Consumption Graphs for Single Query Hotspots

around 25% more than that of DIM (at most 1% of the size of the network). This result shows that applying ZP/ZPR on top of DIM does not add a considerable energy consumption overhead on the sensor nodes.

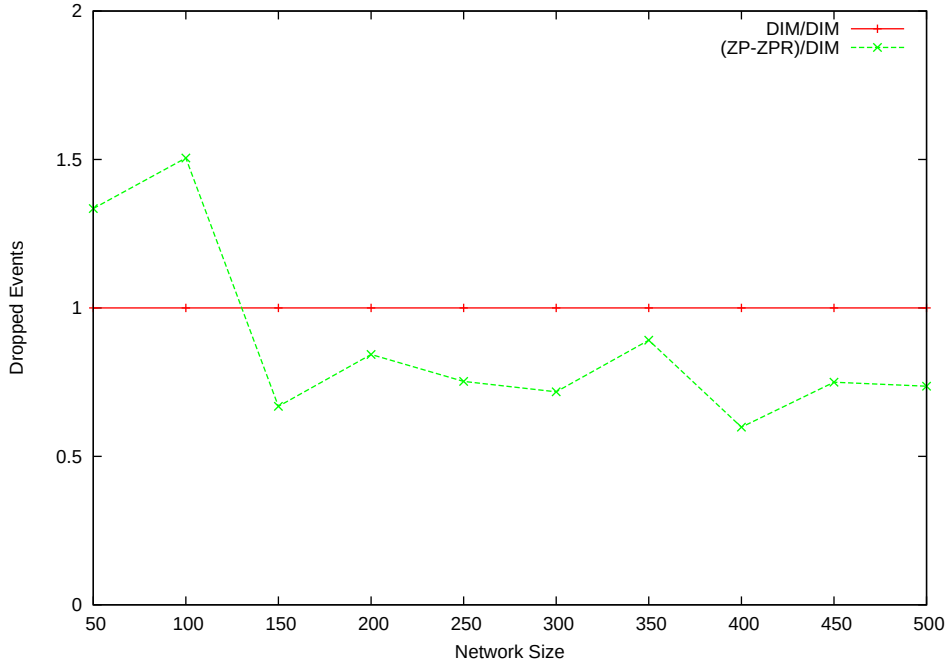
In conclusion, the above three results show that applying ZP/ZPR achieves good performance improvements against single query hotspots. QoD improvements are around 25%, while energy consumption overheads are around 7%.

4.2.4.3 Multiple Simultaneous Static Query Hotspots In this subsection, we study the performance of ZP/ZPR compared to that of DIM when facing multiple simultaneous query hotspots. We study the performance of (R2), and energy consumption (R3). The results presented in this section are based on simulating networks with two simultaneous hotspots. Recall that an x% multiple hotspot means that at least x% of the queries fall in the two hotspots with each query falling in any of the two hotspots with equal probability, i.e., at least $x/2$% of the queries are expected to fall in each of the two hotspots.

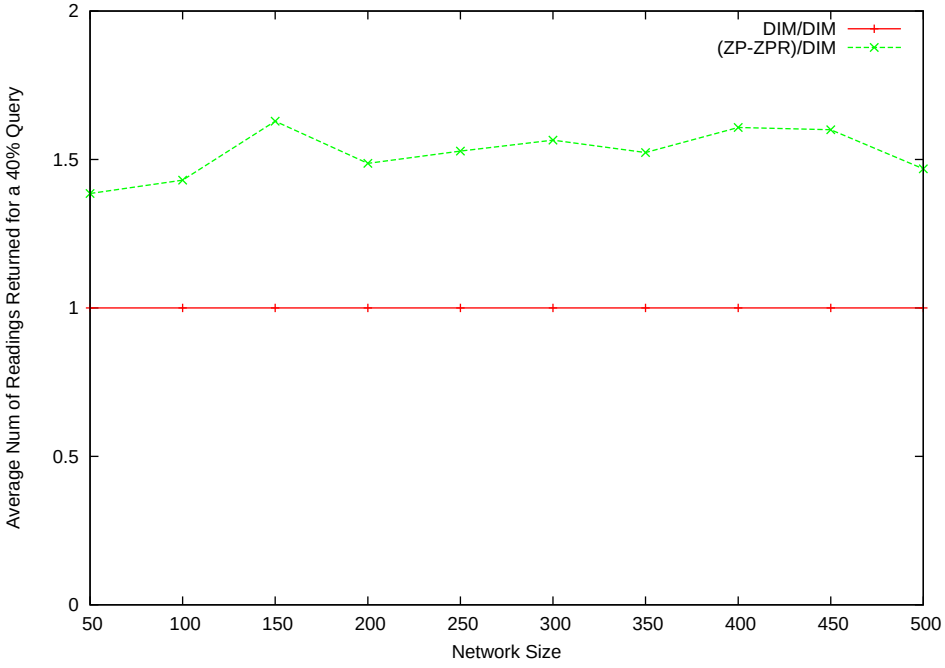
R1. QoD: Figures 36(a) and 36(b) compare the performance of ZP/ZPR with that of DIM in terms of the number of dropped events and the average result size for 40% queries when facing 80% multiple query hotspots, respectively. The first figure shows ZP/ZPR improves QoD by around 20% over DIM. The second figure shows that the average query result size of 40% queries for ZP/ZPR is around 50% larger than that of DIM. This result shows that ZP/ZPR highly improves the QoD performance of the DIM scheme when facing multiple hotspots. Similar results were achieved for hotspots of sizes up to 80%. It is important to note that the QoD improvements achieved for multiple query hotspots are slightly less than those achieved for single query hotspots.

R2. Load Balancing: Figure 37 compares the performance of ZP/ZPR to that of DIM in terms of the average the number of full nodes when facing 80% multiple query hotspots. The figure shows that the two schemes perform similarly in terms of load balancing. This result shows that the ZP/ZPR scheme is capable of simultaneously decomposing multiple hotspots without disturbing the storage load balancing of the DIM scheme or causing the formation of storage hotspots in the sensor network.

R3. Energy Consumption: Figures 38(a) and 38(b) compare the performance of ZP/ZPR



(a) Dropped Events



(b) 40% Query

Figure 36: ZP/ZPR: QoD Graphs for 80% Multiple Query Hotspots

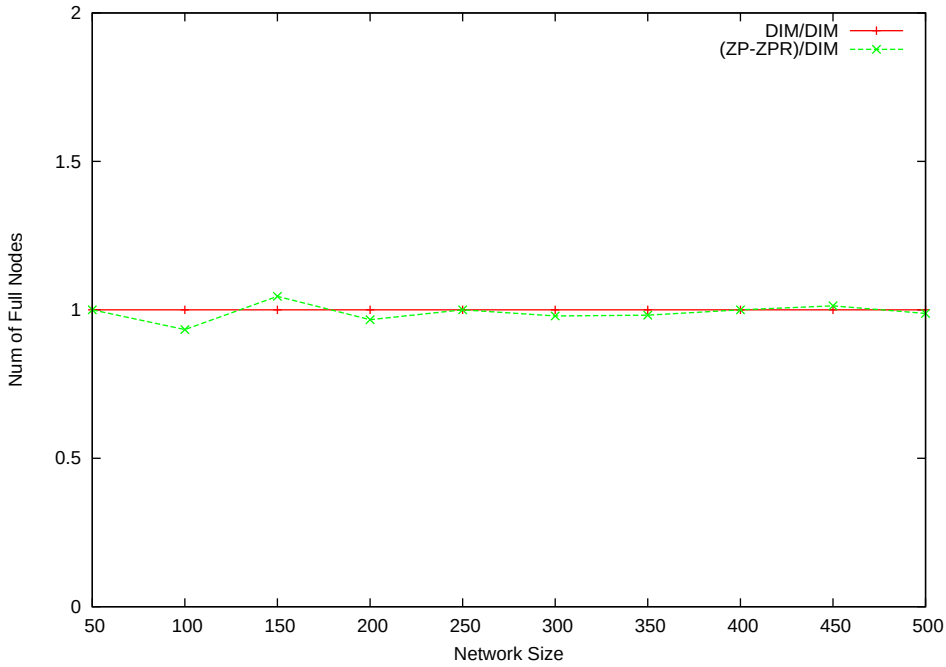
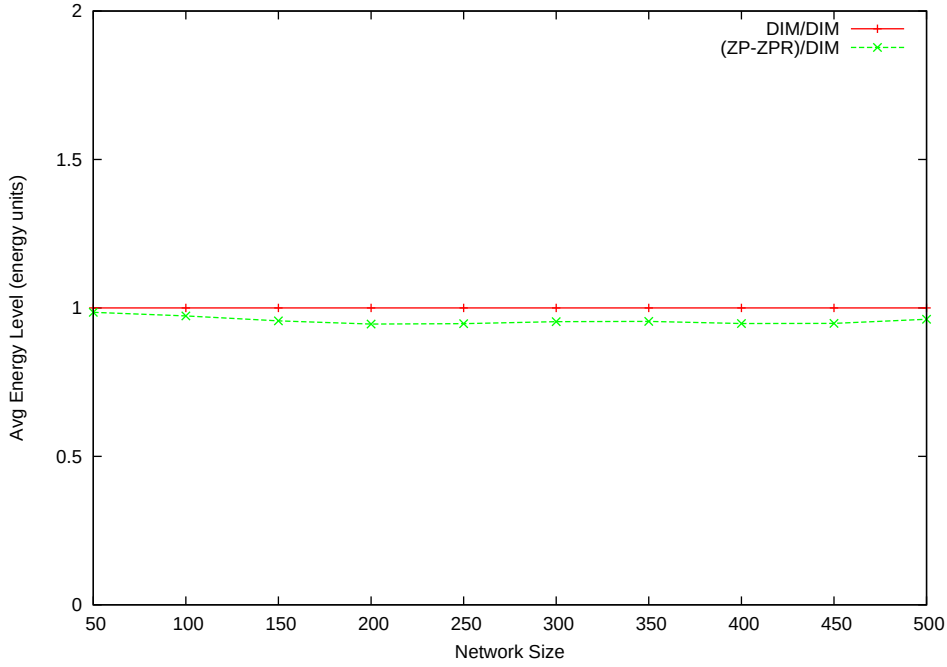


Figure 37: ZP/ZPR: Number of Full Nodes for 80% Multiple Query Hotspots

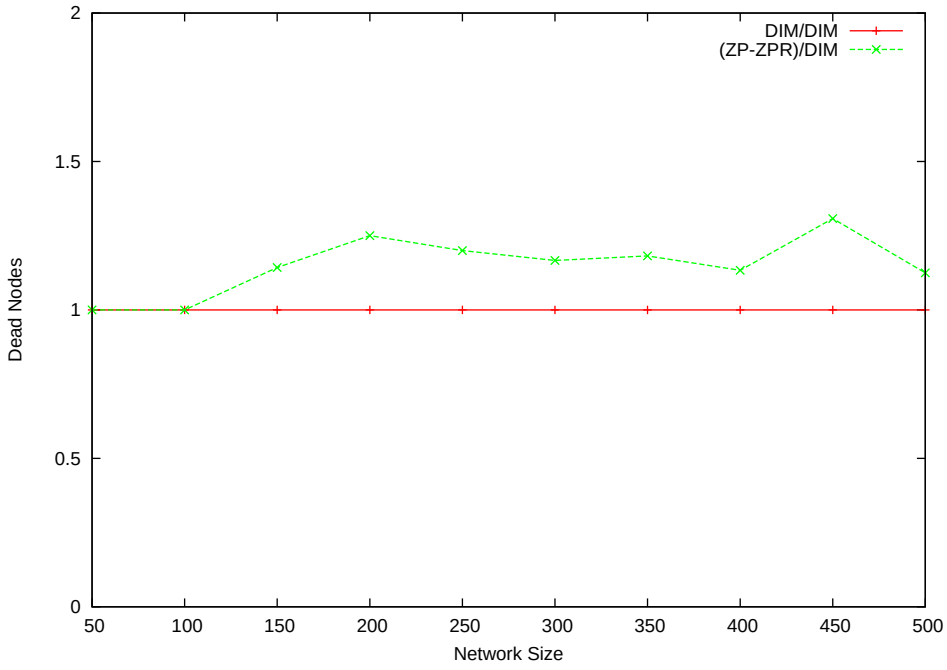
with that of DIM in terms of the average node energy and the number of dead nodes when facing 80% multiple query hotspots, respectively. The first figure shows that ZP/ZPR performs worse than that of DIM by around 5%. The second figure shows that ZP/ZPR scheme slightly increases the number of dead nodes by around 20% compared to DIM (around 1.25% of the whole network size). Overall, the two figures show that ZP/ZPR achieves very good energy savings by decomposing multiple query hotspots while not imposing a high energy consumption load on the network nodes. This was valid for hotspots of sizes up to 80%.

In conclusion, the above three results show that ZP/ZPR achieves good performance improvements for all network sizes when facing multiple query hotspots. QoD improvements are around 20% while energy consumption overhead is around 5%.

4.2.4.4 Moving Query Hotspots In this subsection, we study the performance of ZP/ZPR compared to that of DIM when facing a moving query hotspot. We study the performance of the schemes in terms of QoD (R1), load balancing (R2), and energy con-



(a) Average Node Energy



(b) Dead Nodes

Figure 38: ZP/ZPR: Energy Consumption Graphs for 80% Multiple Query Hotspots

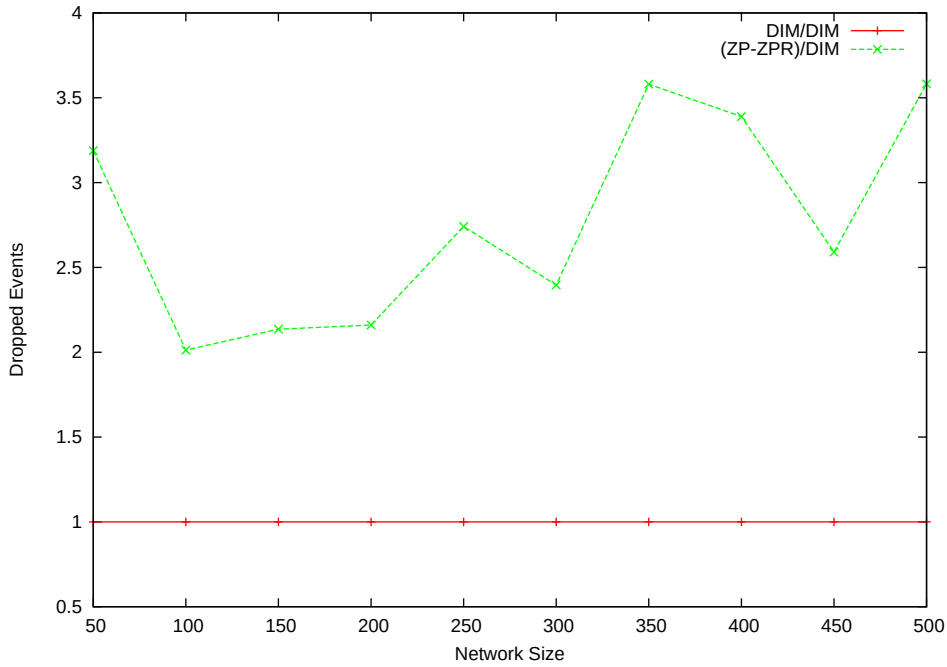


Figure 39: ZP/ZPR: Dropped Events for a 60% Moving Query Hotspot

sumption (R3). We simulated an x% moving hotspot as follows. Each run has been divided into 5 steps. The hotspot starts in range $[t_1, t_1 + i]$ in the first step, then moves on to $[t_1 + i, t_1 + 2i]$ in the second step, etc. In each of the steps, x% of the generated queries fall in the step's hotspot range. We simulated hotspot sizes up to 50%.

R1. QoD: Figure 39 compares the performance of ZP/ZPR to that of DIM in terms of the number of dropped events for a 60% moving query hotspot. The main observation from the figure is that DIM performs better than ZP/ZPR versus moving hotspots. In terms of QoD, ZP/ZPR performs around 300% worse than DIM. This can be explained as follows. As the hotspot moves, ZP/ZPR tries to decompose it by sending part of the hotspot storage responsibility away from the hotspot area. Due to the mobility of the hotspot, ZP/ZPR causes the formation of new hot areas where event collisions increase. This subsequently affects the QoD of the ZP/ZPR scheme as compared to that of DIM.

R2. Load Balancing: Figure 40 presents the number of full nodes for networks facing an 80% moving query hotspot. The Figure shows that the performance of networks applying

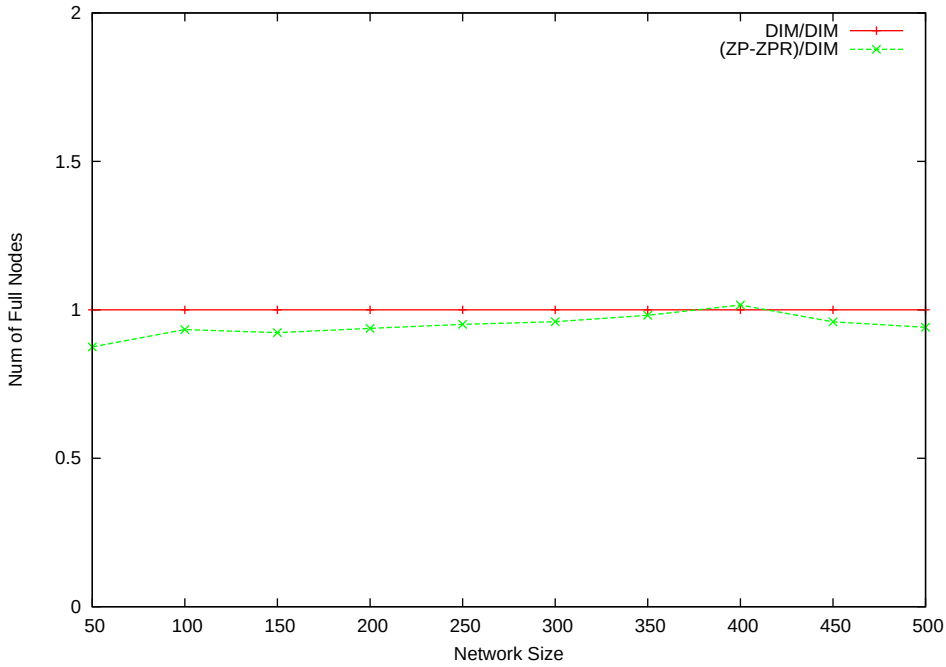
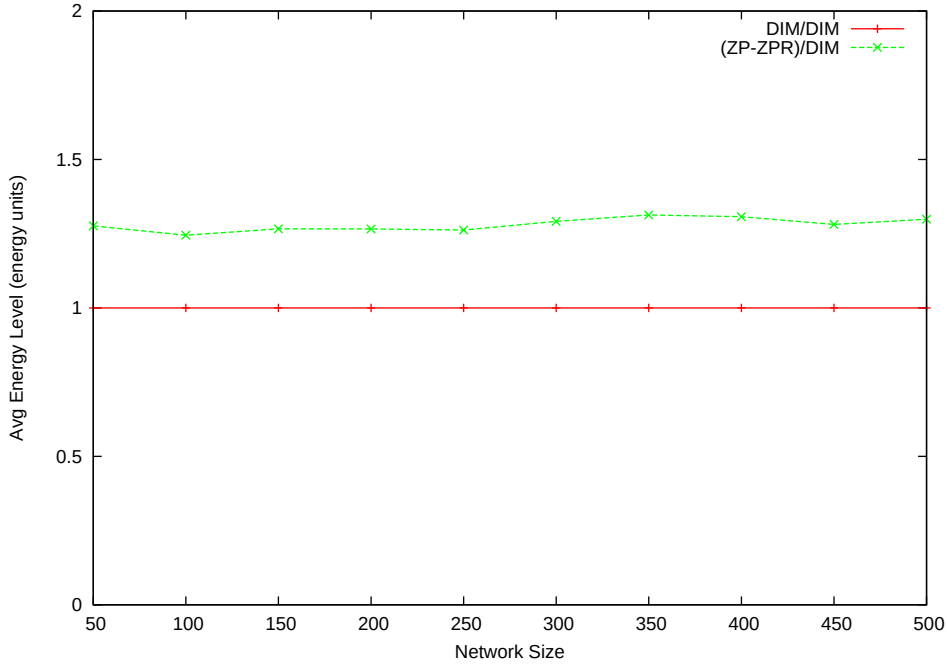


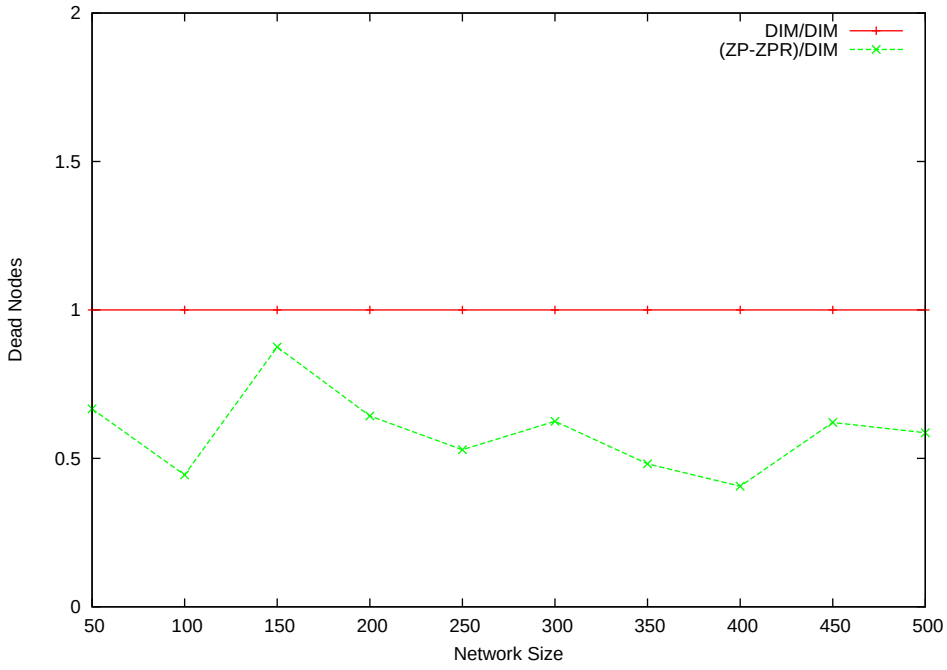
Figure 40: ZP/ZPR: Number of Full Nodes for an 80% Moving Query Hotspot

ZP/ZPR on top of DIM is equal to those applying the basic DIM scheme for all network sizes. This result shows that the storage load balancing of the two schemes is comparable. This shows that ZP/ZPR does not cause any major disturbance to the load-balanced storage in the network.

R3. Energy Consumption: Figures 41(a) and 41(b) compare the performance of ZP/ZPR with that of DIM in terms of the average node energy and the number of dead nodes when facing an 80% moving query hotspot, respectively. The first figure shows that the ZP/ZPR performance is 25% better than DIM. The second figure shows that applying ZP/ZPR on top of DIM highly reduces the number of dead nodes (incurred by DIM) by around 50% (around 3% of the network size). This result shows that, although the application of ZP/ZPR reduces the QoD as it increases the number of collisions, it has an important effect on improving the energy consumption imposed on the different networks nodes. This improvement comes due to the decomposition of the moving query hotspot among a larger number of nodes, thus, achieving a better energy consumption load balancing across the sensor network.



(a) Average Node Energy



(b) Dead Nodes

Figure 41: ZP/ZPR: Energy Consumption Graphs for an 80% Moving Query Hotspot

Table 4: ZP/ZPR Performance (Relative to DIM) for Query Hotspots

Hotspot Type	QoD Improvements	QoS Overheads
Single Static Hotspots	25%	7%
Multiple Static Hotspots	20%	5%
Moving Hotspots	−300%	−25%

The above three results have shown ZP/ZPR decreases QoD as it causes additional collisions in the network. In fact, ZP/ZPR decreases QoD by around 300% compared to DIM. In terms of energy consumption, ZP/ZPR is 25% better than DIM.

In conclusion, the ZP/ZPR scheme improves both QoD and energy savings when facing single and multiple query hotspots, while it only improves energy savings against moving query hotspots.

4.2.4.5 Discussion In this section, we studied ZP/ZPR for single, multiple, and moving hotspots. Additionally, we studied the individual effect of energy thresholds (E_i) and the average access frequency thresholds (Q_i and $threshold_1$) on the ZP/ZPR performance. For single storage hotspots, QoD improvements of ZP/ZPR are around 25% over DIM, while energy consumption overhead are around 7% over DIM. For multiple hotspots, QoD improvements are around 20% over DIM, while energy consumption overhead is around 5% over DIM. For moving hotspots, DIM is outperforming ZP/ZPR in terms of QoD. Table 4 summarizes the ZP/ZPR performance for query hotspots.

Now that we have completely presented and studied the performance of the ZP/ZPR scheme, we move on to present the ZS/ZPR/ZPR scheme whose goal is to detect and decompose mixed hotspots.

4.3 LOCAL DETECTION AND DECOMPOSITION OF MIXED HOTSPOTS

Based on the previous two sections, there are many similarities between the ZS and the ZP/ZPR schemes. Considering the hotspot detection scheme, each of the two schemes depends on comparing the node's load with those of its neighbors. The load can be the storage load, as in ZS, or the query load, as in ZP. Once a hotspot is detected in a given zone, each of the ZS and ZP schemes copes with the hotspot by decomposing the zone into smaller zones and passing the storage responsibility of parts of these zones to some of the node's neighbors. ZPR's Replication substitutes the zone partitioning in case the hot zone is a fairly small subrange of attribute range. The similarities between the ZS and ZP/ZPR schemes open the door for the possibility of using the mix of ZS/ZP/ZPR to detect and decompose mixed hotspots. In the following subsection, we describe the strategy used by ZS/ZP/ZPR in coping with mixed hotspots. Recall that a mixed hotspot is composed of a collection storage and query hotspots simultaneously taking place in the network.

4.3.1 The ZS/ZP/ZPR Scheme

The basic idea of the ZS/ZP/ZPR scheme is to blend the similar components of the ZS and the ZP/ZPR schemes to form new unified components. Branching decisions are then added to decide the sequence of applying the non-similar components. ZS/ZP/ZPR mainly blends the hotspot detection of the ZS and that of ZP/ZPR into a new hotspot detection technique that determines whether a storage and/or a query hotspot is arising in any sensor node. Based on the hotspot nature, either the DMC, the PC, or both are triggered for evaluation. Furthermore, the SZL and the TZL are blended into one list type that we denote by the *Decomposed Zone List (DZL)* that contains the zone bit-code, the original zone donor, and the final node destination. The third entry of the DZL acts like the final migrator when ZS is applied or the final receiver in case ZP is applied. We now describe the details of ZS/ZP/ZPR.

The first step of ZS/ZP/ZPR is to detect hotspots. Each node detects hotspots arising

in its zone responsibility by keeping track of its storage load and the AQFs of its zones throughout the network operation. Periodically, the node compares these counters with the corresponding counters of its neighbors. The comparison technique is exactly similar to those individually applied by ZS and ZP/ZPR to detect storage and query hotspots, respectively. The result of this step is to determine whether a node is experiencing a normal load, or whether it is experiencing a storage and/or a query hotspot. This concludes the first step.

The second step in ZS/ZP/ZPR is to decompose the hotspot arising in any node once they are detected. This can be done as follows. In case one (isolated) hotspot type is detected in a given type, that is either a storage hotspot or a query hotspot, and not both of them simultaneously, the corresponding scheme is triggered. That is the DMC (and thus ZS) is triggered whenever a storage hotspot is encountered while the PC (and thus ZP/ZPR) is triggered for decomposing query hotspots.

In case two simultaneous hotspots of different types arise at the same node, i.e., a storage and a query hotspot, the appropriate action is taken based on whether the hotspots are in the same zone or no. In case the hotspots are in the same zone, this means that the zone must be decomposed into two or more parts to decompose the two hotspots. As the storage hotspot may be spanning other neighboring nodes, the DMC needs to be applied in order to find a suitable migrator among the node's neighbors to share the node's zone responsibility with. Applying ZP will not be possible in this case as the partitioned zone may be passed to a node already experiencing a storage hotspot. Unlike in ZS, the zone decomposition process may be done in a non-uniform way, i.e., the zone must not be split in two equal parts. Instead, the zone may be split into non-equally bit-code sized zones to make sure that the resulting sub-zones have uniform storage and AQF loads. An example for that decomposition is that a 101 zone may be decomposed into 1010, 10110, and 10111 sub-zones.

The second case for the two hotspots arising in the same node is for them to be initiating in two different zones. In this case, both the DMC and the PC need to be applied, the two zones have to be decomposed, and their storage responsibilities have to be shared with neighboring nodes. However, it is possible that the node may not have enough energy for either evaluating both the DMC and the PC, or applying the actual ZS and ZP/ZPR processes. In this case, two options are possible.

1. Compare the sizes (in terms of the load and AQF ratios) and spans (in terms of the number of neighbors experiencing the same hotspot) of the two hotspots. Decompose the more severe one. If both are of equal size and span, select the one to decompose either at random or based on a pre-specified prioritization of one hotspot type over the other. We adopt this technique in our experimental evaluation.
2. Always prioritize the decomposition of one hotspot type over the other. For example, deciding to always decompose query hotspots if they arise regardless of whether a storage hotspot is experienced or no. This decision can be taken a priori during the network setup phase based on the expected nature of the network application, usage, query load, node types, etc.

As it is clear from the above presentation, combination of the ZS and the ZP/ZPR schemes should be able to decompose mixed hotspots regardless of whether they are correlated or no. Furthermore, blending the ZS and the ZP/ZPR schemes comes with no additional implementation overheads as the mix ZS/ZP/ZPR does not add any new major step in addition to the hotspot detection and decomposition phases already implemented individually by the ZS and the ZP/ZPR schemes. Our actual ZS/ZPZ/ZPR implementation does not exceed 140 lines of C++ code. Once run, it does not take more than 2 KBytes of memory. Furthermore, our experimental study showed that the DZL size does not exceed, in average, that of the SZL or that of the TZL. This shows in conclusion that the implementation overhead that ZS/ZP/ZPR will impose on sensor nodes will be relatively small.

Experimental evaluation shows that the main advantages of ZS/ZP/ZPR are:

- Improving *QoD* by distributing the hotspot events (readings/queries) among a larger number of sensors. Improvements ranged from 75% over DIM for single mixed hotspots to at least 50% over DIM for multiple mixed hotspots.
- Increasing the *energy savings* by balancing energy consumption among sensor nodes. Energy consumption overhead additionally imposed by ZS/ZP/ZPR was around 12% (per node) over DIM for all hotspot types.

This was valid for hotspots of sizes ranging from 40% to 80%.

4.3.2 ZS/ZP/ZPR Experimental Evaluation

In this section, we study the performance of the final ZS/ZP/ZPR version for both uniform loads. and mixed hotspots. We compare the ZS/ZP/ZPR performance to that of DIM against uniform loads (Section 4.3.2.1), single mixed hotspots (Section 4.3.2.2), multiple mixed hotspots (Section 4.3.2.3), and moving mixed hotspots (Section 4.3.2.4). For each of the hotspot settings, we compare the two schemes in terms of QoD, load balancing, and energy consumption.

After experimentally verifying that this achieves the best ZS/ZP/ZPR, we set the parameters of the ZS/ZP/ZPR scheme to their default values already concluded from the individual ZS and ZP/ZPR studies.

Throughout this section, the node storage capacity is equal to 30 readings and the node initial energy capacity is equal to 70 units. Therefore, a *full sensor node* is defined to be a sensor node having 30 readings in its cache. Similarly, a node is depleted (and consequently considered dead) as soon as it consumes 70 energy units. Once a node is dead, all readings stored in this node are considered lost. Based on the DIM scheme, the storage responsibility (a subset of the attribute range) of the dead node is assigned to one of its direct neighbors. Recall that we define an event to be either a reading or a query.

For each of our experiments, we study three aspects: QoD (R1), load balancing (R2), and energy consumption (R3). For the QoD, we study the number of dropped events (readings/queries) and the average node storage. We refer to the percentage of QoD improvement to be the percentage of decrease in event drops. For the load balancing, we study the number of full nodes. As for energy consumption, we study the average node energy and the number of dead nodes. The average node energy is the one that defines the improvement or the downgrading in the energy consumption performance. To be statistically significant, we conducted 5 simulation runs for each of the experiments and taken the average of values across all runs.

For each of the hotspot types, we conducted experiments on different hotspot sizes ranging from 20% to 80%. Unless otherwise stated, performing well on the large hotspot sizes, i.e., $[60\%, 80\%]$, implies a good performance on the moderate sized hotspots, i.e., $[40\%, 60\%]$.

In most of the cases, the performance burden imposed to the network by small hotspots, i.e., hotspots less than 40%, does not justify the cost paid to detect and decompose the hotspots.

The main learned lessons out of the experimental evaluation of ZS/ZP/ZPR can be summarized in the following points:

1. Impose around 2% energy consumption overhead per node over DIM for networks with no hotspots.
2. When facing single mixed hotspots, ZS/ZP/ZPR achieves a QoD improvement of around 75% over DIM. Towards this, ZS/ZP/ZPR imposes an energy consumption overhead of 12% per node over DIM.
3. For multiple hotspots, ZS/ZP/ZPR improves QoD by around 50% to 70% while imposing a 12% energy consumption overhead over DIM.
4. For moving hotspots, ZS/ZP/ZPR scores a 50% QoD improvement while introducing an energy consumption overhead of 12% over DIM.

Our experimental results are presented in Figures 42 to 53. It should be noted that we only concentrate on the comparison between the ZS/ZP/ZPR and the DIM schemes as DIM continues to perform highly better than the LS and GHT schemes against mixed hotspots.

4.3.2.1 Uniform Loads In this section, we study the performance of the ZS/ZP/ZPR for uniform loads. Our goal is to measure the energy consumption overhead imposed by our schemes when the network experiences no hotspots. It is clear that the energy consumption overhead of the ZS/ZP/ZPR scheme represents an upper bound for the energy consumption overhead individually imposed by the ZS and the ZP/ZPR schemes. Therefore, we only study the uniform loads case for ZS/ZP/ZPR to avoid repetition.

Figure 42 compares the performance of our ZS/ZP/ZPR scheme to that of DIM in terms of average energy for uniform loads. The figure shows that the implementation of the ZS/ZP/ZPR schemes on top of DIM does not add a considerable burden and does not cause an intolerable performance degradation when the sensor network experiences uniform loads. Decrease in average node energy is around 2%.

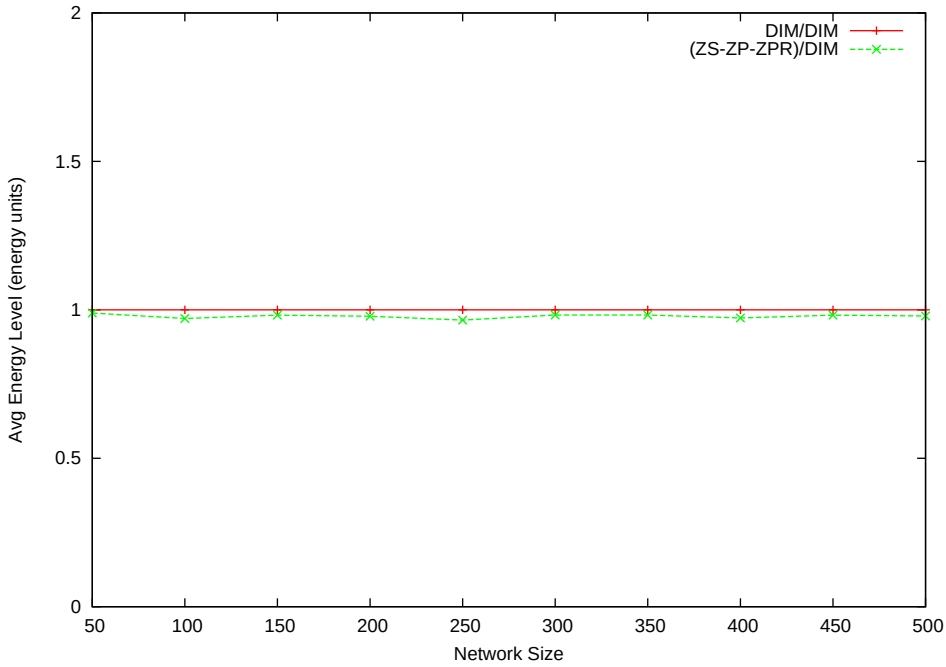
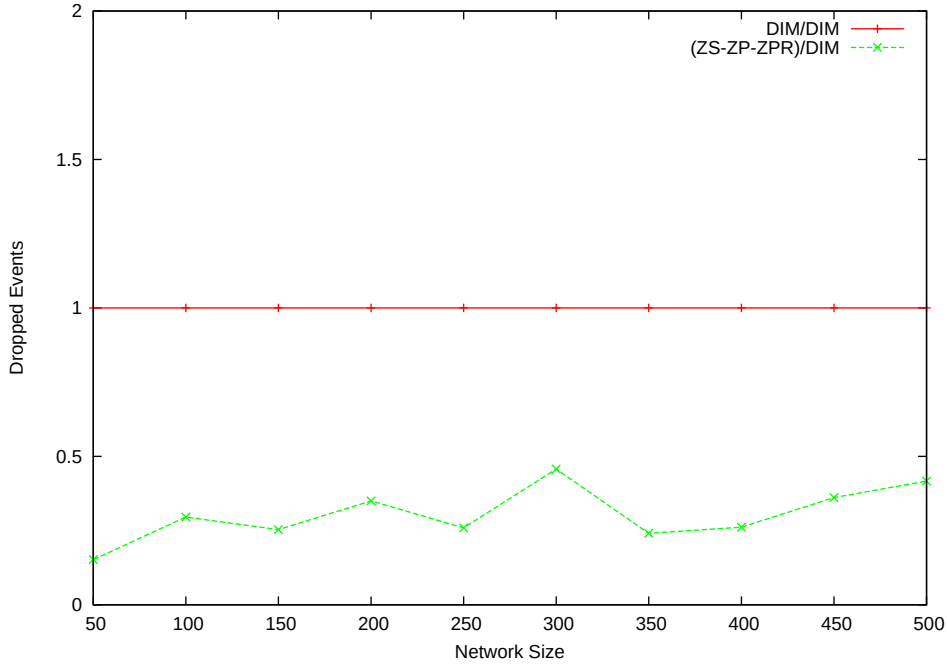


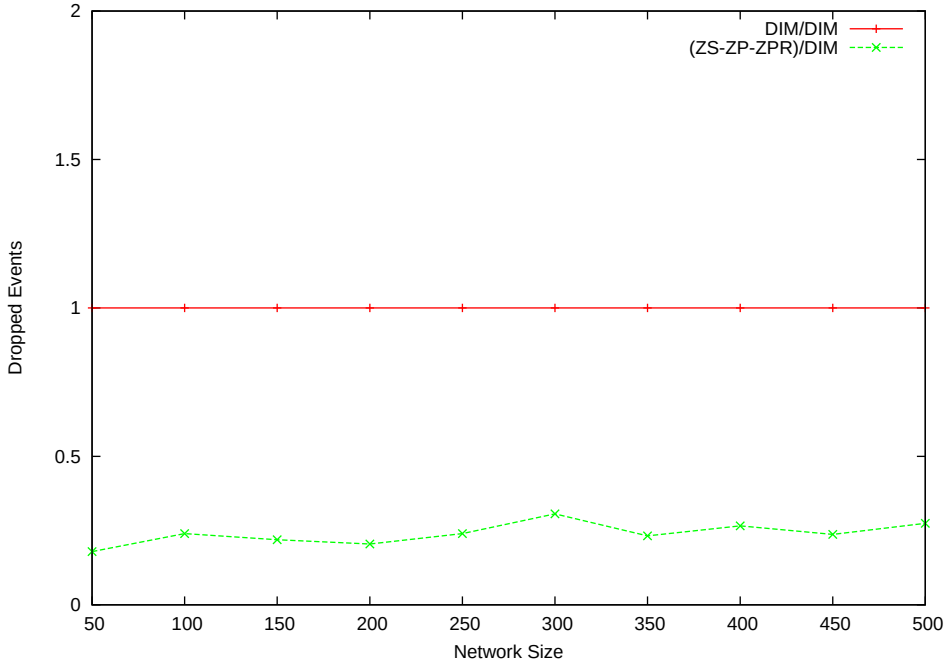
Figure 42: ZS/ZP/ZPR: Average Node Energy for Uniform Loads

4.3.2.2 Single Static Mixed Hotspots The following three results compare the performance of the ZS/ZP/ZPR scheme to that of the DIM for single static mixed hotspots. Comparison is based on QoD (R1), load balancing (R2), and energy consumption (R3).

R1. QoD: Figures 43(a) and 43(b) compare the performance of ZS/ZP/ZPR to that of DIM in terms of the number of dropped events when facing single static mixed hotspots of sizes 60% and 80%, respectively. The figures show that ZS/ZP/ZPR achieves around 75% QoD improvement over DIM. This shows the high improvement that ZS/ZP/ZPR achieves, in terms of increasing the data persistency and the query answering capability, compared to the basic DIM scheme. Figure 44(a) presents the average query result size of 40% queries when facing an 80% single mixed hotspot. The figure shows that results of the queries for ZS/ZP/ZPR are around 3 times larger than those for DIM. Figure 44(b) compares the two schemes in terms of average node storage when facing 60% single storage hotspots. The figure shows that ZS/ZP/ZPR increases the average node storage by around 25% per node for all networks sizes. The results are valid for hotspots of sizes up to 80%.



(a) 60% Single Mixed Hotspot



(b) 80% Single Mixed Hotspot

Figure 43: ZS/ZP/ZPR: Dropped Events for Single Mixed Hotspots

All four figures show the high QoD improvement that ZS/ZP/ZPR achieves compared to the DIM scheme. An important observation is that this improvement exceeds the improvements that ZS and ZP/ZPR individually scored against single storage and query hotspots, respectively. This shows the high ability of the ZS/ZP/ZPR scheme to take benefit of the correlation among the two hotspot types to improve the overall DIM performance.

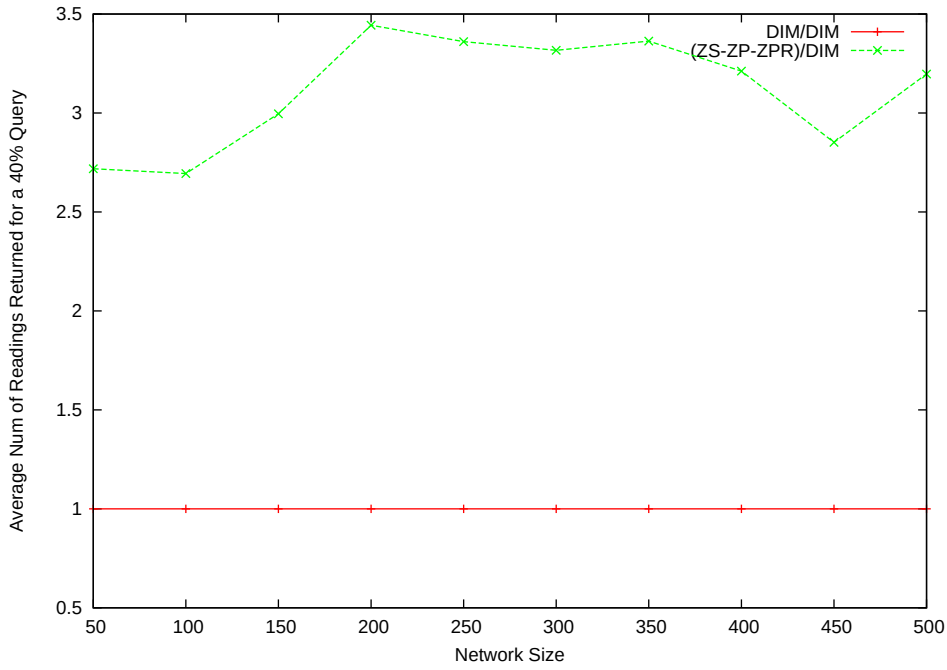
R2. load balancing: Figure 45 compares the performance of the two schemes in terms of the number of full nodes when facing a 60% single mixed hotspot. The figure shows that ZS/ZP/ZPR has a constant number of full nodes as opposed to the number of full nodes for DIM increasing linearly with the increase in the network size. This shows that ZS/ZP/ZPR highly improves the DIM load balancing performance. The main observation is that the amount of improvement increases proportionally to the network size.

R3. Energy Consumption: Figures 46(a) and 46(b) compare the performance of the two schemes in terms of dead nodes and average node energy when facing 60% single mixed hotspots, respectively. The first figure shows that the introduction of the ZS/ZP/ZPR scheme multiplies the number of dead nodes by 2.25 times. The increase in node deaths is around 3.4% of the size of the network. The second figure shows that ZS/ZP/ZPR reduces the average node energy by around 12%. The two figures collectively show that ZS/ZP/ZPR introduces a moderate energy consumption overhead on the DIM scheme.

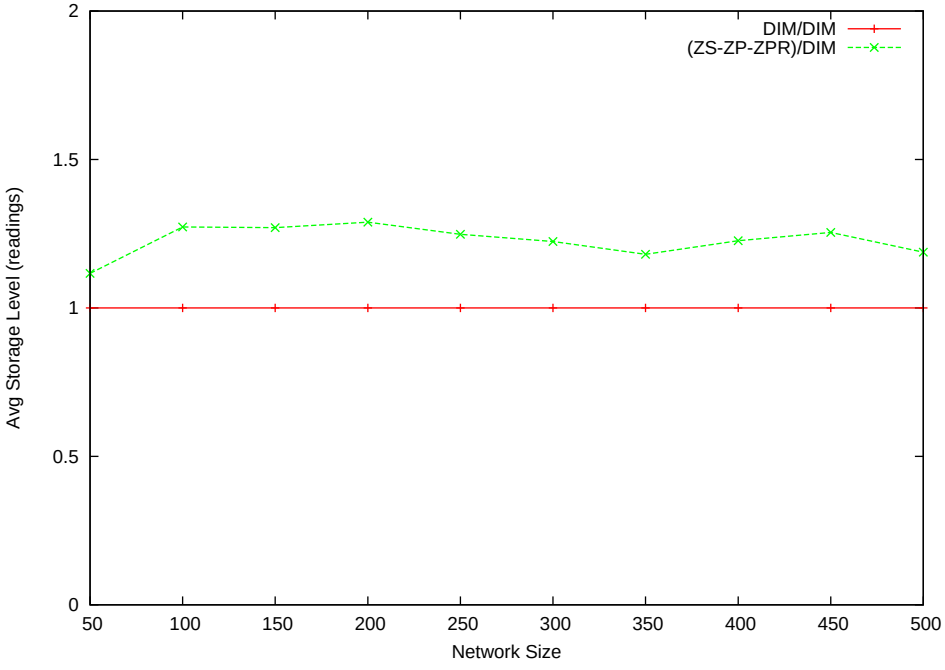
In conclusion, ZS/ZP/ZPR improves QoD by around 75% over DIM while increasing the energy consumption overhead imposed on each node in the network by around 12%. This shows its high ability to cope with single mixed hotspots with a moderate energy consumption cost imposed on the sensor network nodes. This can be considered as a good achievement when comparing this energy consumption cost to the amount of QoD improvement that the scheme achieves compared to the basic DIM.

4.3.2.3 Multiple Simultaneous Static Mixed Hotspots

The following three results compare the ZS/ZP/ZPR performance to that of the DIM when facing multiple static mixed hotspots. Comparison is based on QoD (R1), load balancing (R2), and energy consumption (R3). The results presented in this section are based on simulating networks with two simultaneous mixed hotsposts. Recall that an x% multiple hotspot means that at least x%



(a) 40% Query for an 80% Single Mixed Hotspot



(b) Average Node Storage for a 60% Single Mixed Hotspot

Figure 44: ZS/ZP/ZPR: QoD Graphs for Single Mixed Hotspots

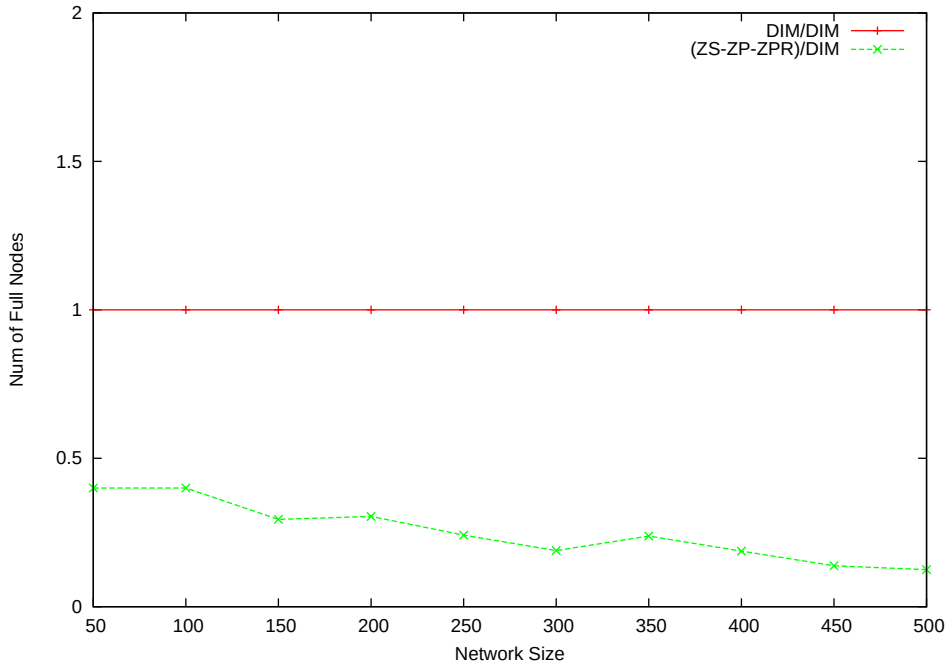
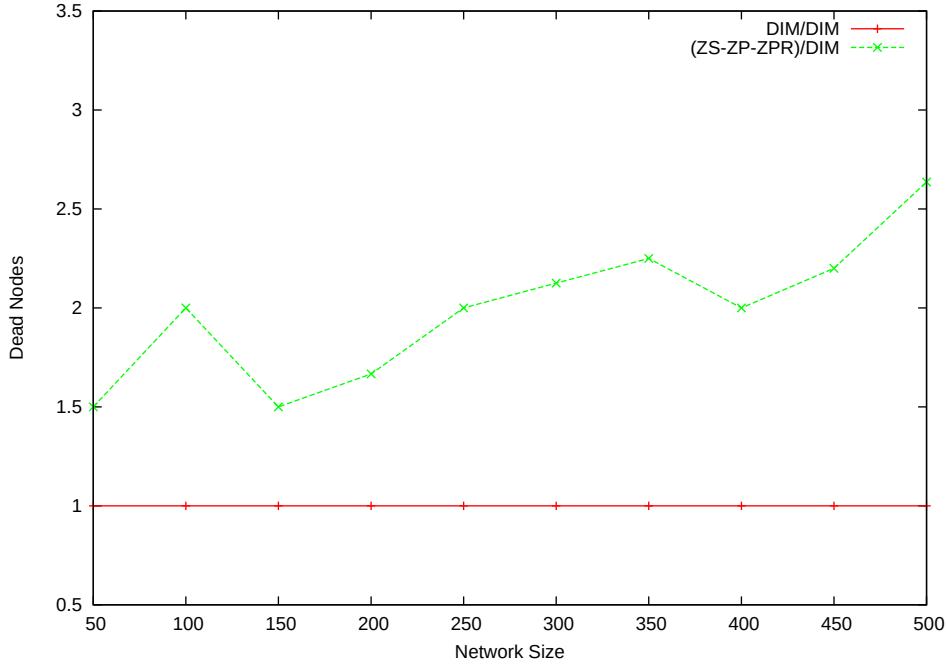


Figure 45: ZS/ZP/ZPR: Number of Full Nodes for a 60% Single Mixed Hotspot

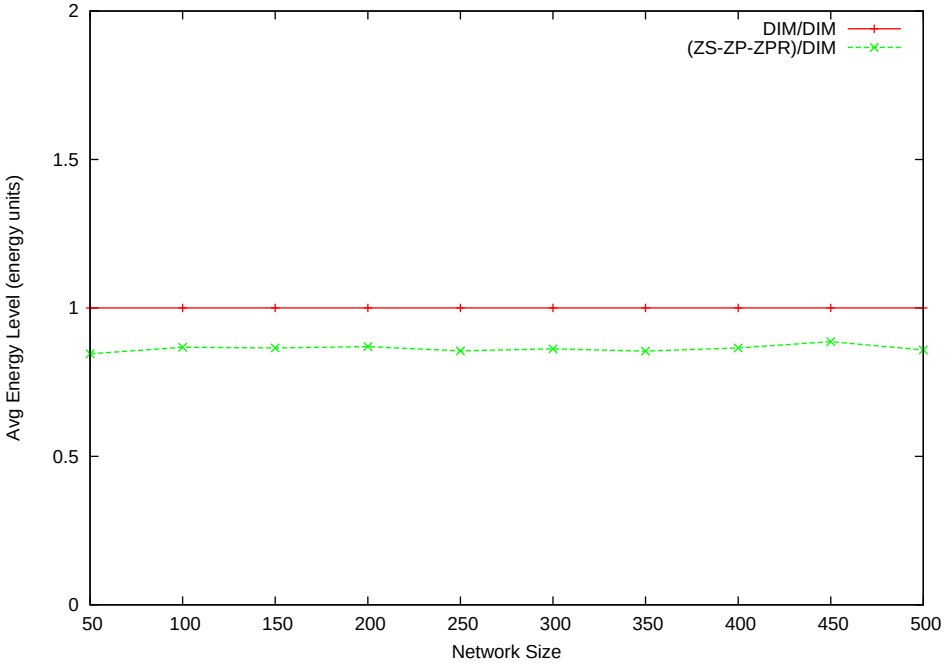
of the readings fall in the two hotspots with each reading/query falling in any of the two hotspots with equal probability, i.e., at least $x/2\%$ of the readings/queries are expected to fall in each of the two hotspots.

R1. QoD: Figures 47(a) and 47(b) compare the number of dropped events of the two schemes when facing 60% and 80% multiple mixed hotspots, respectively. The figure shows that ZS/ZP/ZPR highly improves the DIM performance by reducing the amount of dropped events by around 50% to 70%. We achieved similar results for hotspots of sizes up to 80%. However, it should be noted that the improvement in performance is less than that achieved against single mixed hotspots. This is due to the slight effect of the increased collisions symptom that the ZS schemes (and consequently the ZS/ZP/ZPR scheme) sometimes faces in the case of multiple hotspots.

Figure 48(a) shows the improvement that ZS/ZP/ZPR achieves, compared to DIM, in terms of the average result size of 40% queries when facing 80% multiple mixed hotspots. Basically, ZS/ZP/ZPR at least doubles the query result sizes compared to DIM. Figure

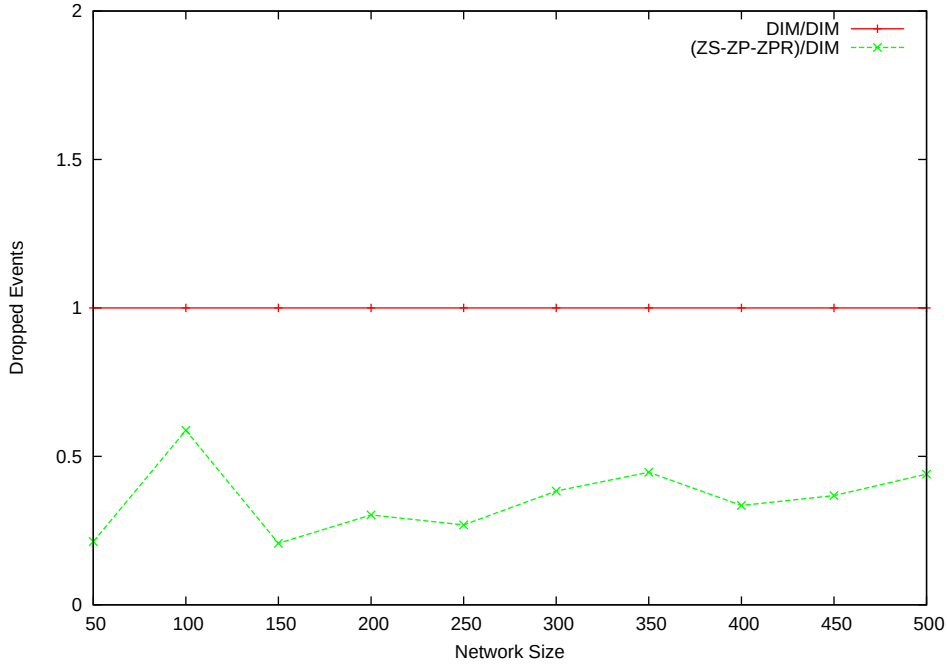


(a) Dead Nodes

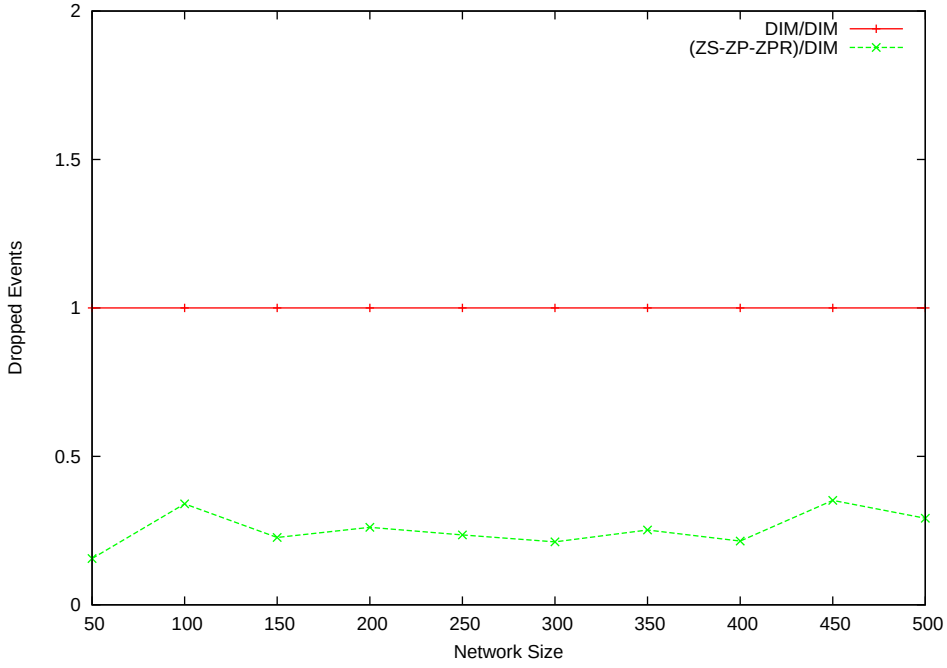


(b) Average Node Energy

Figure 46: ZS/ZP/ZPR: Energy Consumption Graphs for a 60% Single Mixed Hotspot

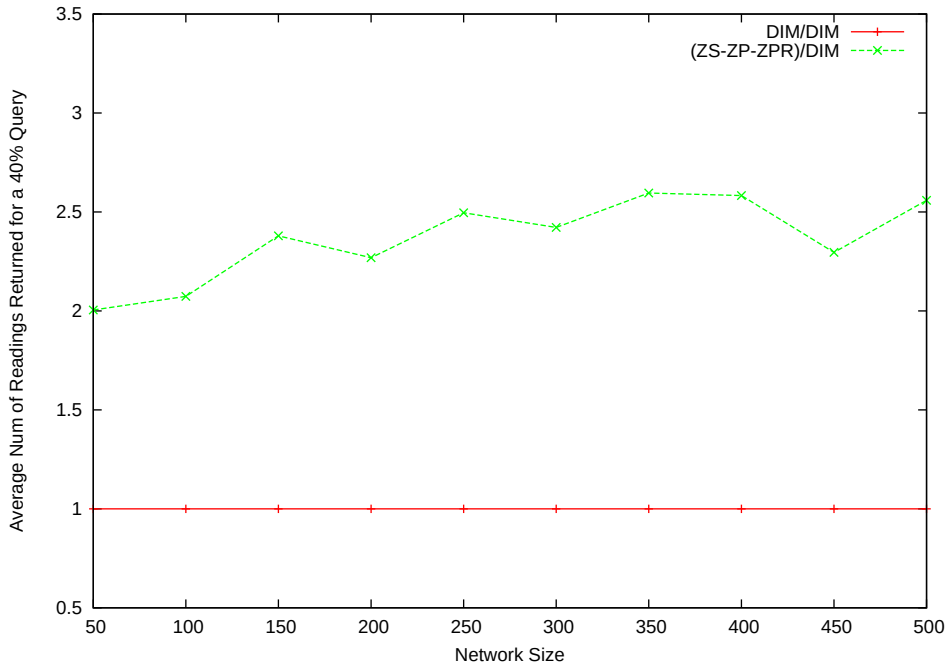


(a) 60% Multiple Mixed Hotspot

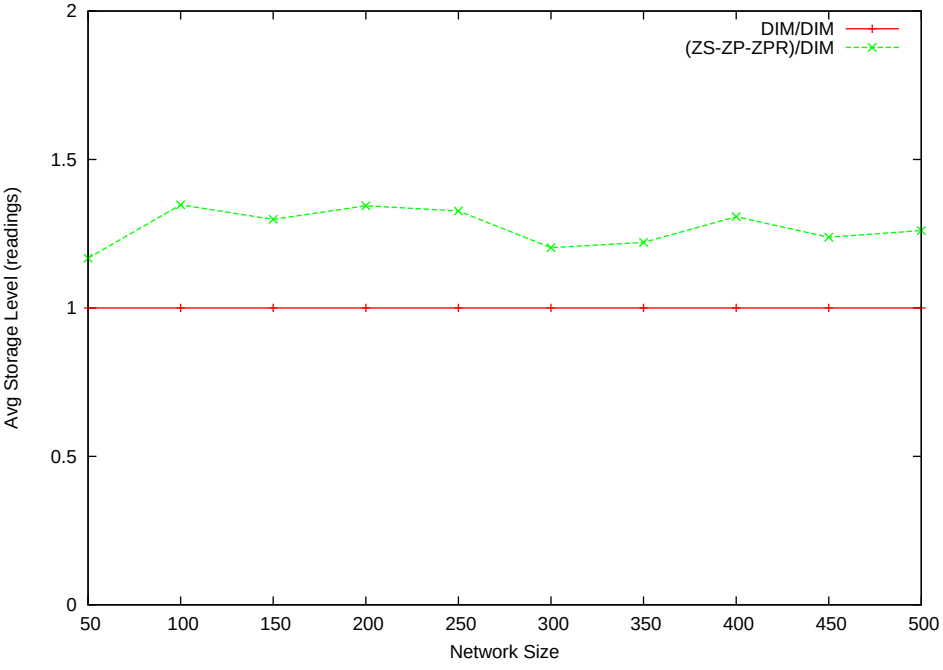


(b) 80% Multiple Mixed Hotspot

Figure 47: ZS/ZP/ZPR: Dropped Events for Multiple Mixed Hotspots



(a) 40% Query for an 80% Multiple Mixed Hotspot



(b) Average Node Storage for a 60% Multiple Mixed Hotspot

Figure 48: ZS/ZP/ZPR: QoD Graphs for Multiple Mixed Hotspots

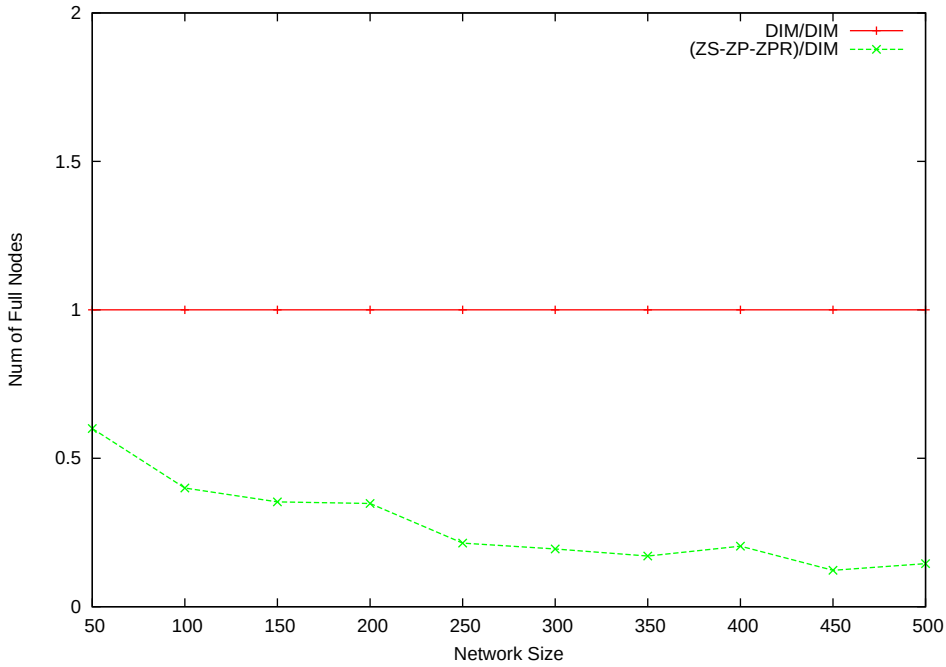
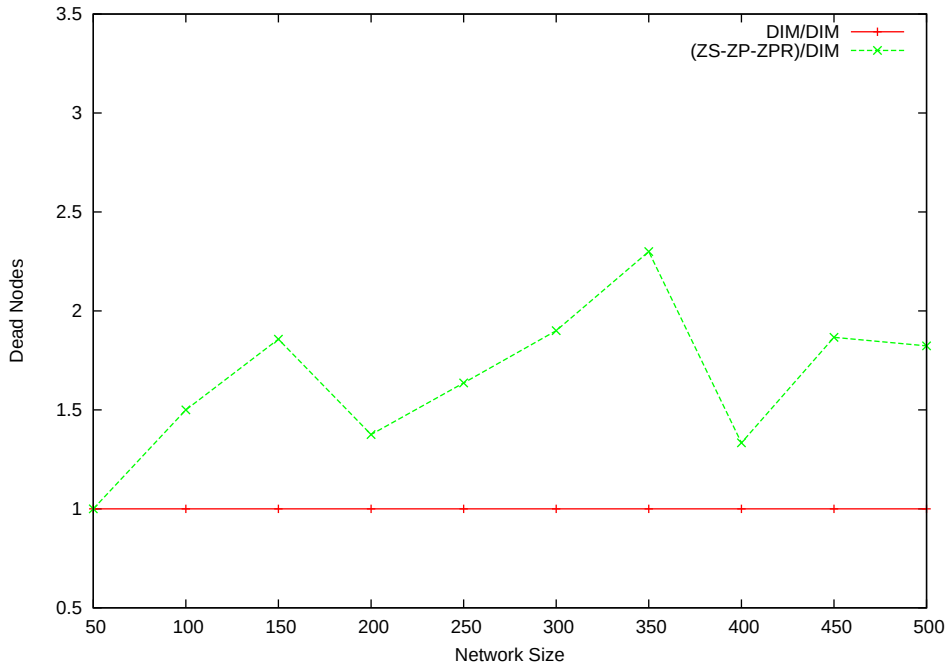


Figure 49: ZS/ZP/ZPR: Number of Full Nodes for 60% Multiple Mixed Hotspot

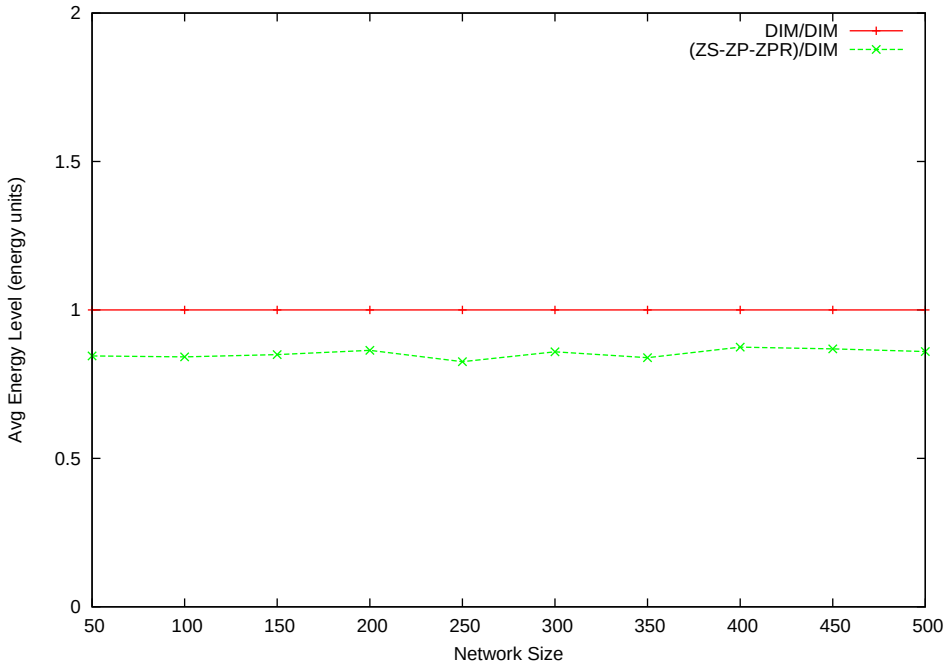
48(b) shows that ZS/ZP/ZPR achieves around 30% increase in the average node storage when facing 60% multiple mixed hotspots. All four figures collectively show the high QoD improvement that ZS/ZP/ZPR achieves compared to DIM. The results are valid for hotspots of sizes up to 80%.

R2. load balancing: Figure 49 shows that ZS/ZP/ZPR highly improves the DIM storage load balancing by highly reducing the number of full nodes when facing 60% multiple mixed hotspots. As in the case of single mixed hotspots, ZS/ZP/ZPR achieves a constant number of full nodes as opposed while that of DIM is proportional to the network size.

R3. Energy Consumption: Figures 50(a) and 50(b) compare the two schemes in terms of the number of dead nodes and the average node energy when facing 80% multiple mixed hotspots, respectively. The two figures shows that ZS/ZP/ZPR imposes a moderate energy consumption overhead imposed compared to the DIM scheme. ZS/ZP/ZPR increases node deaths by around 3% of the network size when compared to DIM. In terms of energy consumption overhead, ZS/ZP/ZPR imposes an increase of about 12% over DIM.



(a) Dead Nodes



(b) Average Node Energy

Figure 50: ZS/ZP/ZPR: Energy Consumption Graphs for 80% Multiple Mixed Hotspots

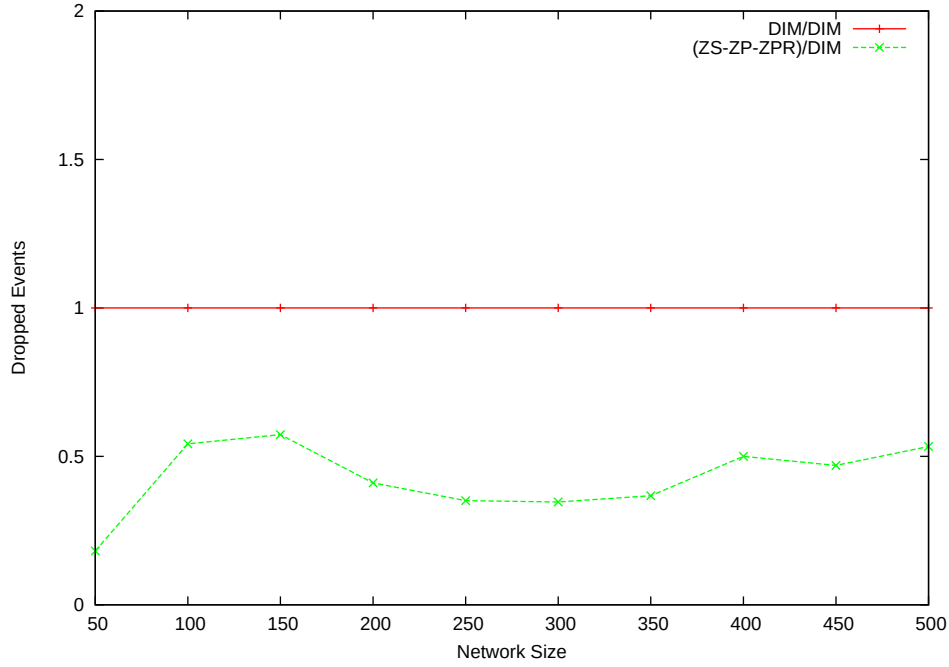
In conclusion, ZS/ZP/ZPR helps DIM in coping with multiple mixed hotspots. Towards that, ZS/ZP/ZPR improves DIM's QoD by 50% to 70% while imposing a 12% energy consumption overhead on DIM. The relative cost of the QoD improvement the case of multiple mixed hotspots is slightly higher than that in the case of single mixed hotspots.

4.3.2.4 Moving Mixed Hotspots The following three results compare the ZS/ZP/ZPR performance to that of the DIM for moving static mixed hotspots. Comparison is based on QoD (R1), load balancing (R2), and energy consumption (R3). We simulated an x% moving hotspot as follows. Each run has been divided into 5 steps. The hotspot starts in range $[t_1, t_1 + i]$ in the first step, then moves on to $[t_1 + i, t_1 + 2i]$ in the second step, etc. In each of the steps, x% of the generated readings/queries fall in the step's hotspot range. We simulated hotspot sizes up to 50%.

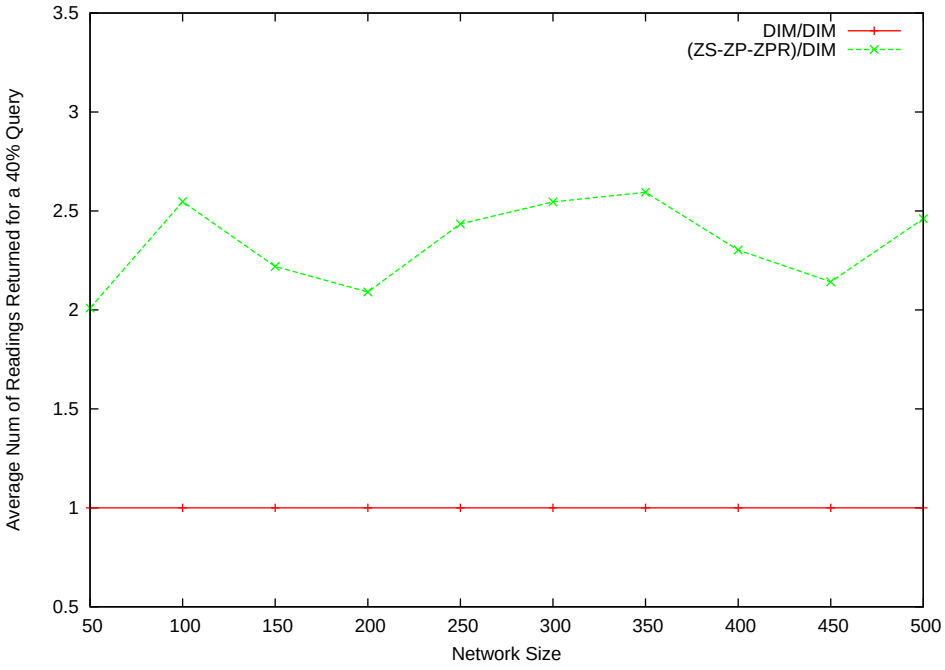
R1. QoD: Figure 51(a) shows that ZS/ZP/ZPR achieves around 50% QoD improvement compared to DIM by reducing the number of dropped events when facing 80% moving mixed hotspots. Similarly, Figure 51(b) shows that ZS/ZP/ZPR highly at least doubles the average result size of a 40% query compared to DIM. Both figures show that ZS/ZP/ZPR achieves a good QoD improvement for mixed hotspots when implemented on top of DIM. It should be noted that the improvement is less than those in the cases of single and multiple hotspots. However, it is performing much better than ZS and ZP/ZPR against moving storage and query hotspots, respectively.

R2. load balancing: Figure 52 compare the load balancing ability of the two schemes in terms of the number of full nodes for 60% moving mixed hotspots. The figure shows that ZS/ZP/ZPR improves the DIM load balancing by reducing considerably the number of full nodes by around 70% compared to DIM. This shows the ZS/ZP/ZPR ability to load-balance the data falling in moving mixed hotspots across the sensor network.

R3. Energy Consumption: Figure 53(a) and 53(b) compare ZS/ZP/ZPR and DIM in terms of dead nodes and average node energy when facing 60% and 80% moving mixed hotspots, respectively. The first figure shows that ZS/ZP/ZPR increases the number of dead nodes by around 5% of the network size. The second figure shows that the drop in the average node energy is around 12% per node. Together, both figures show that ZS/ZP/ZPR



(a) Dropped Events



(b) 40% Query

Figure 51: ZS/ZP/ZPR: QoD Graphs for an 80% Moving Mixed Hotspot

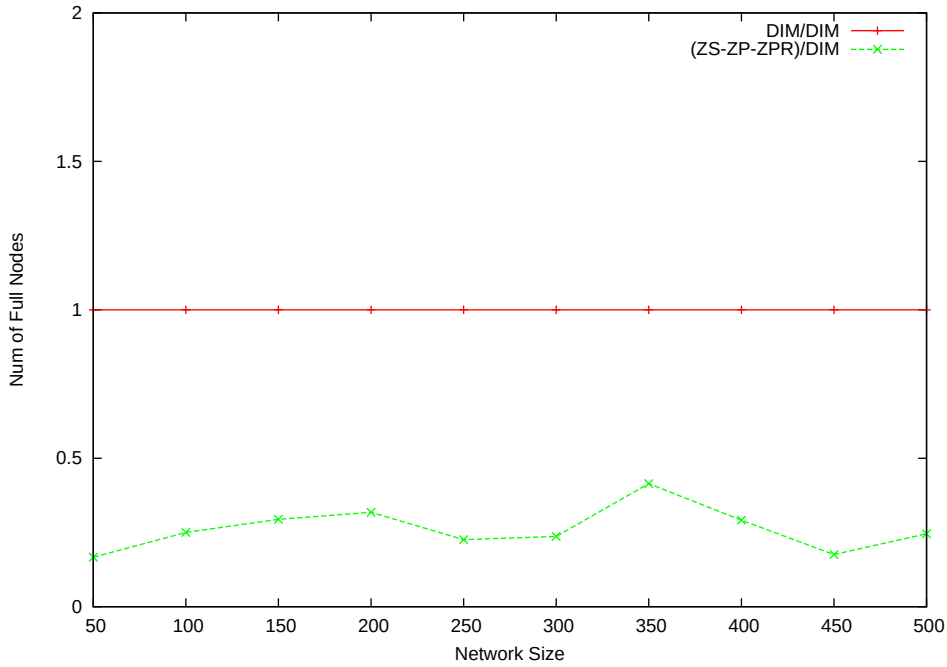
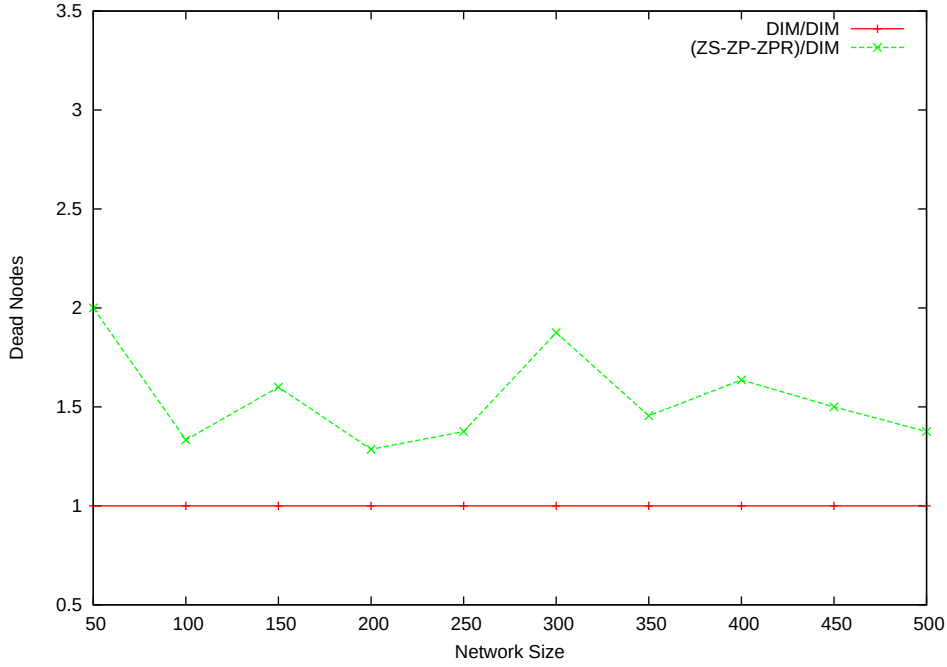


Figure 52: ZS/ZP/ZPR: Number of Full Nodes for 60% Moving Mixed Hotspot

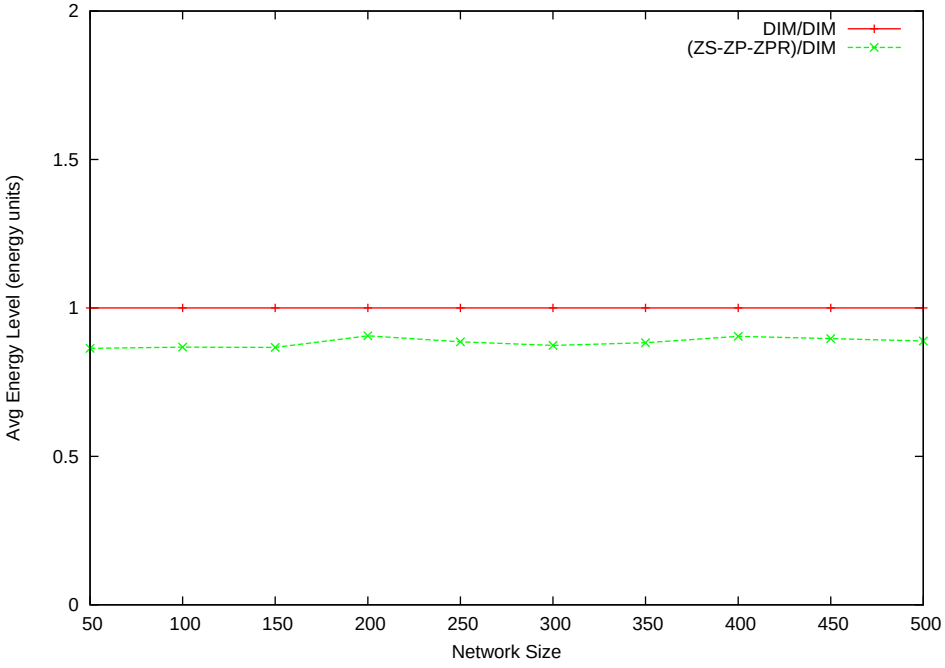
adds a moderate energy consumption overhead when implemented on top of DIM and facing moving mixed hotspots. The overhead is comparable to that imposed in the cases of single and multiple mixed hotspots.

The above three results show that ZS/ZP/ZPR is able to improve the performance of DIM when facing moving mixed hotspots. The QoD improvements are around 50% and the energy consumption overhead is around 12%. The relative cost of the QoD improvement slightly higher than those experienced by ZS/ZP/ZPR in the cases of single and multiple mixed hotspots.

4.3.2.5 Discussion In conclusion, ZS/ZP/ZPR is able to cope with mixed hotspots with the drawback of imposing a moderate energy consumption overhead on the different network nodes. QoD achievements are ranging from 70% in the case of single hotspots, 50% to 70% in the case of multiple hotspots, and 50% in the case of moving hotspots. This comes with an energy consumption overhead of around 12% for all hotspot types. Table 5 summarizes



(a) Dead Nodes for a 60% Moving Mixed Hotspot



(b) Average Node Energy for an 80% Moving Mixed Hotspot

Figure 53: ZS/ZP/ZPR: Energy Consumption Graphs for Moving Mixed Hotspots

Table 5: ZS/ZP/ZPR Performance (Relative to DIM) for Mixed Hotspots

Hotspot Type	QoD Improvements	QoS Overheads
Single Static Hotspots	70%	12%
Multiple Static Hotspots	50% – 70%	12%
Moving Hotspots	50%	12%

the ZS/ZP/ZPR performance for mixed hotspots.

4.3.3 Learned Lessons from the Experimental Evaluation

In this section, we will discuss some of the insights than can be taken from the experimental evaluation of our ZS, ZP, ZS/ZP/ZPR schemes. Our main goal is to study the benefits of implementing our schemes versus their implementation overhead. Also, we would like to determine the environments that are most fitting for our schemes, in terms of network size, hotspot types and sizes, etc.

Before discussing the learned lessons out of our experimental evaluation, we summarize our results in the following points:

1. Our schemes are best suited for decomposing single hotspots while introducing a tolerable energy consumption overhead on the sensor nodes across the network.
2. For single hotspots, QoD improvements are around 50% for single storage hotspots (using MHZS), 25% for single query hotspots, and 75% for single mixed hotspots while energy consumption overheads are 5%, 7%, and 12% respectively.
3. For multiple hotspots, QoD improvements are around 20% for single storage hotspots (using SHZS), 20% for single query hotspots, and 50% to 70% for single mixed hotspots while energy consumption overheads are 2%, 5%, and 12% respectively.
4. For large-scale hotspots (especially those larger than 80%) and moving hotspots, the performance improvements of our schemes, in terms of QoD, become limited. Additionally, the energy consumption overhead becomes relatively large (above 15%). This mainly

results from the inefficiency of the local hotspot decomposition strategy followed by our schemes against complex hotspot settings.

In general, experiments have shown that the schemes achieve good results in terms of improving the QoD of DIM for the different types of hotspots. The QoD improvements of our schemes were varying depending on the hotspot type. In general, our schemes perform better when facing single hotspots rather than when facing multiple or moving hotspots. This result can be explained by the fact that our schemes do not introduce any major changes to the underlying DCS index structure. As our schemes do not implement any global load balancing scheme that coordinates the hotspot load balancing across the network, their ability to deal with hotspots arising in different locations is limited. Additionally, the local hotspot decomposition might introduce additional hotspots in the network if the dissipation of the data of the different hotspots leads them to a single new location which becomes the new hotspot area.

The network size was another important factor in determining the efficiency of our schemes. For most of the hotspot types, our schemes did not add much benefit for small networks (less than 150 nodes). The effect of hotspots is not considered relatively high on these networks in terms of the negative effects of hotspots on QoD. Therefore, the addition of our schemes did not help DIM perform any better. The effect of our schemes became to appear for sensor networks of medium sizes (150 to 350 nodes). This effect became very obvious for larger networks.

Concerning hotspot sizes, the performance improvements of our schemes were very limited for small hotspot sizes (less than 40%). This is due to the fact that the negative effects of such hotspots on the DIM performance are already limited. Additionally, the relatively short durations of our experiments helped in further reducing the negative effects of small hotspots. Therefore, it was hard to achieve a tangible performance improvement for these small hotspot sizes out of implementing our schemes. Our schemes started to score better improvements when hotspot sizes became larger, starting from 50% to 80%. As the negative effects of these hotspots are very obvious on the DIM performance, the addition of our schemes improved the DIM ability to deal with hotspots. Decomposing hotspots of larger sizes became a problem for our schemes. This is due to the fact that the amount of events

falling in these hotspots became too large for our local schemes to compete with. We believe this is mainly due to the fact that our schemes do not introduce load balancing to the underlying index structure. This definitely limits their gains against large hotspots.

The energy consumption overhead is also another important factor to take into account when deciding to implement any of our schemes. As our results show, the overhead is low for uniform loads. This is mainly due to piggy-backing the information needed by the DMC and PC on the regular messages sent within the network. This helps our schemes in introducing a small overhead throughout the network operation. For single hotspots, the overhead was low to moderate for most of the cases. It mainly comes from the continuous migration of readings in the network. Note that the zone sharing and partitioning processes continuously take place in order to increase the DIM ability to deal with hotspots. This overhead is higher for multiple hotspots regardless whether they are static or dynamic due to the increase in the energy consumed by the different network nodes in data migration.

One of the limitations of our experimental evaluation is that we did not consider simultaneous updates and queries. This limited our ability to determine the effect of such a case on the correctness of query results for the ZPR scheme. However, as we discussed earlier, our experimental evaluation scenario and our hotspot formation technique was fitting for the ZP scheme rather than for the ZPR scheme. We believe that, in real-world scenarios, the application of the ZP scheme would be more frequent than that of the ZPR scheme. We plan for validating this assumption when experimenting our schemes on sensor network testbeds in the near future.

In general, we believe that our schemes are fitting medium-sized networks that are expected to decompose single hotspots of medium sizes. Due to the local hotspot decomposition technique of our schemes, they are not suited to deal with large and consistent hotspots nor with multiple hotspots (both static and dynamic). Our local schemes are well fitting for search/discovery sensor network applications, where limited hotspots may exist and solving them is required not to consumer a lot of energy from the sensor network. Chapter 5 presents our KDDCS scheme whose global load balancing technique is perfectly fitting to deal with these hotspot settings.

4.4 SUMMARY

In this chapter, we presented a set of local schemes to detect and decompose different types of hotspots imposed on DCS sensor networks, including storage, query, and mixed hotspots. Our schemes use the concept of data migration to decompose hotspots without changing the underlying index structure of the DIM scheme. Our experimental results show that our schemes achieve good QoD and load balancing achievements, especially against single hotspots, while adding a tolerable energy consumption overhead on the different sensor nodes. The performance improvements of our schemes become limited in the cases of large-scale single hotspots and multiple hotspots (both static and dynamic). An important benefit of our local schemes is their low messaging overhead compared to the DIM scheme. Another advantage of our schemes is their ability to be applied in sensor networks with different storage capacities. The next chapter will present our KDDCS scheme to improve the sensor network performance against against these cases.

5.0 HOTSPOT AVOIDANCE

In the previous chapter, we have presented a set of local schemes to detect and decompose hotspots of various types. Though being effective, our schemes were limited in their ability to cope with hotspot once their sizes and/or numbers increase. In this chapter, we provide a load-balanced in-network DCS scheme based on k-d trees, that we, not surprisingly, call *K-D tree based DCS (KDDCS)*. KDDCS has the ability of avoiding hotspots of different types by maintaining the underlying k-d tree balanced throughout the network operation. We present the details of KDDCS in the following sections.

5.1 OVERVIEW ON KDDCS

The main design goal of KDDCS is to avoid the formation of hotspots. We define the *hotspot avoidance* to be the ability to deal with the hotspot in its early stages and successfully disseminating it before it severely arises. Similar to the DIM scheme, our KDDCS scheme stores data in-network by mapping the sensor nodes to the leaves of a k-d tree. In KDDCS, the refinement of regions in the formation of the k-d tree has the property that the numbers of sensors on both sides of any 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 be uniformly distributed over the sensors if the events are uniformly distributed over the range of possible events. We present a modification of Greedy Perimeter Stateless Routing (GPSR) [32] routing, namely *Logical Stateless Routing (LSR)*, for the routing of events from their generating sensors to their storage sensors. LSR is competitive with the GPSR routing scheme 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 rebalance the tree throughout the network lifetime. We use the KDTR algorithm to avoid the formation of storage and/or query hotspots.

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 sensor nodes. In general, the implementation overheads (including the memory and processing overheads) of the different KDDCS components are very suitable for the limited resources of sensor nodes (especially for motes).

Experimental evaluation shows that the main advantages of KDDCS, when compared to the plain DCS schemes, such as DIM and GHT, as, well as to our previously presented local hotspot detection and decomposition schemes including ZS, ZP/ZPR, and their mix ZS/ZP/ZPR, are:

- Coping with hotspots of all types and sizes, especially those of complex structure such as multiple and moving hotspots.
- Increasing the *Quality of Data (QoD)* by distributing the hotspot events (readings and/or queries) among a larger number of sensors.

- Improving *data persistence* by balancing the storage responsibility among sensor nodes.
- Achieving a comparable *Quality of Service (QoS)* by imposing low to moderate energy consumption overheads compared to the other schemes while balancing energy consumption overheads among the network nodes.

The rest of this chapter is organized as follows. Section 2 highlights 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 rebalancing algorithm. Sections 6, 7, and 8 describe how KDDCS avoids storage, query, and mixed hotspots, respectively. Section 9 highlights the packet loss handling mechanism in KDDCS. Section 10 discusses the learned lessons out of the experimental evaluation. Finally, Section 11 concludes the chapter.

5.2 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. We define an event to be either a reading or query.

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 storage-sensor* mapping that determines which sensor will store a reading or answer a query. 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 storage 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 this phase, each sensor locally determines its binary address by uniformly splitting the overall service area in a round 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 sensor readings 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 reading are 1001. This is because temperature is in the top half of the range $[30, 70]$, pressure is in the bottom half of

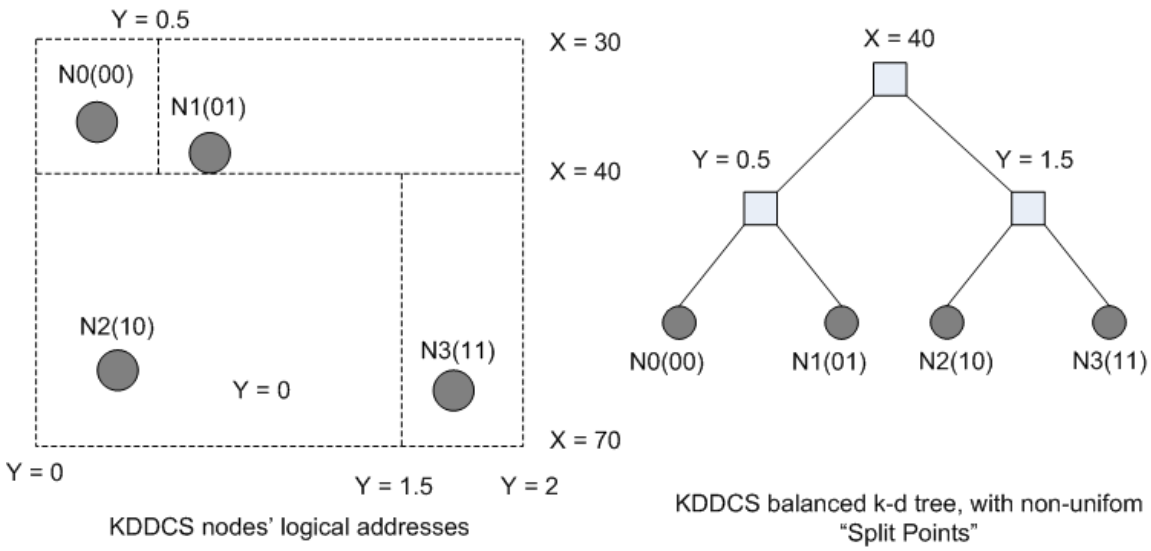


Figure 54: KDDCS k-d Tree

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 reading should be routed toward the geometric location specified by code 1001.

In DIM, events are routed using GPSR. An event is routed to the geographic zone with an address matching the high order bits of the reading bit-code. In our example, the sensor 10 will store this reading 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.

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 54 for the same simple input shown in Figure 1. The *second* difference is that in determining the storage sensor for

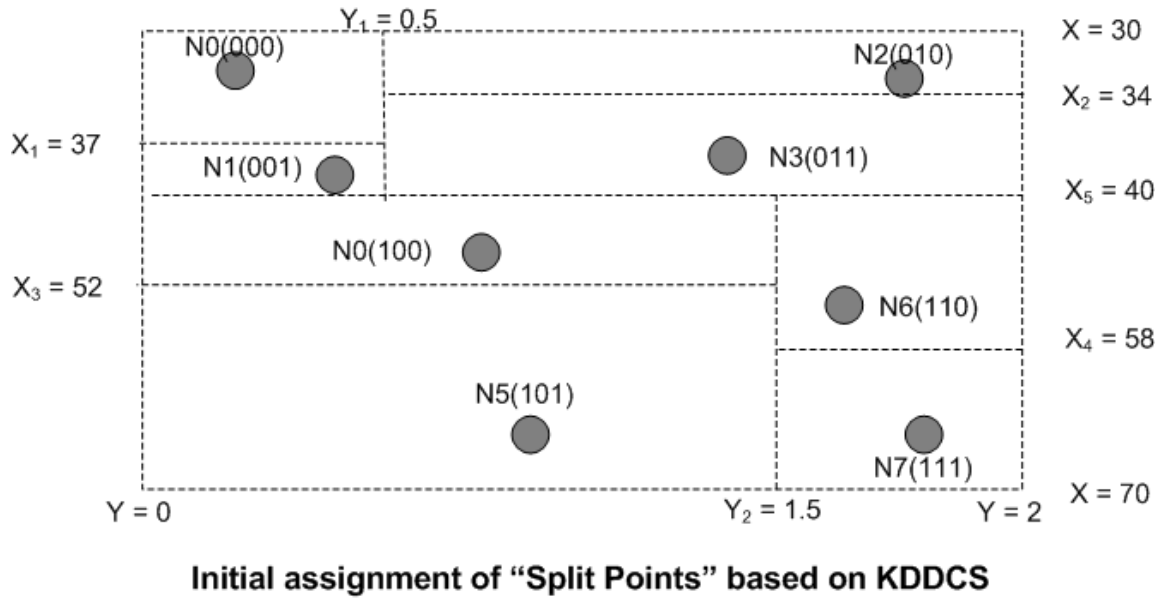


Figure 55: KDDCS Initial k-d Tree

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 readings in the ranges $[a, b]$ and $[b, c]$. Thus, in KDDCS, the storage of readings 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. Figure 55 shows another example for a larger sensor network together with the k-d tree formed by KDDCS for this network.

5.3 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 initiates a process to collect 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 the service area. 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, or 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 sensor has to send is $O(\log^2 n)$ since there are likely only $O(\log n)$ candidates on the first step and only one 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 of values more than the computed median V can not be much more than $L/2$.

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.

5.4 THE KDDCS COMPONENTS

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 storage sensors.

5.4.1 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 56. We now describe in some greater detail how a recursive step works.

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

Figure 56: Logical Address Assignment Algorithm

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 $O(\log^2 n)$.

5.4.2 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 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 for storing events falling in the temperature range $(30, 65)$, while the right child subtree will store events falling in the range $(65, 70)$. Figure 54 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. This can be represented by the geographic coordinates of the upper left corner (x_1, y_1) and those of the lower right corner of this region, (x_2, y_2).
- The split line separating its two children (either $x = k_1$ or $y = k_1$ where $x_1 < k_1 < x_2$ and $y_1 < k_2 < y_2$).
- 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.

5.4.3 Incremental Event Hashing and Routing

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

In our scheme, the readings-to-sensors mapping provides a logical address. Essentially, all that the sensor generating the reading can determine from this logical address is the general direction of the storage 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 a reading v_1 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 be the split line corresponding to this region, and R_0 and R_1 be the two sub-regions 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.

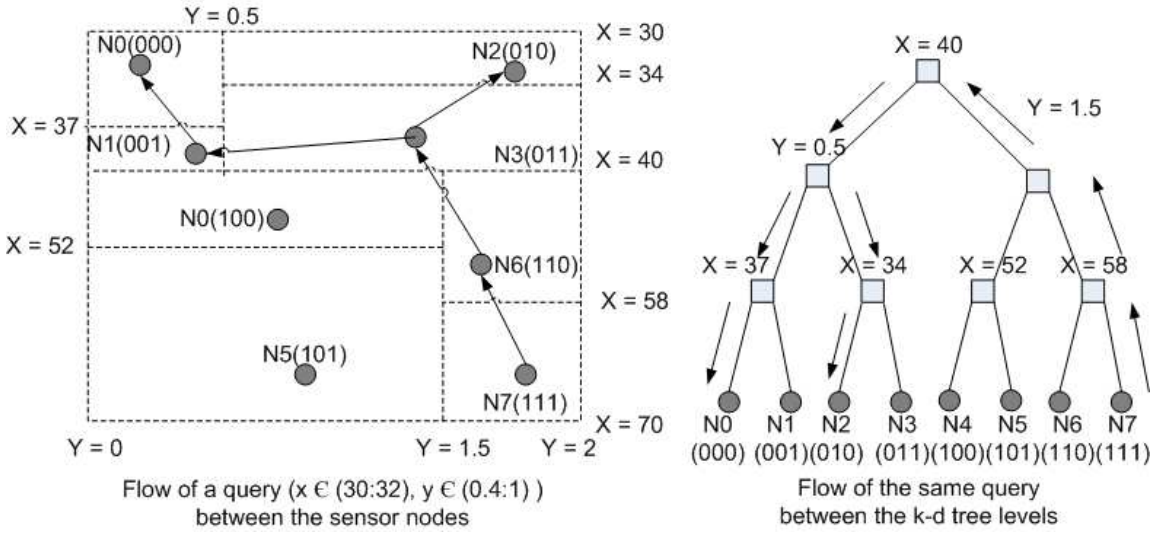


Figure 57: Example of Routing a Query on KDDCS

We explain how range queries are routed by means of an example. This example also essentially illustrates how readingss are stored. Figure 57 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 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 sub-query, 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 57, and a possible physical routing of this query is shown on the left in Figure 57.

As LSR does not initially know the geometric location of the storage sensor, the route to the storage 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.

5.4.4 Discussion

To summarize, KDDCS is mainly composed of three components: the initial distributed logical address assignment algorithm, the event to bit-code mapping, and the incremental event hashing and routing scheme. The first component is responsible for initializing the sensor logical addresses to form a balanced k-d tree (with sensors as leaves). It is based on distributively solving the weighted split median problem. The event to bit-code mapping is responsible for distributively forming the initial k-d tree over the possible attribute ranges of readings and assigning each sensor a storage responsibility range. Finally, the incremental event hashing and routing is responsible for incrementally routing each reading to its storage sensor for each reading (based on the attribute values of that reading).

5.5 KDTR: K-D TREE REBALANCING ALGORITHM

Based on the KDDCS components presented in the previous section, KDDCS already avoids the formation of storage hotspots resulting from skewed sensor deployments. Also, KDDCS avoids the formation of storage hotspots resulting from skewed reading distributions assuming that the distribution of sensor readings was known a priori. However, storage hotspots may be formed if the initial presumed reading distribution was incorrect or if the reading distribution evolves over time. Similarly, query hotspots may be formed if the query load distribution is skewed. In this section, we present the K-D Tree Rebalancing algorithm (KDTR) to rebalance the load on sensors throughout the network operation. This load can either be storage load, query load, or mixed load.

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

We will present the KDTR algorithm in the context of storage hotspots. In the following sections, we will discuss how to use KDTR to avoid both query and mixed hotspots.

5.5.1 Selection of Subtrees to be Rebalanced

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 readings in one of its child subtrees over the number of readings stored in its other child subtree exceeds some threshold h. This process of identifying nodes to rebalance proceeds in $O(\log n)$ rounds from the leaves to the root of the k-d tree.

We now describe round $i \geq 1$ intuitively occurring 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 sub-regions 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 readings 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 readings 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 $O(d_i \log n)$.

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

5.5.2 Tree Rebalancing Algorithm

Let x be an internal node to the k-d tree that needs to be rebalanced. 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 readings stored at sensor i, and the values are the j'th attributes of the w_i readings stored at that sensor. Thus, the computed attribute split L has the property that, in expectation, half of the readings stored in R have their j'th attribute larger than L, and half of the readings stored in R have their j'th attribute smaller than L.

Let R_0 and R_1 be the two sub-regions 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 readings 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 readings need to be routed from R_0 to R_1. Consider a reading 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 reading e is then routed to t using the routing scheme described in section 5.4.3. The final storage sensor 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 readings 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 that load is balanced between R_0 and R_1. This process can now be recursively applied to R_0 and R_1.

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 readings. 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 periodical rebalance requires each reading to travel at most twice the diameter of the network (assuming that GPSR routes on a direct line). The total number of readings that can be forced to migrate as a result of k new readings being stored is $O(k \log k)$. Thus, the amortized number of migrations per reading is logarithmic, $O(\log k)$ in the number of insertions. This amount of rebalancing per insertion is required for any standard dynamic data structure (e.g. 2-3 trees, AVL trees, etc.).

Figure 58 shows a detailed example illustrating how KDTR works on the sensor network of Figure 55. 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 readings stored at each sensor, a storage hotspot arises in node $N7$ after 6 reading 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 are unbalanced at this point, KDTR selects 11 to be rebalanced. However, the storage load remains skewed toward subtree 1, thus, after another 6 insertions, KDTR rebalances the subtree rooted at 1. After 12 more insertions aiming the right subtree of the root, KDTR rebalances 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 hotspot area increases, the frequency of rebalancing decreases.

We digress slightly to explain a method that one might use to trigger rebalancing, as opposed to fixed time period rebalancing. Each sensor s_i knows the number of readings that are stored in each region corresponding to an ancestor of s_i in the k-d tree when this region

was rebalanced. Let C_j be the number of readings at the last rebalancing 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 readings that have to be stored in R_j, since the last rebalancing, to cause another rebalancing 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 rebalancing might be warranted. Note that the role of leader requires only receiving $O(\log n)$ messages.

5.5.3 Discussion

In summary, the KDTR algorithm is composed of two components: the selection of subtrees to be rebalanced and the actual tree rebalancing algorithm. As for the selection for the trees to be rebalanced, the main idea is find the highest unbalanced node(s) in the k-d tree. This can be done in $O(\log n)$ rounds with a total time complexity of $O(D \log n)$, where D is the diameter of the network. Again, the tree rebalancing algorithm is based on distributively solving the weighted split median problem.

It is important to note that, in addition to load balancing, the application of the KDTR algorithm achieves two very important benefits. The first is that the continuous rebalancing of the k-d tree allows KDDCS to continuously check for node failures and deal with them by rebalancing the storage load among the available nodes in the network at the time of rebalancing. This acts as a first level of fault tolerance for KDDCS. A second advantage is that the continuous rebalancing allows the LSR algorithm to use different routing paths for the same request type as time progresses. This has the advantage of reducing parts of the negative effects of traffic skewness across the sensor network.

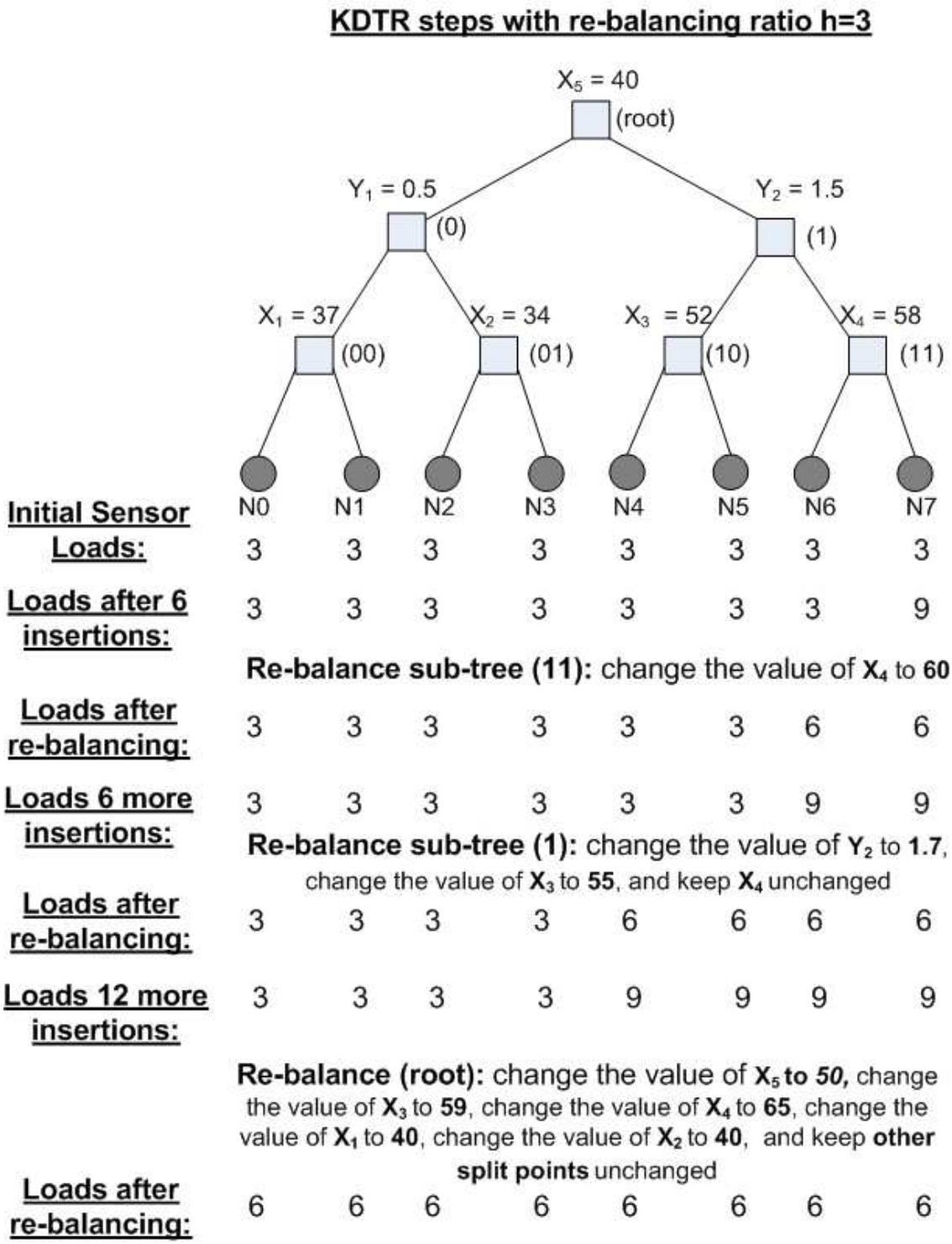


Figure 58: KDTR Example

5.6 AVOIDING STORAGE HOTSPOTS WITH KDTR

As was discussed in the previous sections, the KDDCS scheme will be able to avoid the formation of storage hotspots without applying the KDTR algorithm assuming that the storage load distribution was known, at least to a constant factor, prior to the network operation and that this distribution should be static, i.e., not changing over time. In case the two assumptions cannot be satisfied, the KDTR algorithm, as presented in the previous section, will take the responsibility of avoiding storage hotspots. The KDTR algorithm performs such function through the periodic global rebalancing of the underlying k-d tree of the KDDCS scheme. This rebalancing is performed in a parallel and distributed manner. Its periodicity guarantees the decomposition of any potential hotspot as soon as its indications start to kick-in, and thus, the avoidance of any hotspots.

As the rebalancing in KDTR used the storage load of sensor nodes as its metric, the above KDTR version was geared to avoid storage hotspots. We experimentally study the performance of the KDTR algorithm when facing storage hotspots in the next subsection.

5.6.1 Experimental Evaluation

In this section, we study the performance of KDDCS for storage hotspots. We compare KDDCS with DIM and DIM with ZS for single static storage hotspots (5.6.1.1), multiple static storage hotspots (5.6.1.2), and moving storage hotspots (5.6.1.3).

Throughout this section, the node storage capacity is equal to 30 readings and the node initial energy capacity is equal to 70 units. Therefore, a *full sensor node* is defined to be a sensor node having 30 readings in its cache. Similarly, a node is depleted (and consequently considered dead) as soon as it consumes 70 energy units. Once a node is dead, all readings stored in this node are considered lost. Based on the DIM scheme, the storage responsibility (a subset of the attribute range) of the dead node is assigned to one of its direct neighbors. Recall that we define an event to be either a reading or a query.

For each hotspot type, we conduct experiments for single, multiple, and moving hotspots. For each of our experiments, we compare the performance of KDDCS for the LS scheme, the

GHT scheme, the basic DIM scheme, and the DIM scheme with the ZS scheme implemented on top of it (both SHZS and MHZS).

For each of our experiments, we study three aspects: QoD (R1), load balancing (R2), and energy consumption (R3). For the QoD, we study the number of dropped events (readings/queries) and the average node storage. We refer to the percentage of QoD improvement to be the percentage of decrease in event drops. For the load balancing, we study the number of full nodes. As for energy consumption, we study the average node energy and the number of dead nodes. The average node energy is the one that defines the improvement or the downgrading in energy consumption performance. To be statistically significant, we conducted 5 simulation runs for each of the experiments and taken the average of values across all runs.

For each of the hotspot types, we conducted experiments on different hotspot sizes ranging from 20% to 100%. When presenting our results, we always present the results for the largest hotspot size on which the scheme performed well. Unless otherwise stated, performing well on the large hotspot sizes, i.e., $[60\%, 80\%]$, implies a good performance on the moderate sized hotspots, i.e., $[40\%, 60\%]$. In most of the cases, the performance burden imposed to the network by small hotspots, i.e., hotspots less than 40%, does not justify the cost paid to detect and decompose the hotspots.

To model the worst case performance of KDDCS, we set the scheme to initially start every experiment with a uniform distribution of attribute ranges on sensor nodes. This means that the initial storage responsibility for each of the sensor nodes would be the same for both KDDCS and DIM. The reason behind this selection is to model the efficiency of the KDTR algorithm in dealing with hotspots in cases where the ranges of these hotspots cannot be anticipated in advance, prior to the network operation. Of course, initializing the network with a distribution which partially or fully anticipates the ranges of the hotspots or the distribution of the readings to be stored in the sensor network would boost the KDDCS performance over all other schemes. However, this case may only be realistic if load distributions are known prior to the network operation. As this may not be the common case, we decided to model the general case which would help us analyze the worst case KDDCS performance.

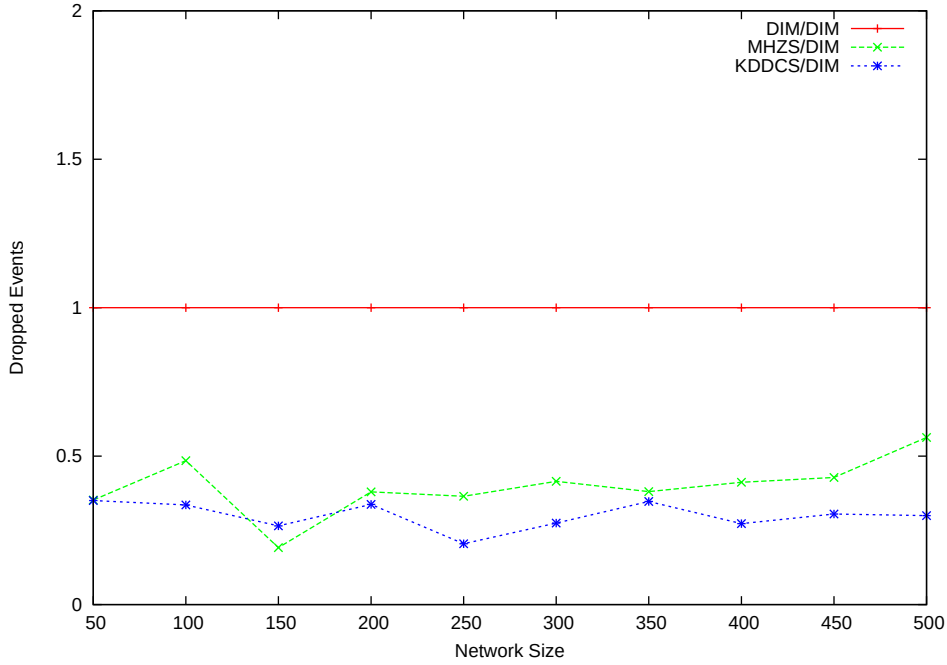
Before presenting the results of our experiments, we highlight the learned lessons out of our experimental evaluation in the following points:

1. KDDCS scores good QoD achievements for the different hotspot types and sizes while introducing a tolerable energy consumption overhead on the sensor nodes across the network.
2. For single storage hotspots, QoD improvements are around 75% while energy consumption overheads are 1%.
3. For multiple static storage hotspots, QoD improvements are around 25% while energy consumption overheads are 1%.
4. For multiple dynamic (moving) storage hotspots, QoD improvements are around 60% while energy consumption overheads are 13%.
5. The performance improvements of KDDCS scale proportionally with the network size.

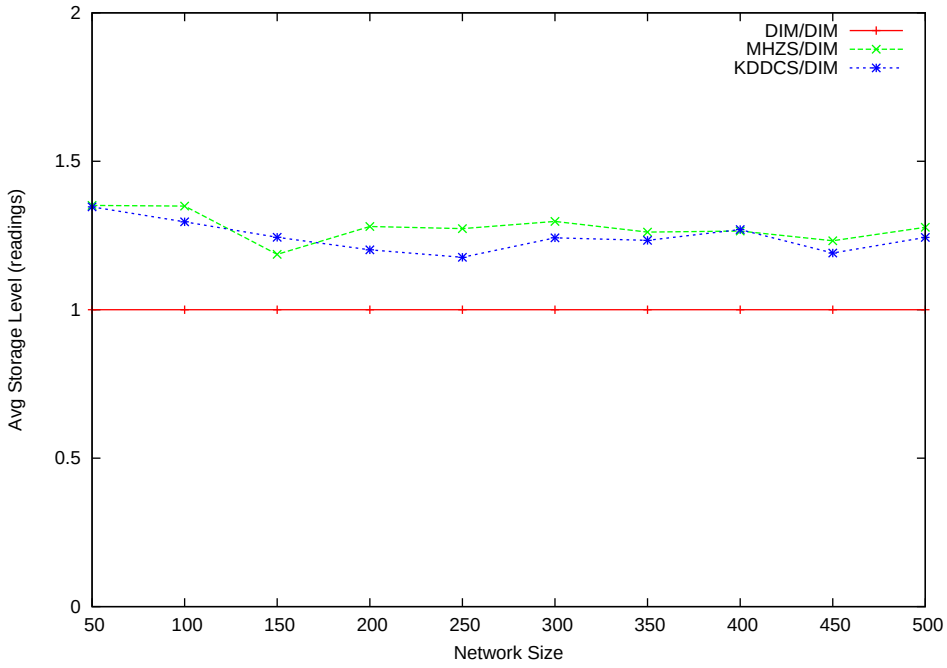
Experimental results are shown in Figures 59 to 67. Recall that DIM already outperforms both the LS and the GHT schemes. Also, we only plot the MHZS performance as a representative of the ZS scheme. That's why we only plot the KDDCS performance together with those of the DIM and DIM/MHZS schemes. For simplicity, we refer to the DIM/MHZS scheme by MHZS.

Though we tested a range of values for the rebalancing ratio h, we only present results corresponding to $h = 2$ to avoid repetition. In general, changing the value of h did not have a major effect on the KDDCS performance. This is due the type of hotspots that we simulated. As our hotspots usually fall in relatively small subranges of the possible attribute ranges, satisfying the rebalancing criteria of the KDTR algorithm only requires a small h value in most of the cases. This results in almost no difference in performance between h values ranging from 1.5 to 3. Concerning the MHZS parameters, we use their default values already presented in Chapter 4.

5.6.1.1 Single Static Storage Hotspots The following three results compare the KDDCS performance compared to those of DIM and MHZS for single static hotspots.



(a) 80% Single Storage Hotspot



(b) 80% Single Storage Hotspot

Figure 59: KDDCS: QoD Graphs vs Single Storage Hotspots

R1. QoD: Figure 59(a) compares the KDDCS performance to those of DIM and MHZS schemes in terms of the number of dropped events for an 80% single storage hotspot. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The figure shows that KDDCS and MHZS have comparable event drops for all network sizes. Both schemes improve the QoD of the basic DIM scheme by around 75%. The main observation in the figure is that the performance of KDDCS scales with the network size in a way that keeps the same percentage of QoD improvement for all network sizes. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

Figure 59(b) compares the performance of the three schemes in terms of the average node storage for an 80% single storage hotspot. Recall that the average node storage is measured in terms of sensor readings. The figure shows that KDDCS increases the average node storage achieved by DIM by around 30% for all network sizes. KDDCS performs comparably to MHZS for all network sizes.

Overall, the above figures show that KDDCS highly increases data persistence as compared to DIM. Though KDDCS may perform similarly to MHZS in this small-scale simulation, the performance of KDDCS may highly exceed that of MHZS in the case of persistent hotspots that span a large duration of the network operation.

R2. Load Balancing: Figure 60 shows the number of full nodes of the three schemes for an 80% single storage hotspot. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The figure shows that KDDCS highly exceeds DIM in terms of number of full nodes. The number of full nodes for KDDCS is around 30% times larger than that for DIM. On the other hand, KDDCS decreases the number of full nodes compared to MHZS by around 5% to 20% (Recall that all percentages are related to DIM). As discussed in R1, this is due to the larger number of nodes KDDCS devotes to store the hotspot data. Successfully maintaining the hotspot data within the sensor caches while distributing them among a larger number of sensor nodes results in decreasing the number of nodes reaching their full capacity. It is important to note

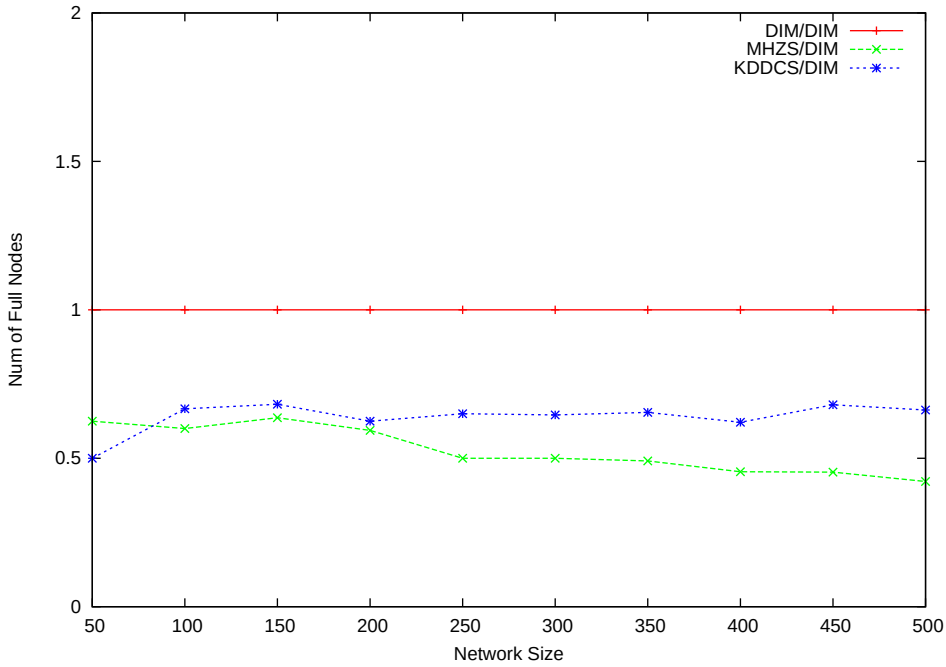
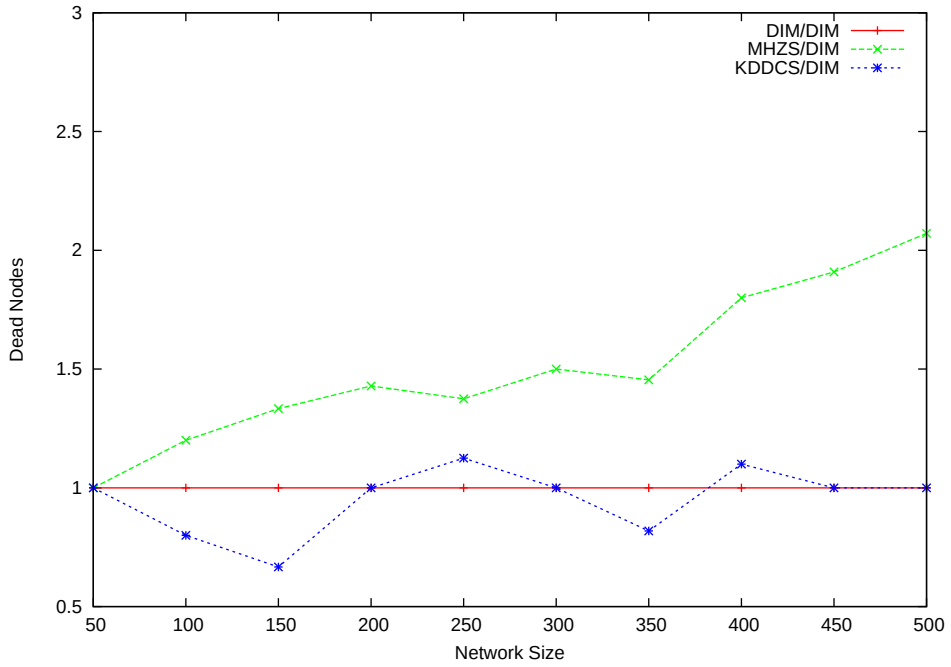


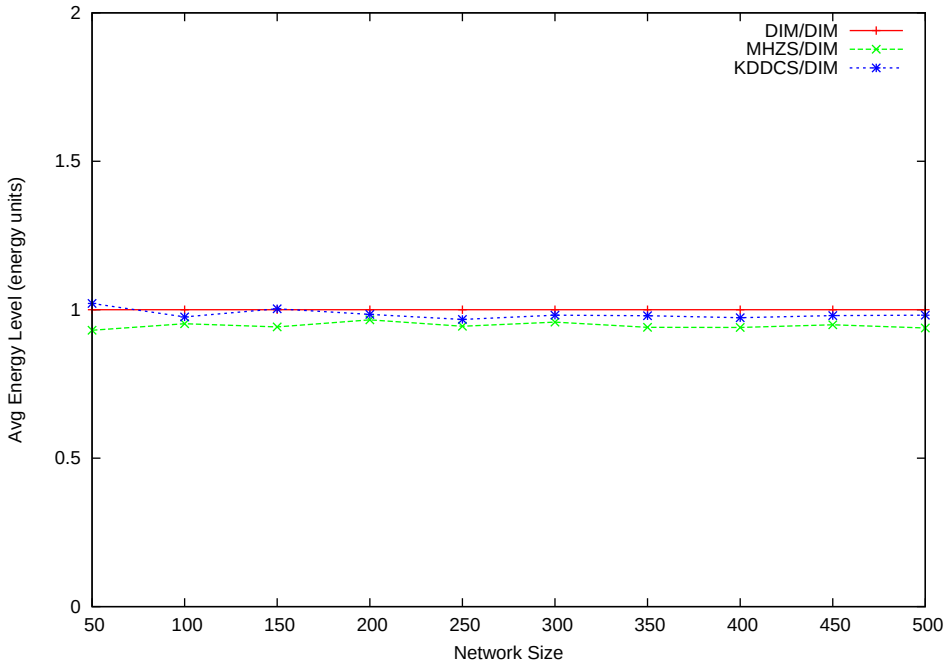
Figure 60: KDDCS: Number of Full Nodes for an 80% Single Storage Hotspot

that the KDDCS scheme does not let nodes drop readings. Instead, once they reach, or be close to, their capacity, rebalancing takes place. Thus, a larger amount of sensor nodes become involved in storing the hotspot data. It is also important to note how KDDCS scales with the network size while maintaining the same percentage of improvement over other schemes for all network sizes. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

R3. Energy Consumption: Figure 61(a) and 61(b) compare the performance of the three schemes in terms of dead nodes and average node storage for an 80% single storage hotspot. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The first figure shows that KDDCS achieves a very comparable performance to that of DIM, while MHZS performs



(a) 80% Single Storage Hotspot



(b) 80% Single Storage Hotspot

Figure 61: KDDCS: Energy Consumption Graphs vs Single Storage Hotspots

worse than the DIM scheme by 50% to 100%. The reason behind that is that KDDCS copes with hotspots in a very early stage of their formation. This helps in reducing the overhead imposed on nodes responsible for storing the hotspot data throughout the network operation. Although MHZS decomposes hotspots, its response to the hotspot formation is not quick enough. This continues to impose a large energy consumption overhead on the nodes falling in the center of the hotspot and increases the death among them by the end of the network operation time. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

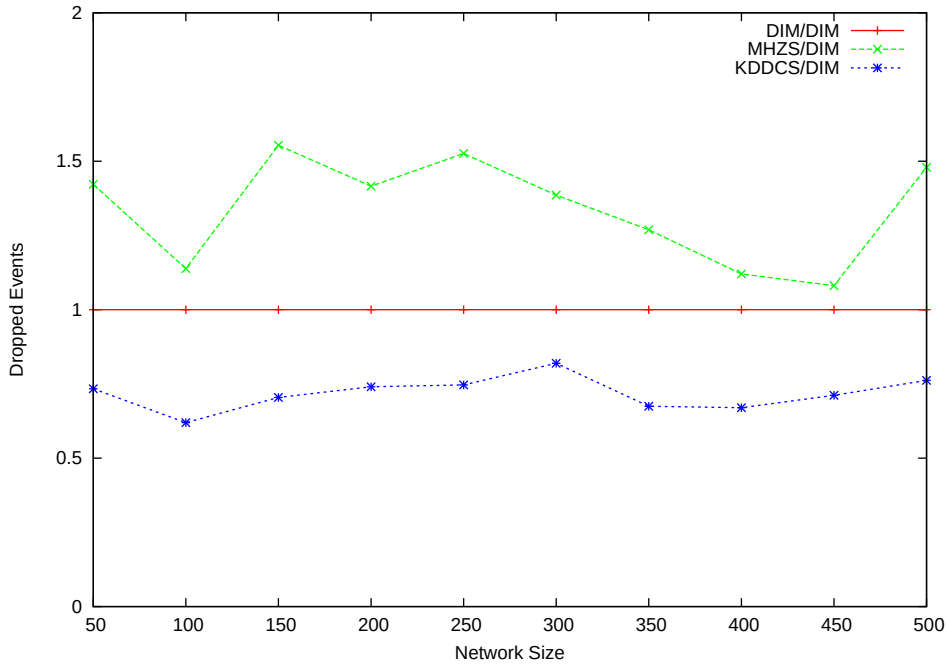
The second figure shows that KDDCS imposes a 1% energy consumption overhead on DIM while MHZS imposes 3%. This can be explained by the fact that KDDCS increases the node participation throughout the network by devoting a larger number of nodes to store the hotspot data. This results in involving more nodes in both storing and routing data. Consequently, this decreases the average energy level of the sensor nodes throughout the network. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

In conclusion, KDDCS achieves a 75% QoD improvement over DIM. This comes at almost no additional cost (an additional 1% energy consumption overhead that KDDCS imposes on the sensor nodes in the network as compared to DIM). Unlike MHZS, KDDCS performance scales for any hotspot size and does not increase the number of dead nodes as MHZS does.

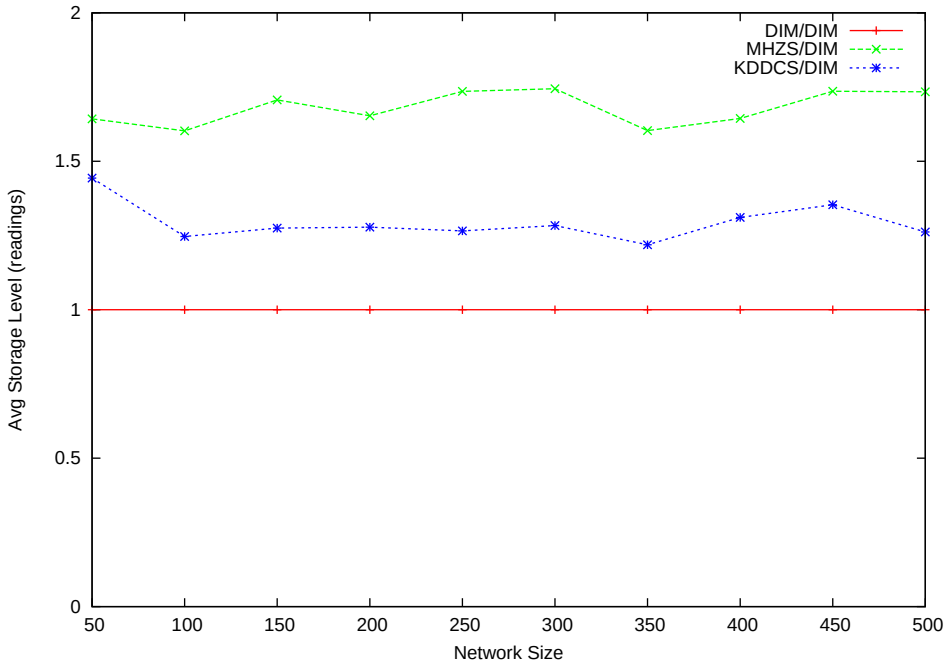
5.6.1.2 Multiple Simultaneous Static Storage Hotspots

We now study the performance of our KDDCS scheme compared to DIM and MHZS for multiple simultaneous static storage hotspots. We conduct our study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

R1. QoD: Figures 62(a) and 62(b) compare the performance of the three schemes in terms of dropped readings and average node storage for 60% multiple storage hotspots. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The first figure shows that KDDCS outperforms DIM, which in turn outperforms MHZS for all network sizes. In fact, KDDCS improves DIM's QoD by around 25% for all network sizes. This shows the high ability of our global KDDCS scheme



(a) Dropped Events for 60% Multiple Storage Hotspots



(b) Average Node Storage for 60% Multiple Storage Hotspots

Figure 62: KDDCS: QoD Graphs vs Multiple Storage Hotspots

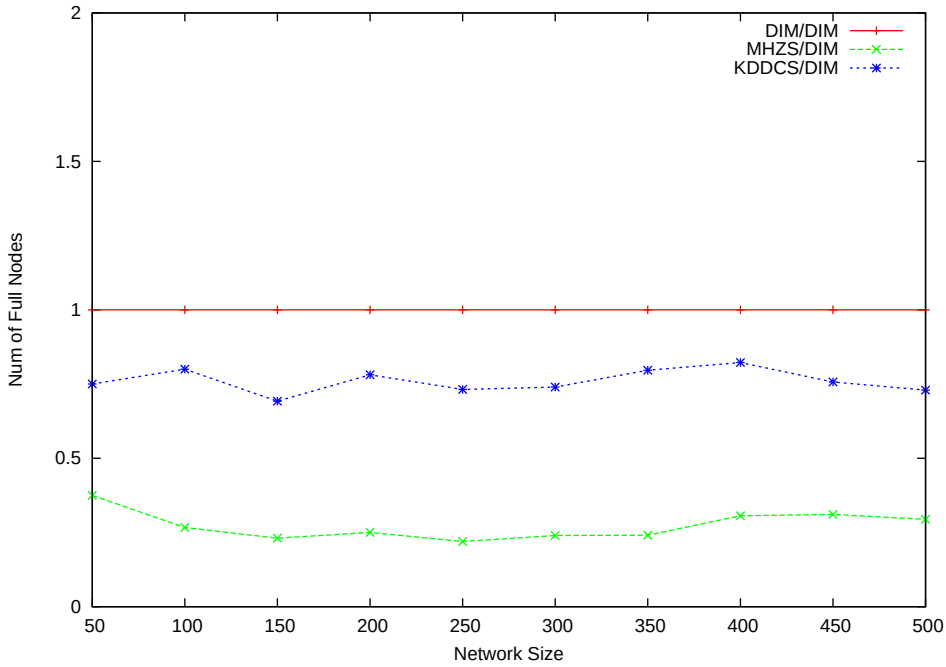


Figure 63: KDDCS: Number of Full Nodes for 60% Multiple Storage Hotspots

to maintain a higher portion of the data falling in the ranges of the simultaneous hotspots. Recall that MHZS suffered from the increased collision symptom for multiple hotspots and note its effect on degrading the performance of MHZS.

The second figure shows that KDDCS improves DIM's average node storage by 33% for all network sizes. This is due to load balancing the storage responsibility of the hotspot data among a larger amount of sensor nodes, which consequently reduces the average storage load imposed on each of the sensor nodes. Both figures show the high KDDCS ability to cope with multiple storage hotspots compared to the other two schemes. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

R2. Load Balancing: Figure 63 compares the number of full nodes of the different schemes for 60% multiple storage hotspots. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The figure shows that KDDCS decreases the number of full nodes of the DIM scheme by around

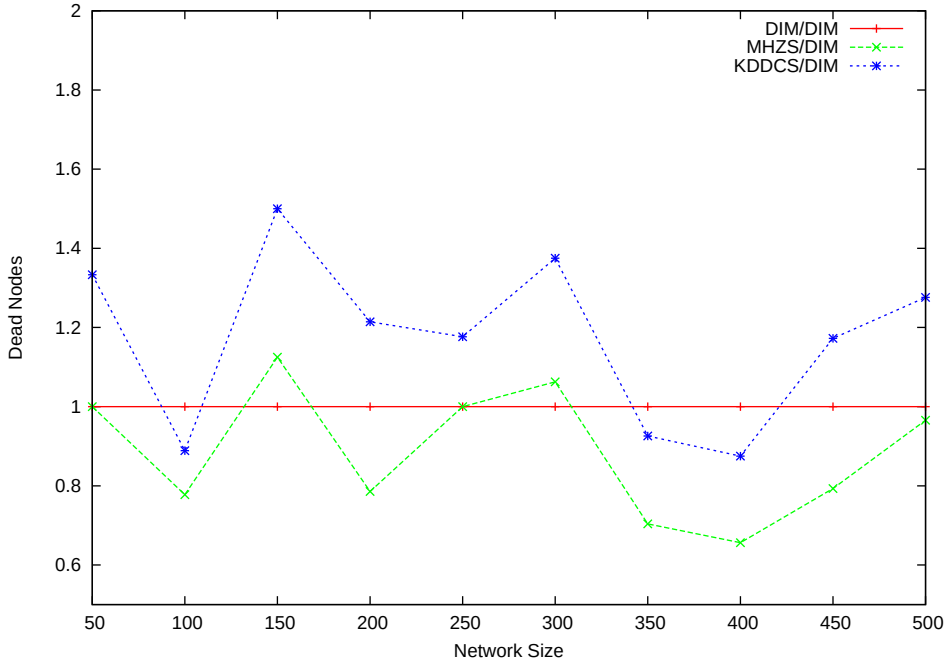
20%. This is because the DIM scheme does not load-balance the storage hotspot data which forces all the nodes falling in the hotspot area to reach their full capacity and start dropping readings. This is unlike KDDCS which reduces the burden imposed on these nodes by load balancing storage responsibility on a larger number of sensor nodes, thus reducing the storage share of each of these nodes and consequently reducing the number of full nodes. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

R3. Energy Consumption: Figures 64(a) and 64(b) compare the performance of the three schemes in terms of dead nodes and average node energy for 80% and 60% multiple storage hotspots, respectively. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The first figure shows that KDDCS acts very comparably to DIM in terms of energy consumption. As for the second figure, it shows that KDDCS performance ranges from 10% better to 20% worse than that of DIM. Knowing that both schemes impose a number of dead nodes of around 5% to 6% of the overall network size, one can realize that both schemes perform quite comparably in terms of node deaths. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

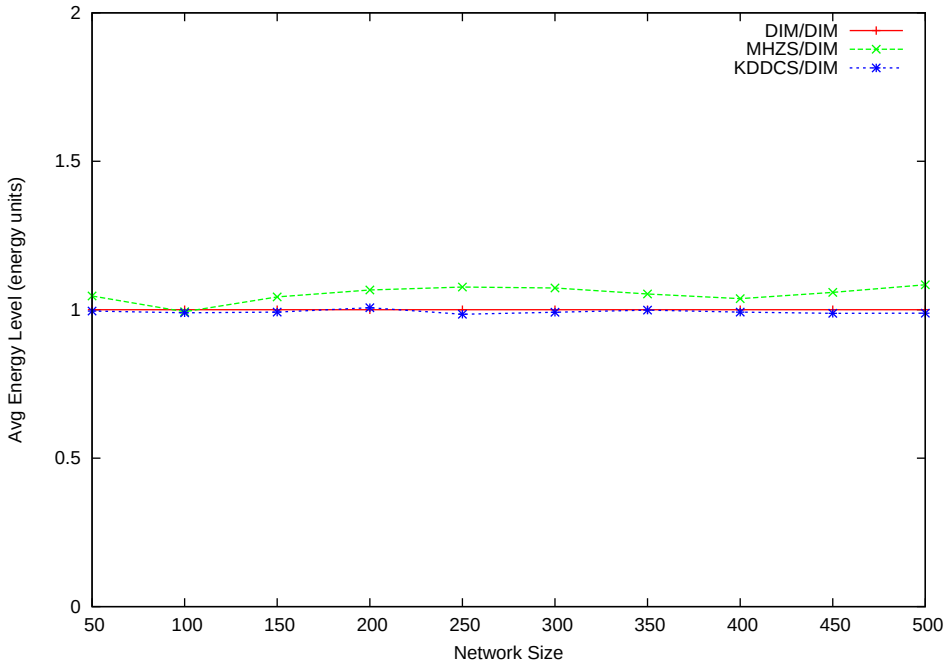
In conclusion, the above three results show that KDDCS has a much higher ability to efficiently deal with multiple simultaneous hotspots compared to the other two schemes. KDDCS achieves a 25% QoD improvements without imposing any additional energy consumption overhead on sensor nodes. This is considered as a very good achievement for KDDCS compared to both DIM and MHZS.

5.6.1.3 Moving Storage Hotspots We now study the KDDCS performance for the third type of hotspots we are interested in, which are moving storage hotspots. We conduct our study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

R1. QoD: Figures 65(a) and 65(b) compare the QoD of the three schemes in terms of dropped events and average node storage when facing a 40% moving storage hotspot. Recall

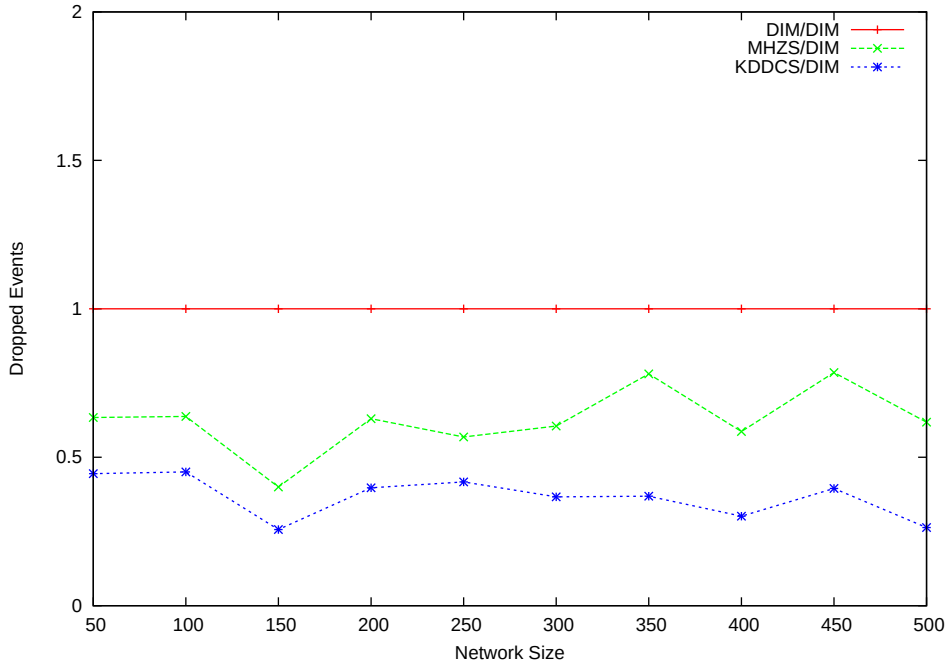


(a) Dead Nodes for 80% Multiple Storage Hotspots

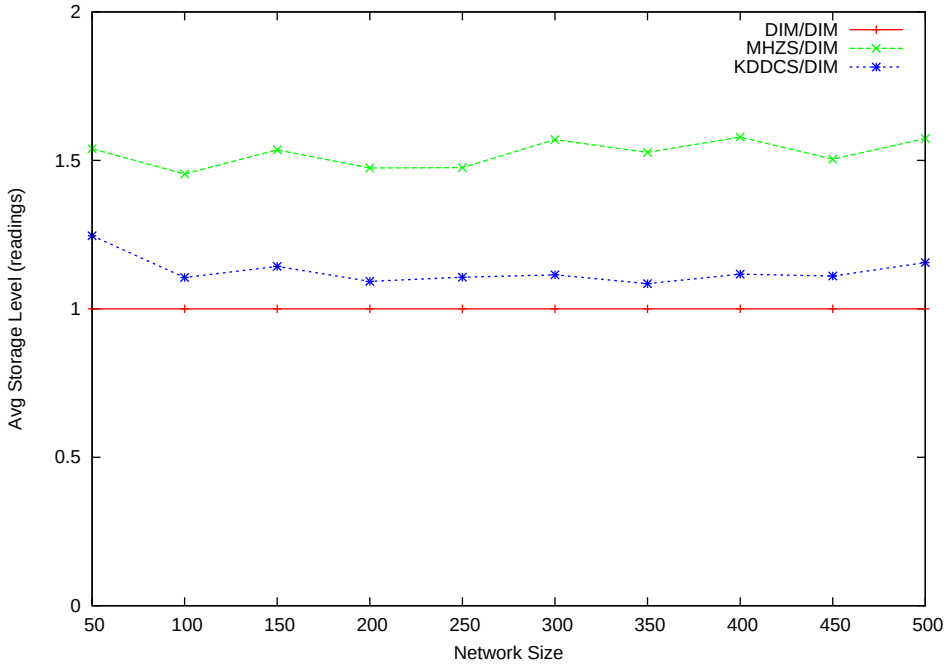


(b) Average Node Energy for 60% Multiple Storage Hotspots

Figure 64: KDDCS: Energy Consumption Graphs vs Multiple Storage Hotspots



(a) Dropped Events



(b) Average Node Storage

Figure 65: KDDCS: QoD Graphs for a 40% Moving Storage Hotspot

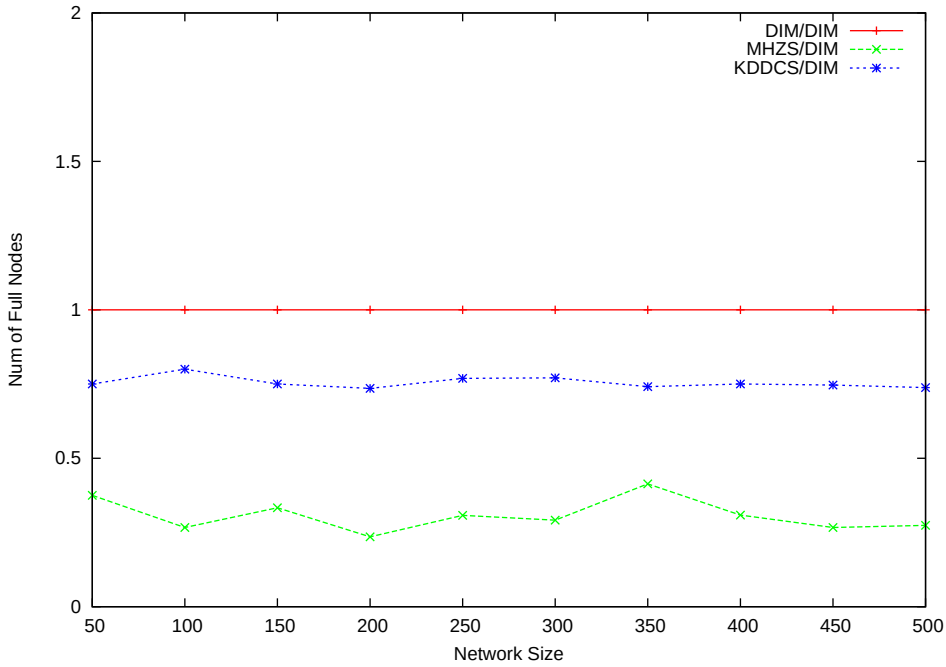


Figure 66: KDDCS: Number of Full Nodes for a 40% Moving Storage Hotspot

that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The first figure shows that KDDCS outperforms the two other schemes in terms of number of dropped readings. KDDCS improves QoD by around 60% over DIM. The important observation in this figure is that KDDCS improvements scale with the network size.

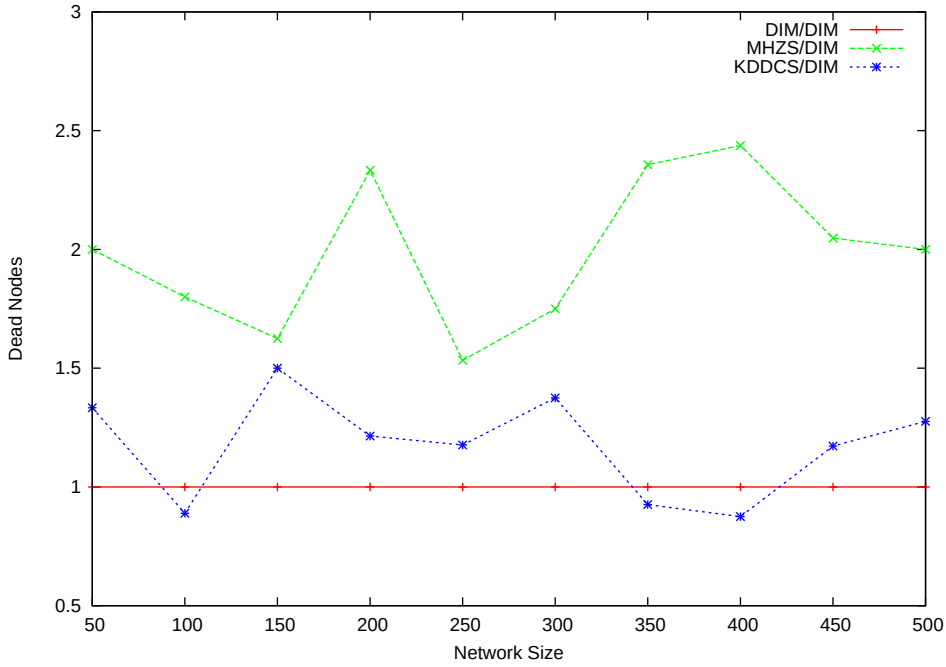
The second one shows that KDDCS continues to react quicker and with a larger-scale than that of MHZS. This increases the number of nodes responsible of the hotspot data and subsequently reduces the average node storage of each of these nodes. The figure reflects this by showing how KDDCS increases the DIM's average node storage by 20%. The figures show the high QoD improvement achieved by KDDCS compared to the other schemes. We achieved similar results for hotspots of sizes up to 80%.

R2. Load Balancing: Figure 66 shows the number of full nodes of the three schemes for a

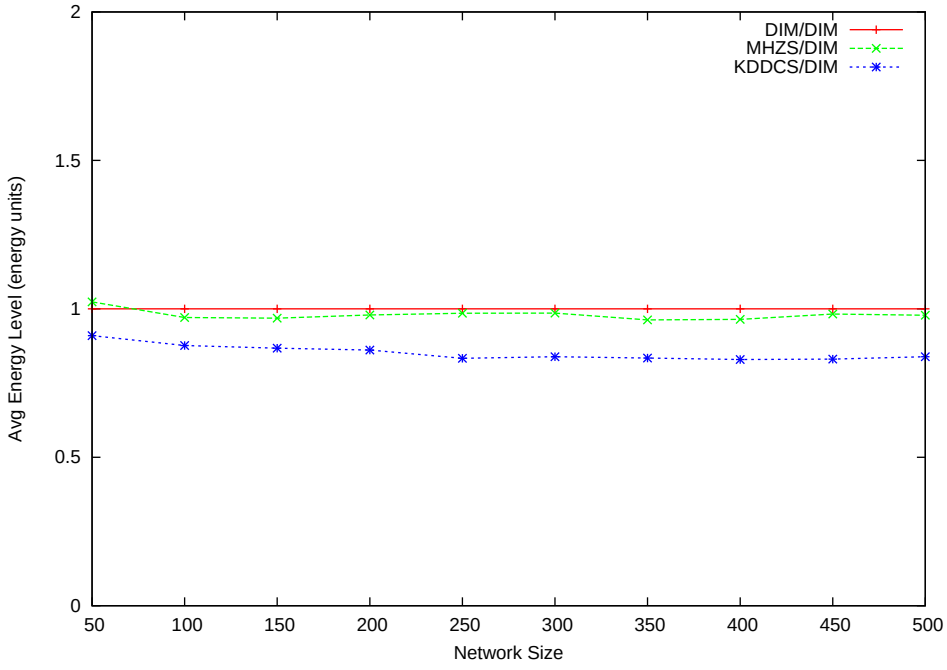
40% moving storage hotspot. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The result is very similar to result R2 from the multiple storage hotspots section. It shows KDDCS ability to load-balance the hotspot data on a larger scale than that of MHZS and in a quicker pace as well. Basically, KDDCS reduces the number of full nodes of DIM by around 20% while it at least doubles that of MHZS. This shows that KDDCS first reduces the number of full nodes falling in the hotspot area, then, keeps expanding the hotspot storage responsibility while being able to maintain the hotspot data in the sensor caches (i.e., without dropping them) to the extent that the new set of nodes responsible for the storage start getting saturated. It should be noted that, if the simulation time is extended, the number of full nodes should decrease as new tree rebalances take place in the network. We achieved similar results for hotspots of sizes up to 80%.

R3. Energy Consumption: Figure 67(a) compares the node deaths of the three schemes for a 40% moving storage hotspot. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The figure shows that KDDCS slightly increases node deaths compared to the basic DIM scheme. The first figure shows that KDDCS continues to perform better than MHZS in terms of node deaths. In fact, KDDCS performance ranges from 10% better than DIM to 40% worse than DIM. Given the fact that the number of node deaths of DIM is around 5% of the network size (for all network sizes), this shows that KDDCS decreases the DIM performance by at most 2% of the network size. On the other hand, MHZS at least doubles node deaths of DIM for all network sizes.

Figure 67(b) compares the schemes in terms of average node energy for a 40% moving storage hotspot. The figure shows that KDDCS slightly reduces the DIM's average node energy by around 13%. This is mainly due to involving a larger number of nodes in the hotspot storage responsibility. Also, this is because of the energy consumption overhead incurred in



(a) Dead Nodes



(b) Average Node Energy

Figure 67: KDDCS: Energy Consumption Graphs for a 40% Moving Storage Hotspot

Table 6: KDDCS Performance (Relative to DIM) for Storage Hotspots

Hotspot Type	QoD Improvements	QoS Overheads
Single Static Hotspots	75%	1%
Multiple Static Hotspots	25%	1%
Moving Hotspots	60%	13%

moving readings among network nodes after each rebalancing operation. As moving hotspots are highly mobile, rebalancing occurs in a higher frequency than that incurred by the static hotspots. However, as the energy consumption increase incurred by KDDCS is not much higher than those of the other schemes, this shows that the energy consumption overhead of the reading migration process incurred by KDDCS when facing dynamic hotspots is fairly small. Note that we achieved similar results for hotspots of sizes up to 80%.

Overall, KDDCS improves QoD by about 60% with a tolerable energy consumption overhead of about 13%, compared to DIM, in the case of moving storage hotspots.

5.6.2 Discussion

In this section, we discussed the ability of the KDTR algorithm to avoid storage hotspots. Based on the usage of KDTR, we experimentally studied the KDDCS performance for single, multiple, and moving storage hotspots. Overall, KDDCS is highly able to cope with storage hotspots of different types and sizes. KDDCS scores good QoD achievements for the different hotspot types and sizes while introducing a slight energy consumption overhead on the sensor nodes across the network (as compared to DIM). For single hotspots, QoD improvements are around 75% over DIM while energy consumption overheads are around 1% over DIM. For multiple static hotspots, QoD improvements are around 25%over DIM while energy consumption overheads are around 1% over DIM. For moving hotspots, QoD improvements are around 60% over DIM while energy consumption overheads are around 13% over DIM. Table 6 summarizes the KDDCS performance for storage hotspots.

5.7 EXTENDING KDTR TO AVOID QUERY HOTSPOTS

As was discussed in the presentation of the ZP/ZPR scheme (Section 4.2), a successful metric to be used as an indication for query hotspots can be the Average Querying Frequency (AQF). Not surprisingly, the same metric can used used by the KDTR algorithm to avoid query hotspots. This is based on the fact that the KDTR mainly applies our distributed algorithm solving the weighted split median problem and this algorithm is a generic algorithm that can be applied using any metric representing the load to be balanced. We describe the AQF-based version of KDTR below. Other than the KDTR modifications, no further changes are introduced to the basic KDDCS components in order to avoid query hotspots.

To be able to cope with query hotspots, each node keeps track of the AQF of its stored zone. Periodically, the KDTR algorithm is applied. The ratios of the AQFs of the two children of each intermediate node of the k-d tree are used to select the maximum tree to be rebalanced. Once the maximum unbalanced tree is determined, the medians of this k-d tree are recursively computed using the weighted split median procedure of the tree rebalancing algorithm with the weight w_i associated with sensor i being the AQF of sensor i. After reading migrations, the resulting k-d tree should re-achieve its AQF-based balance.

It is important to note that the AQF metric for avoiding query hotspots implicitly assumes that the distribution of access frequencies of the subranges (or the readings) within each node should not be uni-tailed, e.g., exponential. Instead, this distribution should be almost symmetric, e.g., a uniform or a normal distribution. This restriction is important for the average of the access frequencies of the readings (or the subranges) to be a fair representative of the expected AQFs of the two sub-zones resulting from evenly splitting the original zone into two parts. Otherwise, AQF cannot be used as a metric for repeatedly load balancing access frequencies in the different levels of the tree.

If this above assumption is not satisfied, the KDTR will need more than one round of rebalancing in order to converge the k-d tree to a load-balanced tree. This is simple because the tree may remain unbalanced after the first round as the hot sub-zone may remain under the responsibility of one or a small number of nodes instead of being split among a large number of sensors. Thus, applying the KDTR again will continue to determine that a subtree

is unbalanced. Repeatedly applying KDTR's tree rebalancing algorithm will have the effect of completely balancing any potential unbalanced subtree.

5.7.1 Experimental Evaluation

In this section, we study the performance of the KDDCS scheme when facing the different types of query hotspots. We compare the performance of the schemes for single static query hotspots (Section 5.7.1.1), multiple static query hotspots (Section 5.7.1.2), and moving query hotspots (Section 5.7.1.3).

Throughout this section, the node storage capacity is equal to 30 readings and the node initial energy capacity is equal to 70 units. Therefore, a *full sensor node* is defined to be a sensor node having 30 readings in its cache. Similarly, a node is depleted (and consequently considered dead) as soon as it consumes 70 energy units. Once a node is dead, all readings stored in this node are considered lost. Based on the DIM scheme, the storage responsibility (a subset of the attribute range) of the dead node is assigned to one of its direct neighbors. Recall that we define an event to be either a reading or a query.

For each hotspot type, we conduct experiments for single, multiple, and moving hotspots. For each of our experiments, we compare the performance of KDDCS for the LS scheme, the GHT scheme, the basic DIM scheme, and the DIM scheme with one ZP/ZPR on top of it. For simplicity, we refer to the DIM/ZP/ZPR scheme by ZP/ZPR.

For each of our experiments, we study three aspects: QoD (R1), load balancing (R2), and energy consumption (R3). For the QoD, we study the number of dropped events (readings/queries) and the average node storage. We refer to the percentage of QoD improvement to be the percentage of decrease in event drops. For the load balancing, we study the number of full nodes. As for energy consumption, we study the average node energy and the number of dead nodes. The average node energy is the one that defines the improvement or the downgrading in the energy consumption performance. To be statistically significant, we conducted 5 simulation runs for each of the experiments and taken the average of values across all runs.

For each of the hotspot types, we conducted experiments on different hotspot sizes rang-

ing from 20% to 100%. Unless otherwise stated, performing well on the large hotspot sizes, i.e., [60%, 80%], implies a good performance on the moderate sized hotspots, i.e., [40%, 60%]. In most of the cases, the performance burden imposed to the network by small hotspots, i.e., hotspots less than 40%, does not justify the cost paid to detect and decompose the hotspots.

To model the worst case performance of KDDCS, we set the scheme to initially start every experiment with a uniform distribution of attribute ranges on sensor nodes. This means that the initial storage responsibility for each of the sensor nodes would be the same for both KDDCS and DIM. The reason behind this selection is to model the efficiency of the KDTR algorithm in dealing with hotspots in cases where the ranges of these hotspots cannot be anticipated in advance, prior to the network operation. Of course, initializing the network with a distribution which partially or fully anticipates the ranges of the hotspots or the distribution of the readings to be stored in the sensor network would boost the KDDCS performance over all other schemes. As this may not be the common case, we decided to model the general case which would help us analyze the worst case KDDCS performance.

Our experimental results are presented in the following sub-sections. Though we tested a range of values for the rebalancing ratio h, we only present results corresponding to $h = 2$ to avoid repetition. In general, changing the value of h did not have a major effect on the KDDCS performance. This is due the type of hotspots that we simulated. As our hotspots usually fall in relatively small subranges of the possible attribute ranges, satisfying the rebalancing criteria of the KDTR algorithm only requires a small h value in most of the cases. This results in almost no difference in performance between h values ranging from 1.5 to 3. Thus, Concerning the ZP/ZPR parameters, we use their default values already presented in Chapter 4.

Before presenting the results of our experiments, we highlight the learned lessons out of our experimental evaluation in the following points:

1. KDDCS scores good QoD achievements for the different hotspot types and sizes while introducing a tolerable energy consumption overhead on the sensor nodes across the network.
2. For single query hotspots, QoD improvements are around 13% while energy consumption overheads are 2.5%.

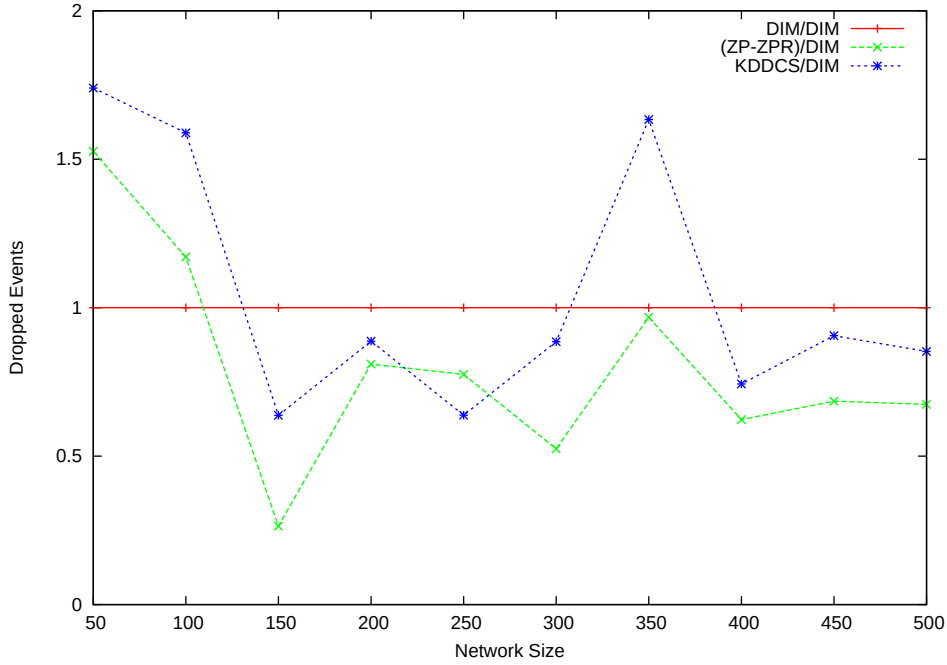
3. For multiple static query hotspots, QoD improvements are around 87% for multiple query hotspots while energy consumption overheads are 15%.
4. For multiple dynamic (moving hotspots), QoD improvements are around 35% for moving query hotspots while energy consumption overheads are around 3%.
5. The performance improvements of KDDCS scale proportionally with the network size.

The results of the simulations are shown in Figures 68 to 76. Recall that DIM already outperforms both the LS and the GHT schemes. That's why we only plot the KDDCS performance together with those of the DIM and DIM/ZP/ZPR schemes. In these figures, we compare the performance of KDDCS with those the basic DIM and the ZP/ZPR schemes, with respect to our different performance measures.

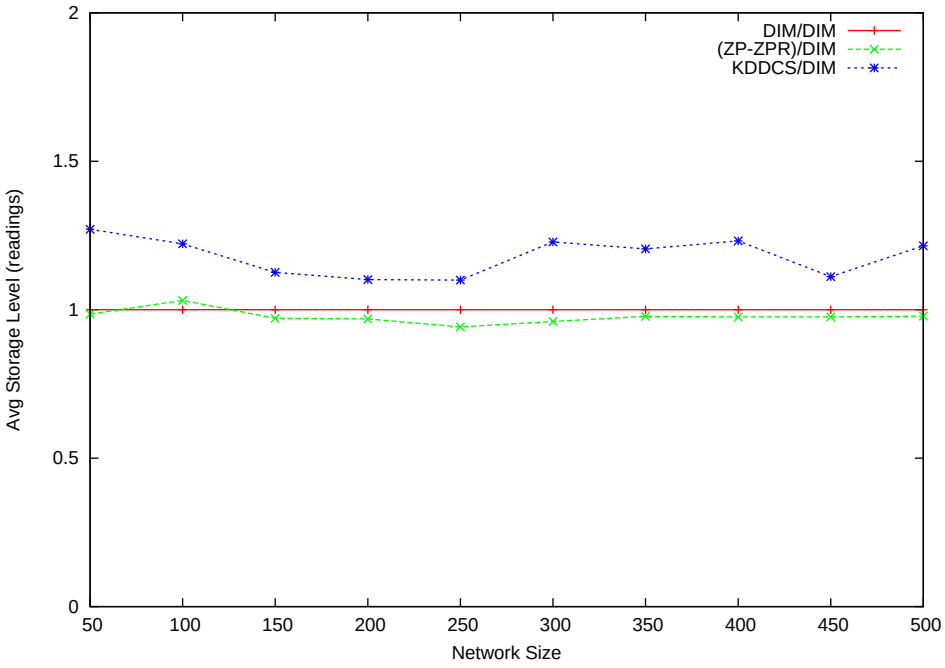
5.7.1.1 Single Static Query Hotspots In this subsection, we compare the performance of KDDCS to those of DIM and ZP/ZPR for single query hotspots. We conduct our study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

R1. QoD: Figures 68(a) and 68(b) compare the performance of the three schemes in terms of dropped readings and average node storage for an 80% single query hotspot. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The first figure shows that KDDCS slightly reduces the number of dropped events by around 13% for most of the network sizes compared to DIM. It is important to observe that the reduction achieved by ZP/ZPR is quite better for most of the cases. The second figure shows that KDDCS increases the average node storage by around 18% compared to the other two schemes (which perform similarly). This result shows that KDDCS achieves a slightly better data persistence/query answering capability in most of the cases by reducing collisions at the hotspot nodes. However, the most important indication is that KDDCS assigns the responsibility of the hotspot to a larger number of nodes, which in turn, increases the average node storage for all network nodes. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

R2. Load Balancing: Figure 69 compares the three schemes in terms of number of full



(a) Dropped Events for an 80% Single Query Hotspot



(b) Average Node Storage for an 80% Single Query Hotspot

Figure 68: KDDCS: QoD Graphs vs Single Query Hotspots

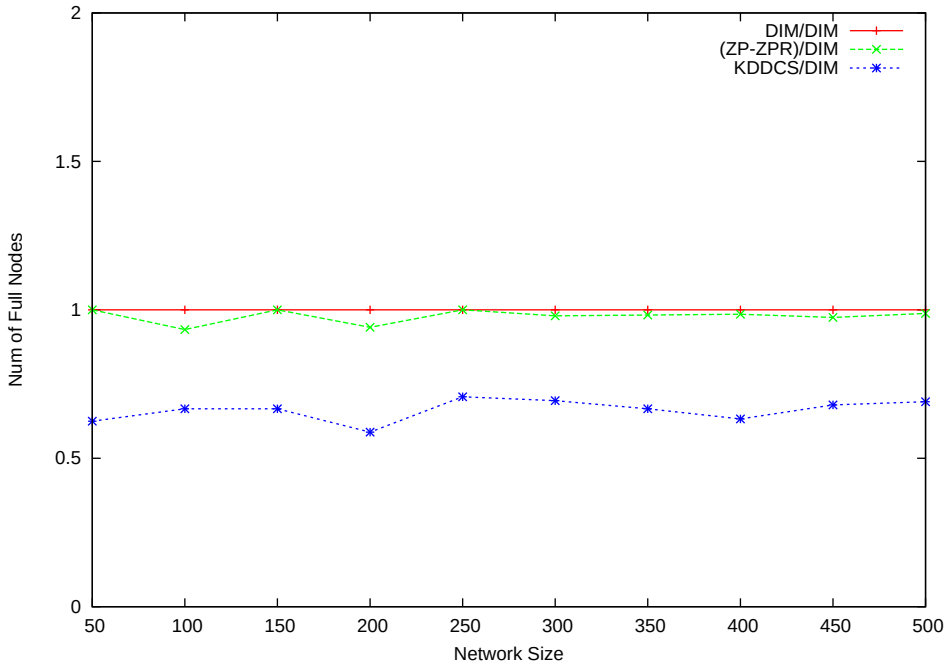


Figure 69: KDDCS: Number of Full Nodes for a 60% Single Query Hotspot

nodes for a 60% single query hotspot. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The figure shows that KDDCS has a lower number of full nodes than those of DIM and ZP/ZPR. Recall that the reading distribution imposed on the sensor network in this experiment is drawn, uniformly at random, from a uniform distribution. The figure shows that both DIM and ZP/ZPR have the almost identical values for the number of full nodes for all network sizes. KDDCS reduces the number of full nodes by around 40%. Recall that the reading distribution imposed on the network was drawn uniformly at random from a uniform distributed over the attribute ranges. Additionally, we use the version of KDDCS that starts with a uniform assignment of ranges to nodes. Thus, the important observation in the figure is that DIM and ZP/ZPR have comparable numbers of full nodes. As reading distribution is not skewed, then, this number represents the number achieved by DIM in the regular cases. As KDDCS decreases the number of full nodes, this means that it achieves a better load balancing of

readings across the sensor network. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

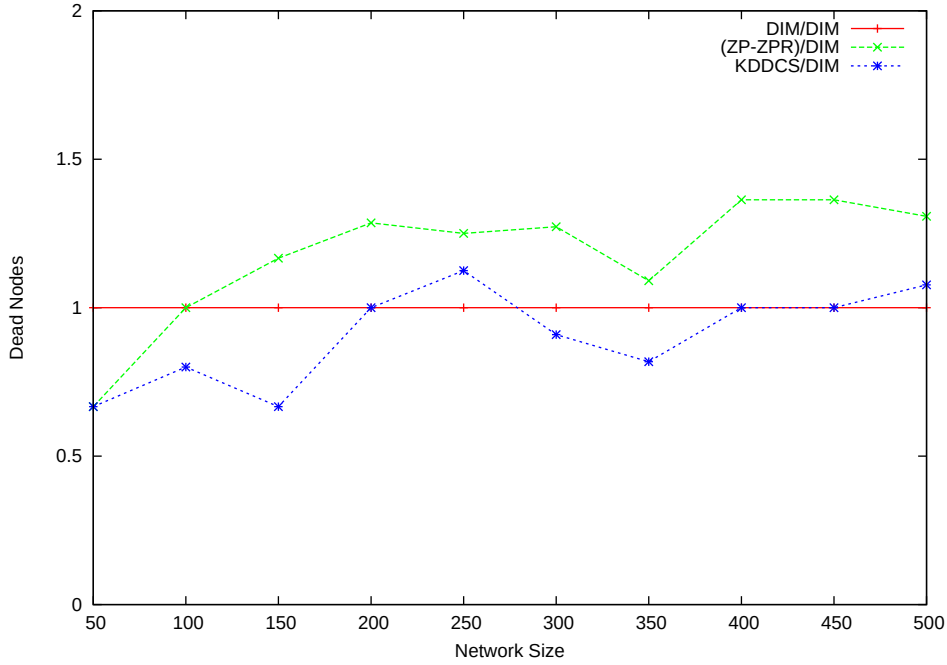
R3. Energy Consumption: Figures 70(a) and 70(b) compare the performance of the three schemes in terms of dead nodes and average node energy for 80% and 60% query hotspots, respectively. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The first figure shows that KDDCS performs quite similarly to the DIM scheme in terms of dead nodes. Although KDDCS increases the number of dead nodes by 30% for small networks, this not viewed as a huge effect as the number of dead nodes scored by DIM in these cases is less than 3% of the network size. The second figure shows that KDDCS slightly decreases the average energy level by around 2.5% compared to the DIM scheme. Both figures show that KDDCS does not impose any remarkable energy consumption overhead on the DIM for single query hotspots unlike the ZP/ZPR scheme. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

In conclusion, KDDCS achieves acceptable QoD improvements over DIM (around 13%) for single query hotspots. Towards this improvement, KDDCS introduces an energy consumption overhead of around 2.5%. The performance gains are expected to be much higher in the case of long-lasting query hotspots.

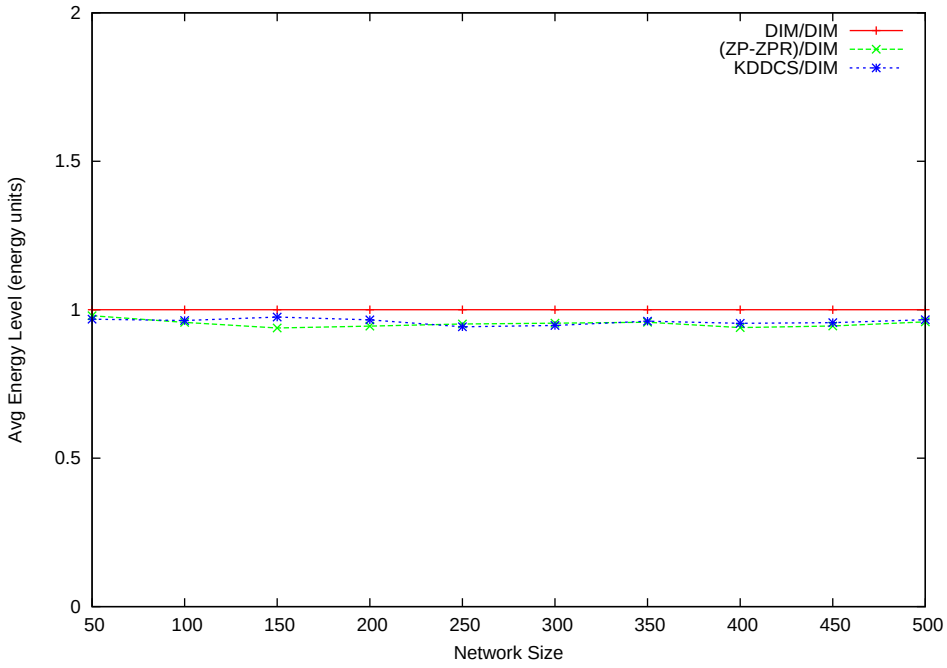
5.7.1.2 Multiple Simultaneous Static Query Hotspots

In this subsection, we compare the performance of KDDCS to those of DIM and ZP/ZPR for mutiple simultaneous query hotspots. We conduct our study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

R1. QoD: Figures 71(a) and 71(b) compare the performance of the three schemes in terms of dropped readings and average node storage for 80% and 60% query hotspots, respectively. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This

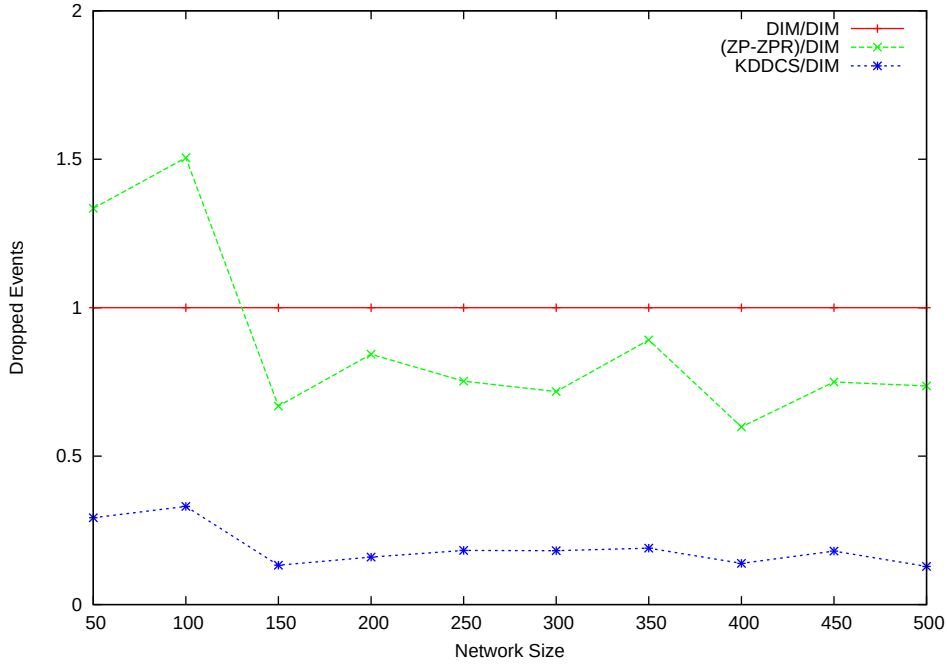


(a) Dead Nodes for an 80% Single Query Hotspot

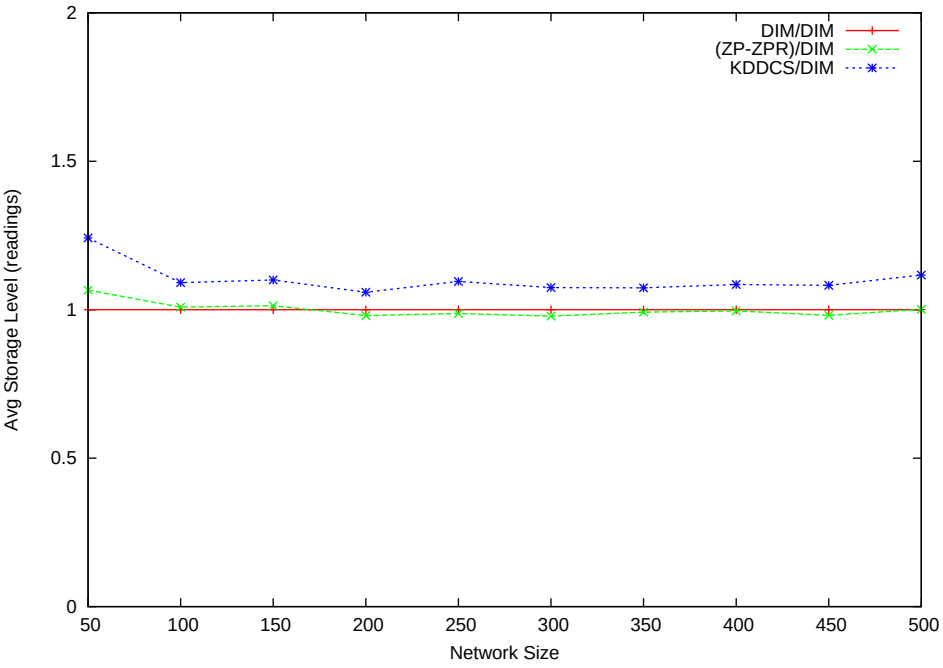


(b) Average Node Energy for a 60% Single Query Hotspot

Figure 70: KDDCS: Energy Consumption Graphs vs Single Query Hotspots



(a) Dropped Events for 80% Multiple Query Hotspots



(b) Average Node Storage for 60% Multiple Query Hotspots

Figure 71: KDDCS: QoD Graphs vs Multiple Query Hotspots

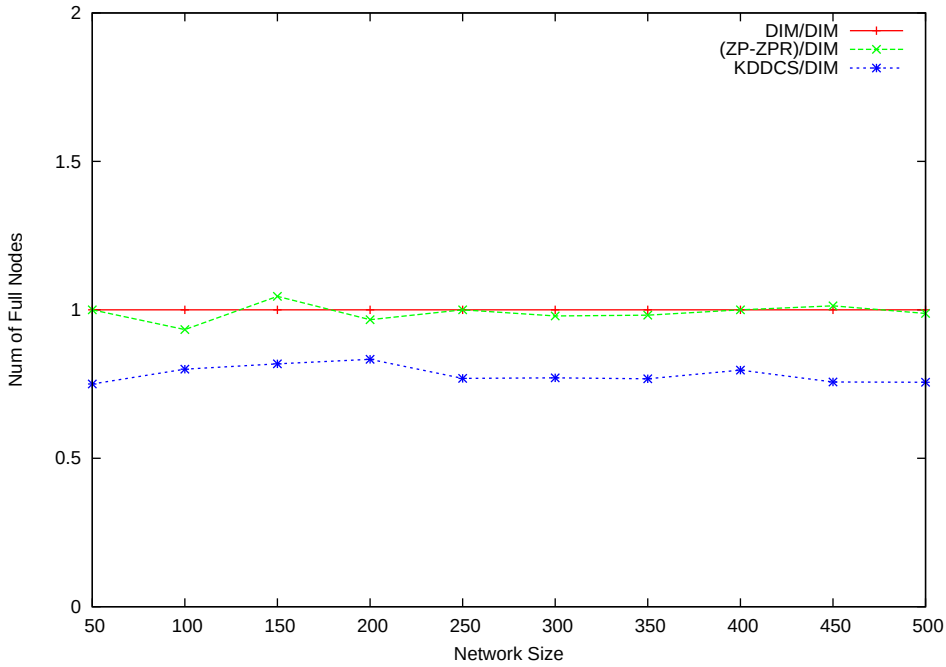


Figure 72: KDDCS: Number of Full Nodes for 80% Multiple Query Hotspots

helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The two figures show that KDDCS achieves a large performance improvement compared to the two other schemes. The first figure shows that KDDCS improves QoD by around 87%. The important note is that the number of dropped events is quite constant for KDDCS. The second figure shows that KDDCS improves the average node storage by around 15%. Both figures show that the benefit of the global load balancing strategy followed by KDDCS shines for a sophisticated hotspot setting such as that of multiple simultaneous query hotspots. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

R2. Load Balancing: Figure 72 compares the numbers of full nodes of the three schemes for 80% multiple query hotspots. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The figure shows that KDDCS reduces the number of full nodes by around 20%. The important

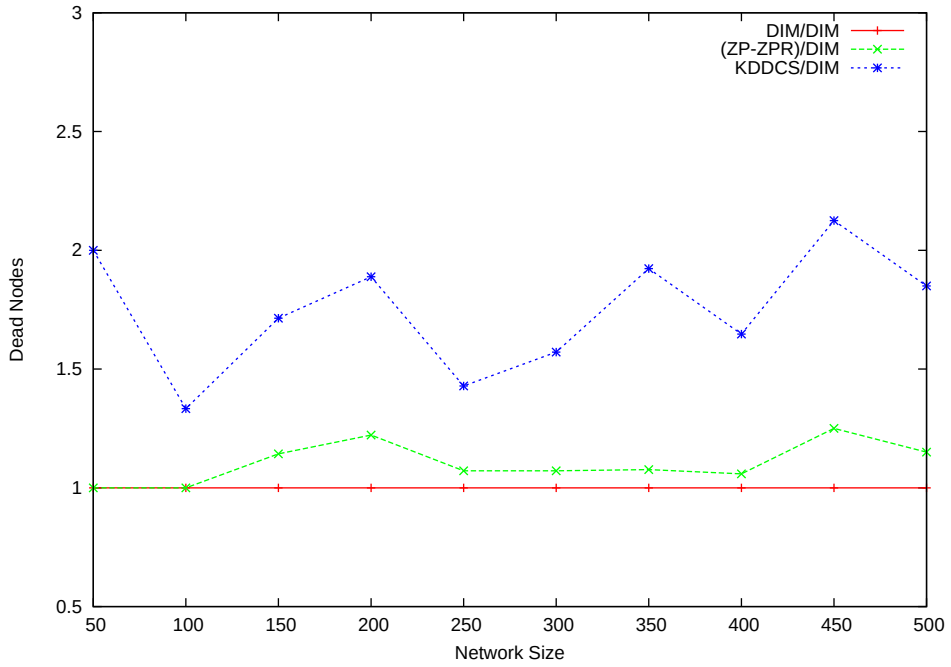
lesson that the figure shows is that reassigning the hotspot data to a larger group of sensors helps achieving a better load balancing in terms of storage load responsibility imposed on the different sensor nodes. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

R3. Energy Consumption: Figure 73(a) compares the number of dead nodes of the three schemes for 60% multiple query hotspots. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The first figure shows that KDDCS moderately the number of dead nodes by around 75% as compared to the other two schemes which perform quite similarly. This shows that the continuously occurring rebalancing process of the KDDCS scheme slightly increases the energy consumption load imposed on some of the network nodes. However, it should be noted that the DIM performance in terms of the number of dead nodes does not exceed 6% of the network size (for all network sizes). This shows that the performance degradation of the KDDCS scheme does not exceed 4.5% of the network size. Figure 73(b) shows the average energy level of the three schemes for 60% multiple query hotspots. The figure shows the KDDCS decreases the average energy level by 15% compared to DIM. This is in part due to the reading migration overhead imposed by KDDCS on the different network nodes. Also, this is due to the ability of KDDCS to load-balance energy consumption among a larger number of network nodes, thus, decrease the number of intact nodes in the network. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

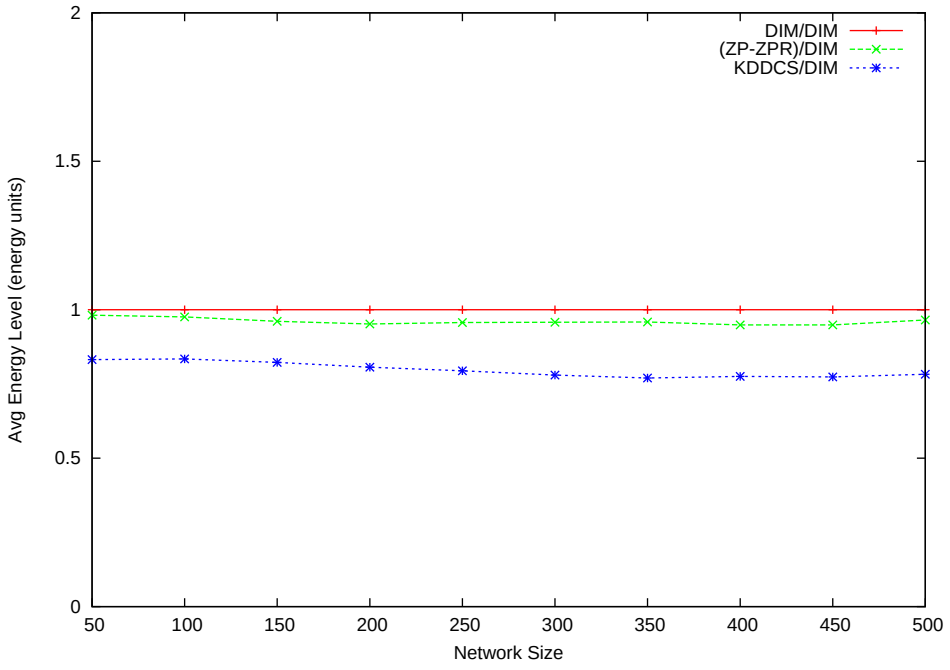
In conclusion, KDDCS achieves a high QoD improvement on top of DIM (around 87%) for multiple simultaneous query hotspots. However, this comes with a moderate energy consumption overhead (around 15% over DIM) that KDDCS introduces to the sensor network.

5.7.1.3 Moving Query Hotspots

In this subsection, we compare the performance of KDDCS to those of DIM and ZP/ZPR for a moving query hotspot. We conduct our study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

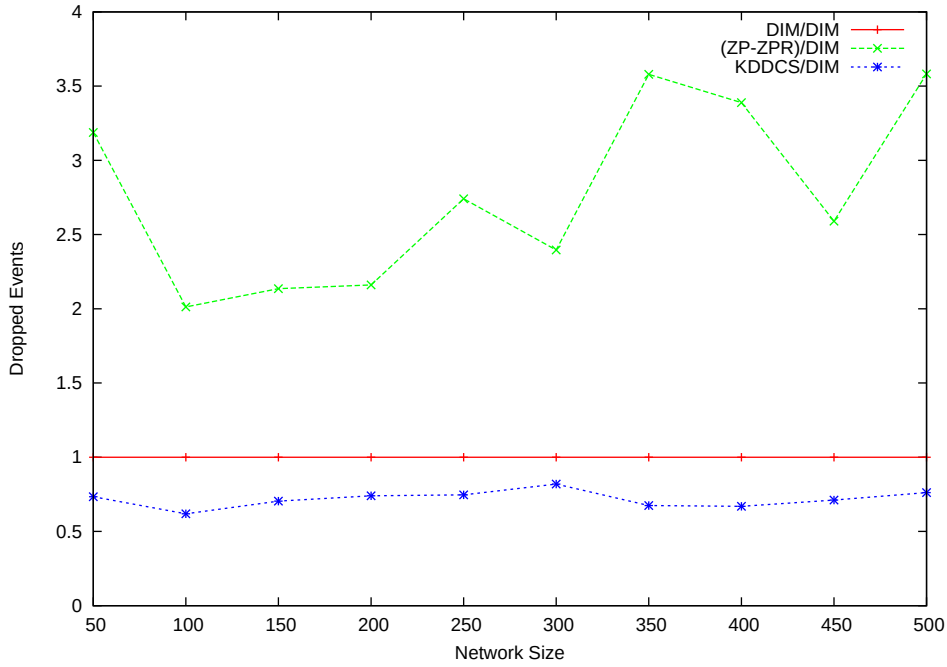


(a) Dead Nodes for 60% Multiple Query Hotspots

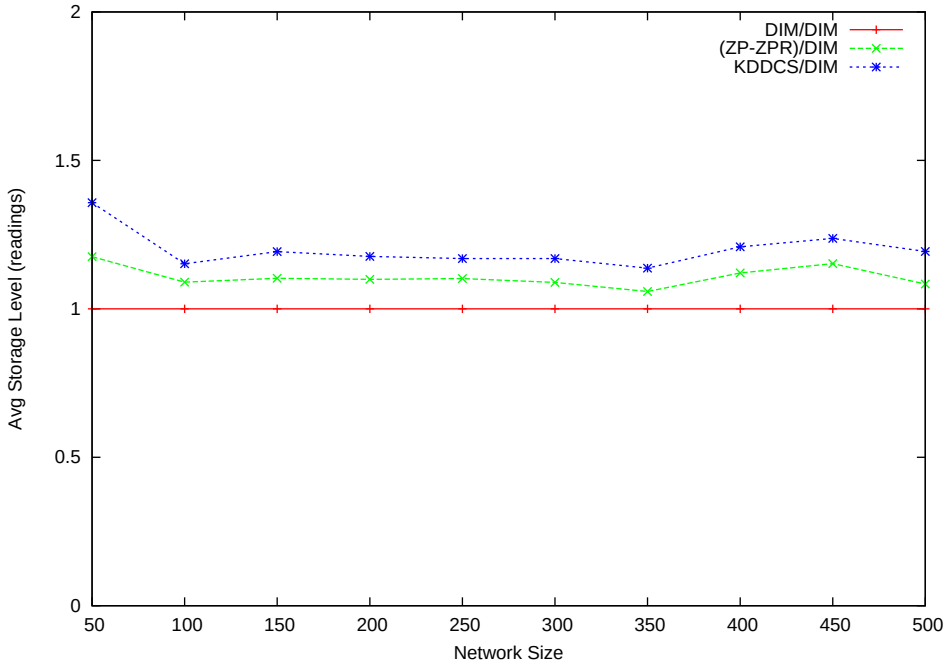


(b) Average Node Energy for 60% Multiple Query Hotspots

Figure 73: KDDCS: Energy Consumption Graphs vs Multiple Query Hotspots



(a) Dropped Events



(b) Average Node Storage

Figure 74: KDDCS: QoD Graphs for a 60% Moving Query Hotspot

R1. QoD: Figures 74(a) and 74(b) compare the performance of the three schemes in terms of dropped events and average node storage for a 60% moving storage hotspot. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The first figure shows that KDDCS improves DIM's QoD by 35%. This shows the high data persistence gain achieved by KDDCS over the other schemes. It is important to note that amount of improvement is comparable to the multiple static query hotspots case, while it is much larger than for the single query hotspots cases. This shows that the benefits of global load balancing are more obvious for more sophisticated hotspots. It is important to note that ZP/ZPR is performs much worse than DIM (around 300% worse). The second figure shows that KDDCS increases the average node storage by 40% over DIM. This demonstrates the KDDCS ability to improve data persistence by assigning the storage responsibility of the hotspot data to a larger amount of sensor nodes across the sensor network. This consequently improves the average storage of sensor nodes. We achieved similar results for hotspots of sizes up to 80%.

R2. Load Balancing: Figure 75 compares the three schemes in terms of number of full nodes for a 60% moving query hotspot. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The figure shows a similar load balancing result to that achieved by KDDCS for multiple simultaneous query hotspots. Basically, KDDCS reduces the DIM's number of full nodes by around 25%. This shows that, when KDDCS decomposes the query hotspots at the beginning of their formation, it achieves a better load balancing of storage among the different sensor nodes in the network. We achieved similar results for hotspots of sizes up to 80%.

R3. Energy Consumption: Figures 76(a) and 76(b) compare the performance of the three schemes in terms of dead nodes and average node energy when facing a 60% moving query hotspot. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed

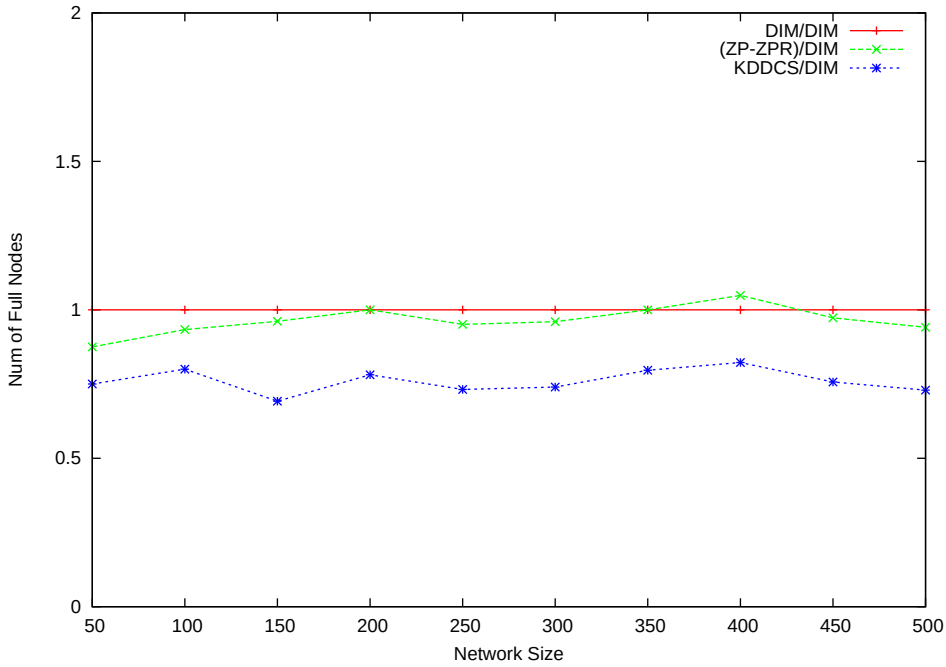
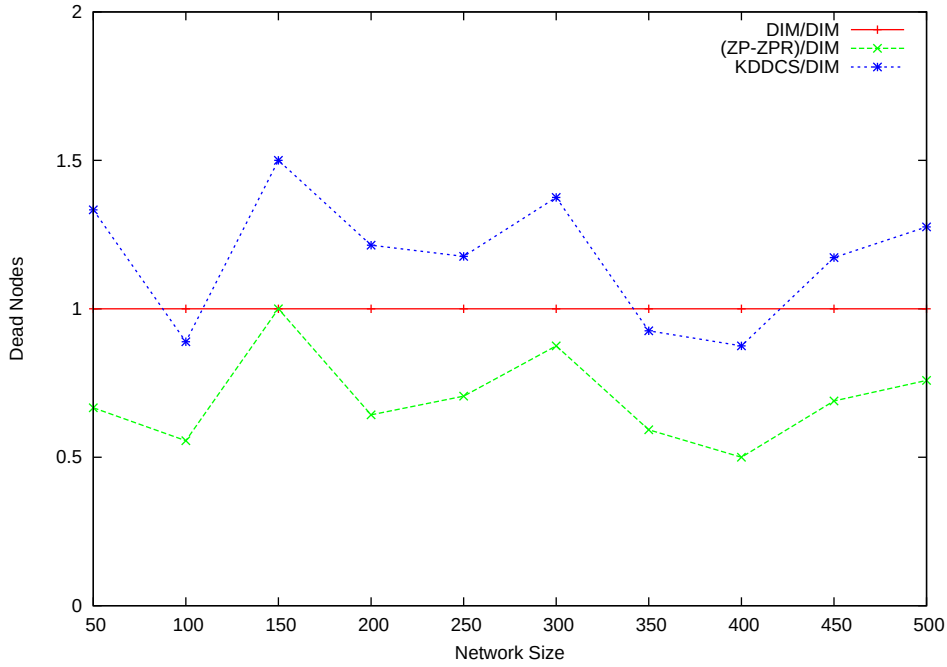


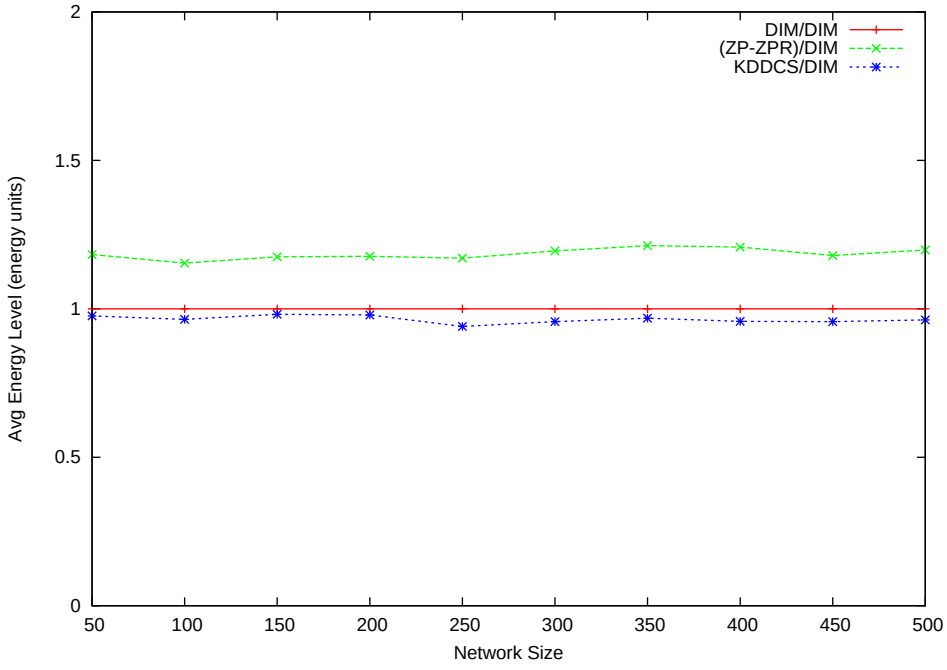
Figure 75: KDDCS: Number of Full Nodes for a 60% Moving Query Hotspot

by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The first figure shows that KDDCS performs slightly worse than DIM in terms of the number of dead nodes. Basically, KDDCS increases node deaths by at around 20% compared to DIM (for most network sizes). The number of dead nodes of DIM does not exceed 8% of the network size (for all network sizes). The second figure shows that KDDCS imposes at most 3% performance overhead over DIM in terms of average node energy. Both figures show that KDDCS does not incur except a slight energy consumption overhead compared to DIM for moving query hotspots.

Overall, KDDCS improves QoD performance compared to DIM and ZP/ZPR by without introducing any remarkable energy consumption overhead. QoD improvements are around 35% over DIM while energy consumption overhead is at most 3%. We achieved similar results for hotspots of sizes up to 80%.



(a) Dead Nodes



(b) Average Node Energy

Figure 76: KDDCS: Energy Consumption Graphs for a 60% Moving Query Hotspot

Table 7: KDDCS Performance (Relative to DIM) for Query Hotspots

Hotspot Type	QoD Improvements	QoS Overheads
Single Static Hotspots	13%	2.5%
Multiple Static Hotspots	87%	15%
Moving Hotspots	35%	3%

5.7.1.4 Discussion In this section, we studied the KDDCS performance for single, multiple, and moving query hotspots. Overall, KDDCS is highly able to cope with query hotspots of different types and sizes. KDDCS scores good QoD achievements for the different hotspot types and sizes while introducing a tolerable energy consumption overhead on the sensor nodes across the network. For single hotspots, QoD improvements are around 13% over DIM while energy consumption overheads are 2.5% over DIM. For multiple static hotspots, QoD improvements are around 87%over DIM while energy consumption overheads are around 15% DIM. For moving hotspots, QoD improvements are around 35% over DIM while energy consumption overheads are 3% over DIM. Unlike ZP/ZPR, the benefits of KDDCS becomes quite obvious for multiple and moving query hotspots. Table 7 summarizes the KDDCS performance for query hotspots.

We now move on to present the KDTR handling of mixed hotspots.

5.8 EXTENDING KDTR TO AVOID MIXED HOTSPOTS

We now move on to discuss the mechanism of avoiding mixed hotspots using the KDTR algorithm. Recall that mixed hotspots are those composed of simultaneous storage and query hotspots. Similar to the case with query hotspots, the modifications that we introduce in this subsection are mainly to the KDTR algorithm, i.e., the basic components of KDDCS remain unchanged. The basic idea to avoid mixed hotspots is to separately detect storage and query hotspots. Depending on the type of imbalance that may exist, the second step

consists of applying the appropriate tree rebalancing algorithm to avoid the formation of all types of hotspots. We explain our new KDTR version below.

The first step in avoiding mixed hotspots is to detect their existence. To perform this task, our KDTR algorithm proceeds with its usual maximum unbalanced subtree selection process. However, in this case, the selection algorithm is performed twice, once using the storage load as a metric and the other using the AQF as a metric.

The result of this unbalancing detection process can be one of the following cases:

1. No unbalanced subtrees exist. In this case, no action is taken.
2. One imbalance type is found, either storage imbalance or querying frequency imbalance. In this case, the corresponding metric to the imbalance type is used in rebalancing the determined subtree.
3. Two unbalancing types are encountered. In this case, the subtrees may be either contained in each other or non-intersecting. We discuss the action for each case below.

In case the unbalanced subtrees are two separate subtrees of the k-d tree, two rebalancing processes are triggered, one for each unbalanced tree. Each process uses the appropriate rebalancing metric. For the case where one of the unbalanced subtrees contains the other, the maximum unbalanced tree is the one that is rebalanced. As for the rebalancing metric, different options exist. The first is to use either the sum or a combination of the storage load and the AQF as the weight w_i of each node i. This has the effect of taking both metrics into account, either evenly, or while giving priority to one of them over the other (by increasing its corresponding multiplier). For example, the multiplier of the metric corresponding to the bigger unbalanced subtree may be twice that of the metric corresponding to the smaller unbalanced subtree. Another possibility is that we can solely use the metric corresponding to the bigger unbalanced subtree relying on the fact that rebalancing it may implicitly rebalance the smaller one. In such a case, the KDTR algorithm may be applied multiple times in order to converge to a fully load-balanced k-d tree. A third possibility is that the degree of unbalancing of the two subtrees may be compared and the more severely unbalanced subtree may be picked to be rebalanced regardless of whether it is the bigger or the smaller subtree.

In conclusion, KDTR can be extended to avoid all types of hotspots including storage,

query, and mixed hotspots. The actual KDTR version applied in for any sensor network can be left as a design parameter based on the expected nature and workloads of the intended sensor network application. Another possibility may be to apply two KDTR versions with different time intervals, e.g., applying the storage hotspots KDTR version every 1 hour while applying the query hotspots one every 10 min in case query hotspots are expected to be the main hotspot source faced by the network. This would have the effect of continuously monitoring the query hotspot formation in the network while not fully neglecting the possibility of formation of storage hotspots without paying the full cost of applying the KDTR algorithm for mixed hotspot avoidance.

5.8.1 Experimental Evaluation

In this section, we study the performance of the final version of the KDTR algorithm versus both uniform loads and mixed hotspots. We compare the KDDCS performance to those of DIM and ZS/ZP/ZPR for uniform loads (Section 5.8.1.1), single mixed hotspots (Section 5.8.1.2), multiple mixed hotspots (Section 5.8.1.3), and moving mixed hotspots (Section 5.8.1.4).

We concentrate on studying correlated mixed hotspots where storage and query hotspots coincide, i.e., fall in the same attribute ranges. Recall that uncorrelated mixed hotspots can be easily considered as *isolated* hotspots of different types, i.e., storage and query hotspots occurring simultaneously in the network. Individually and concurrently dealing with such hotspots should be done by KDDCS in a straightforward manner.

Throughout this section, the node storage capacity is equal to 30 readings and the node initial energy capacity is equal to 70 units. Therefore, a *full sensor node* is defined to be a sensor node having 30 readings in its cache. Similarly, a node is depleted (and consequently considered dead) as soon as it consumes 70 energy units. Once a node is dead, all readings stored in this node are considered lost. Based on the DIM scheme, the storage responsibility (a subset of the attribute range) of the dead node is assigned to one of its direct neighbors. Recall that we define an event to be either a reading or a query.

For each hotspot type, we conduct experiments for single, multiple, and moving hotspots.

For each of our experiments, we compare the performance of KDDCS for the LS scheme, the GHT scheme, the basic DIM scheme, and the DIM scheme with one ZS/ZP/ZPR on top of it. For simplicity, we refer to the DIM/ZS/ZP/ZPR scheme by ZS/ZP/ZPR.

For each of our experiments, we study three aspects: QoD (R1), load balancing (R2), and energy consumption (R3). For the QoD, we study the number of dropped events (readings/queries) and the average node storage. We refer to the percentage of QoD improvement to be the percentage of decrease in event drops. For the load balancing, we study the number of full nodes. As for energy consumption, we study the average node energy and the number of dead nodes. The average node energy is the one that defines the improvement or the downgrading in the energy consumption performance. To be statistically significant, we conducted 5 simulation runs for each of the experiments and taken the average of values across all runs.

For each of the hotspot types, we conducted experiments on different hotspot sizes ranging from 20% to 100%. Unless otherwise stated, performing well on the large hotspot sizes, i.e., $[60\%, 80\%]$, implies a good performance on the moderate sized hotspots, i.e., $[40\%, 60\%]$. In most of the cases, the performance burden imposed to the network by small hotspots, i.e., hotspots less than 40%, does not justify the cost paid to detect and decompose the hotspots.

To model the worst case performance of KDDCS, we set the scheme to initially start every experiment with a uniform distribution of attribute ranges on sensor nodes. This means that the initial storage responsibility for each of the sensor nodes would be the same for both KDDCS and DIM. The reason behind this selection is to model the efficiency of the KDTR algorithm in dealing with hotspots in cases where the ranges of these hotspots cannot be anticipated in advance, prior to the network operation. Of course, initializing the network with a distribution which partially or fully anticipates the ranges of the hotspots or the distribution of the readings to be stored in the sensor network would boost the KDDCS performance over all other schemes. As this may not be the common case, we decided to model the general case which would help us analyze the worst case KDDCS performance.

Our experimental results are presented in the following sub-sections. Though we tested a range of values for the rebalancing ratio h, we only present results corresponding to $h = 2$ to avoid repetition. In general, changing the value of h did not have a major effect on

the KDDCS performance. This is due the type of hotspots that we simulated. As our hotspots usually fall in relatively small subranges of the possible attribute ranges, satisfying the rebalancing criteria of the KDTR algorithm only requires a small h value in most of the cases. This results in almost no difference in performance between h values ranging from 1.5 to 3. Thus, Concerning the ZP/ZPR parameters, we use their default values already presented in Chapter 4.

Before presenting the results of our experiments, we highlight the learned lessons out of our experimental evaluation in the following points:

1. KDDCS scores good QoD achievements for the different hotspot types and sizes while introducing a tolerable energy consumption overhead on the sensor nodes across the network.
2. For single static mixed hotspots, QoD improvements are around 75% for single mixed hotspots while energy consumption overheads are 18%.
3. For multiple static mixed hotspots, QoD improvements are around 95% for multiple mixed hotspots while energy consumption overheads are 29%.
4. For multiple dynamic (moving) mixed hotspots, QoD improvements are around 95% while energy consumption overheads are 35%.
5. The performance improvements of KDDCS scale proportionally with the network size.

The results of the simulations are shown in Figures 77 to 86. Recall that DIM already outperforms both the LS and the GHT schemes. That's why we only plot the KDDCS performance together with those of the DIM and ZS/ZP/ZPR schemes. In these figures, we compare the performance of KDDCS with that of the basic DIM and that the ZS/ZP/ZPR, with respect to our different performance measures. For simplicity, we will refer to ZS/ZP/ZPR by ZS/ZP/ZPR.

We present the results of our study in the next four subsections.

5.8.1.1 Uniform Loads We conclude our performance study by studying the KDDCS performance for uniform loads. Figure 77 compares KDDCS with both DIM and ZS/ZP/ZPR in terms of average node energy for uniform loads. The figure shows that the differences

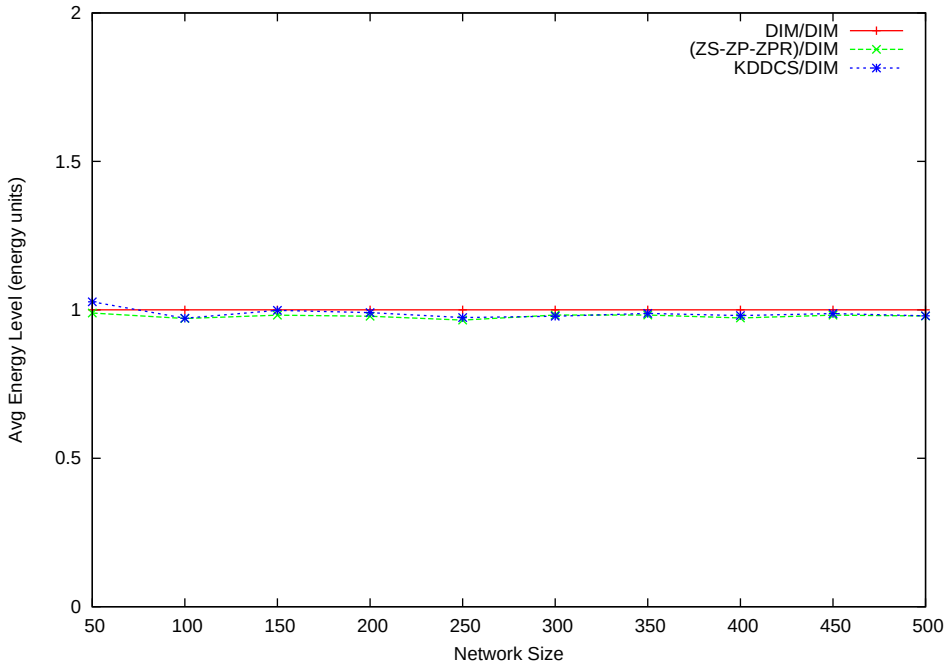
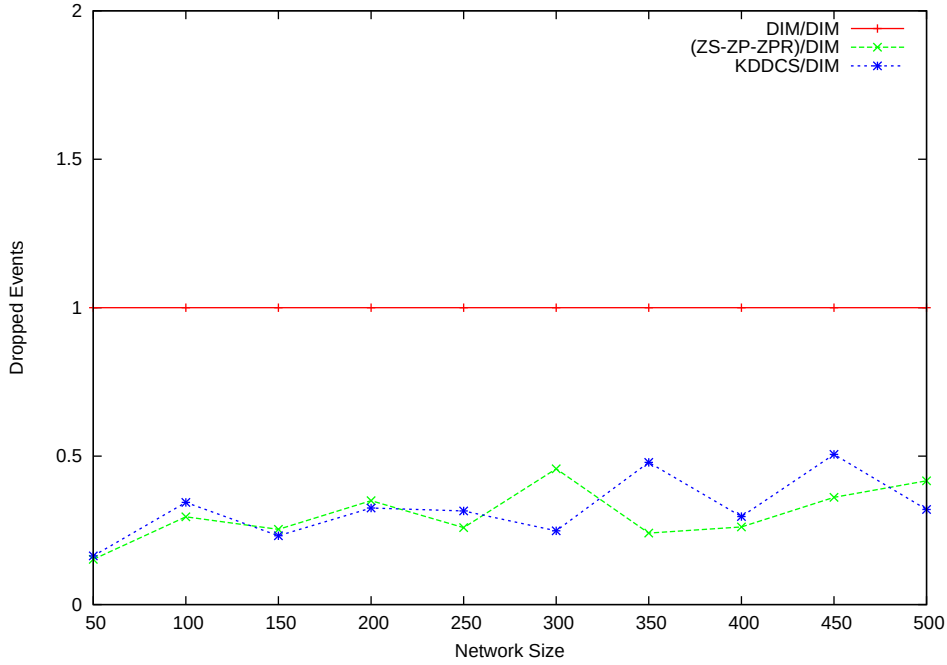


Figure 77: KDDCS: Average Node Energy for Uniform Loads

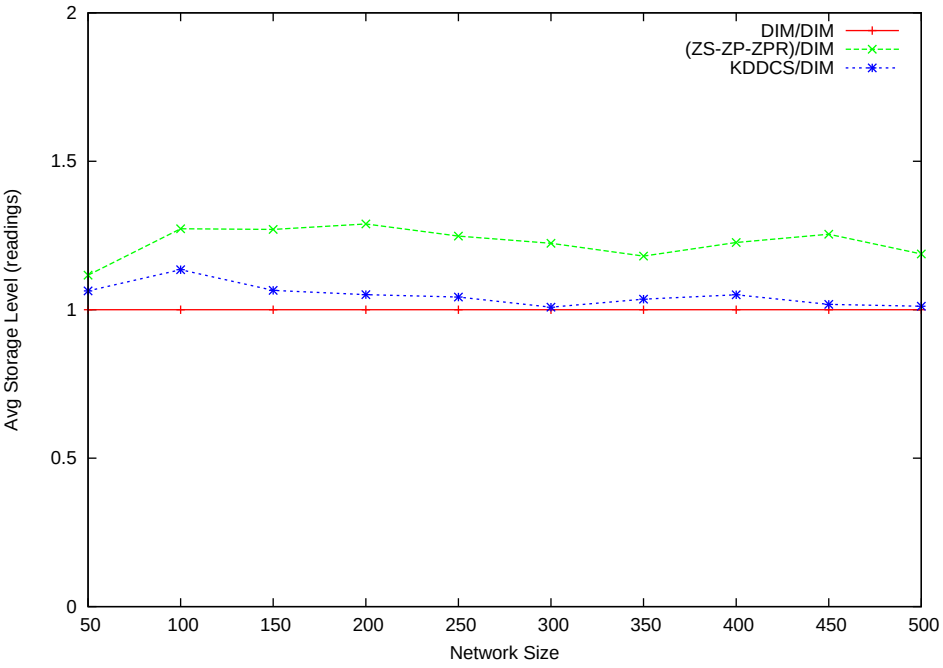
between the schemes are very small. This simply shows that KDDCS does not add any additional energy consumption burden on the sensor network, compared to the DIM scheme, when the network experiences no hotspots.

5.8.1.2 Single Static Mixed Hotspots The following three results compare the KDDCS performance to those of DIM and ZS/ZP/ZPR for single static mixed hotspots. We conduct our study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

R1. QoD: Figures 78(a) and 78(b) compare the performance of the three schemes in terms of dropped readings and average node storage for a 60% single mixed hotspot. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The first figure shows that KDDCS achieves a similar



(a) Dropped Events for a 60% Single Mixed Hotspot



(b) Average Node Storage for a 60% Single Mixed Hotspot

Figure 78: KDDCS: QoD Graphs vs Single Mixed Hotspots

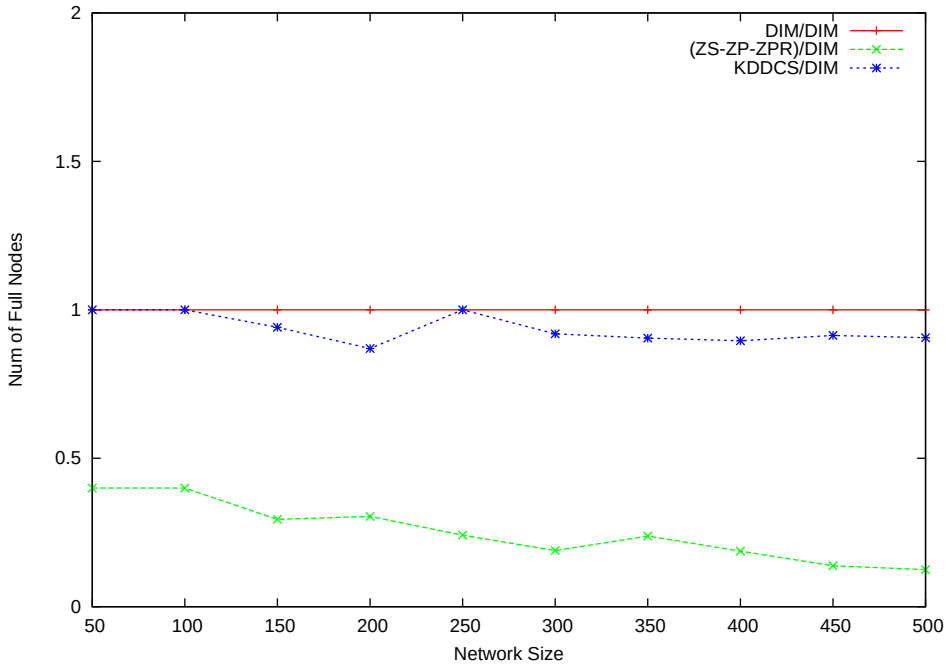


Figure 79: KDDCS: Number of Full Nodes for a 60% Single Mixed Hotspot

performance improvement over DIM to that achieved by ZS/ZP/ZPR (in terms of dropped events). Basically, KDDCS improves QoD by around 75%. The second figure shows that KDDCS increases the average node storage by around 8% compared to the basic DIM scheme. The two figures show the QoD improvement achieved by KDDCS compared to DIM for single mixed hotspots. The important thing to note is that the amount of QoD improvement for single mixed hotspots is equal to that achieved for the case of single storage hotspots and much better than that achieved for single query hotspots. This shows the high benefits of global load balancing when dealing with hotspots of complex structures. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

R2. Load Balancing: Figure 79 shows the number of full nodes of the three schemes for a 60% single mixed hotspots. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The number of full nodes for KDDCS is almost equal to (or at most 5% less than) that of DIM for most network

sizes. The important observation in this figure is that KDDCS continues to achieve a better load balancing of the hotspot data compared to ZS/ZP/ZPR by increasing the number of nodes engaged in the storage responsibility of the hotspot data and thus increasing the overall number of full nodes compared to the ZS/ZP/ZPR scheme. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

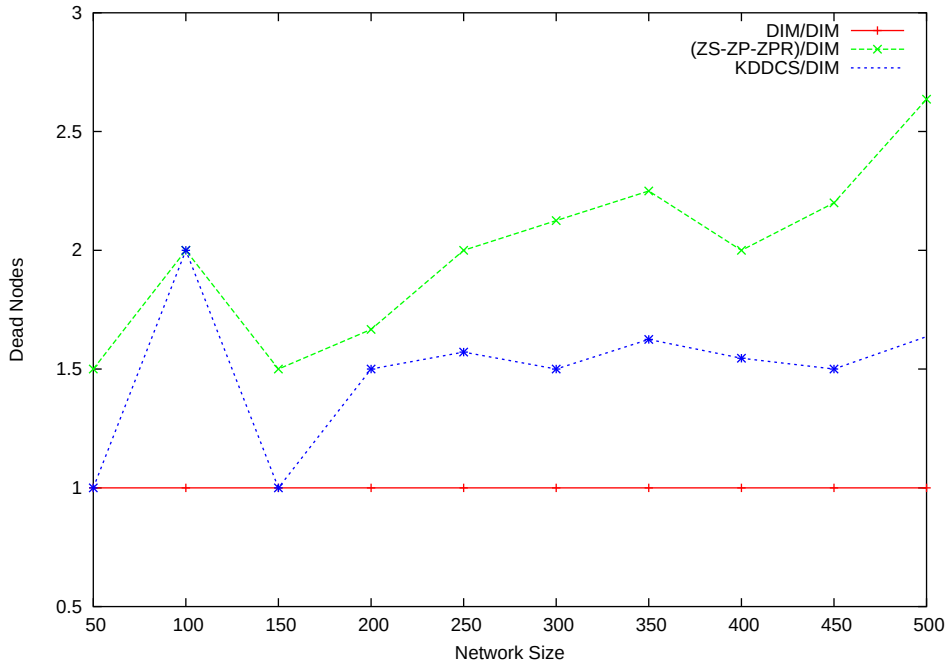
R3. Energy Consumption: Figure 80(a) and 80(b) compare the dead nodes and the average node energy of the three schemes for a 60% single mixed hotspot. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The first figure shows that KDDCS increases the number of dead nodes by around 40% compared to the DIM scheme. The second figure shows that KDDCS load-balances the energy consumption among a larger number of sensor nodes, thus, reduces the average node energy level of the different nodes in the sensor network by around 18% compared to DIM. The two figures show that the energy consumption overhead of KDDCS is moderate in the case of single mixed hotspots. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

Overall, KDDCS largely improves the QoD, by around 75%, compared to the DIM scheme for single mixed hotspots. This comes with a slight energy consumption overhead of about 18% over DIM.

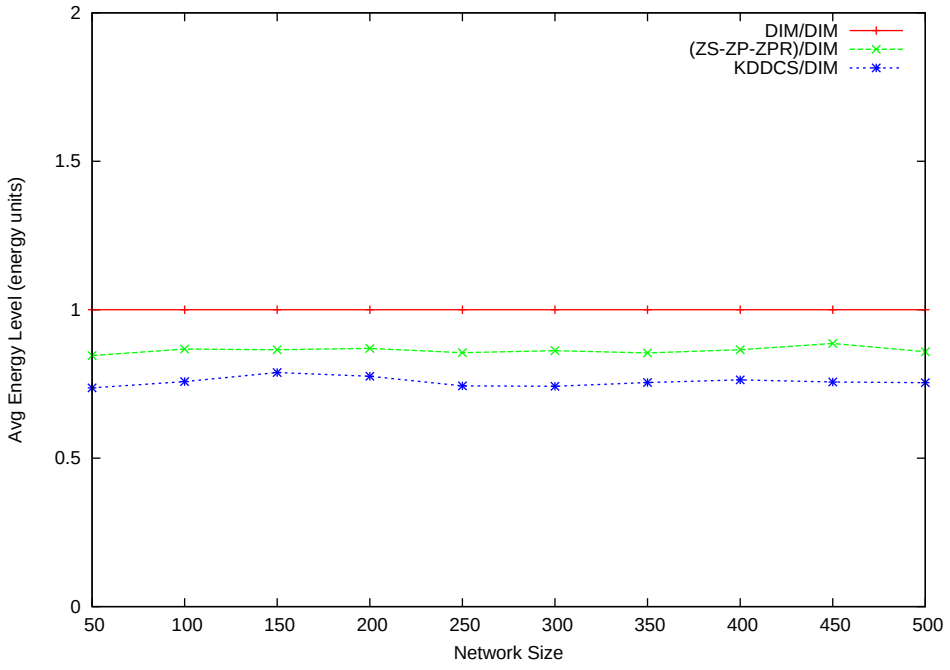
5.8.1.3 Multiple Simultaneous Static Mixed Hotspots

The following three results compare the KDDCS performance to those of DIM and ZS/ZP/ZPR for multiple static mixed hotspots. We conduct our study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

R1. QoD: Figures 81(a) and 81(b) compare the performance of the three schemes in terms of dropped readings and average node storage for an 80% mutiple mixed hotspot. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a

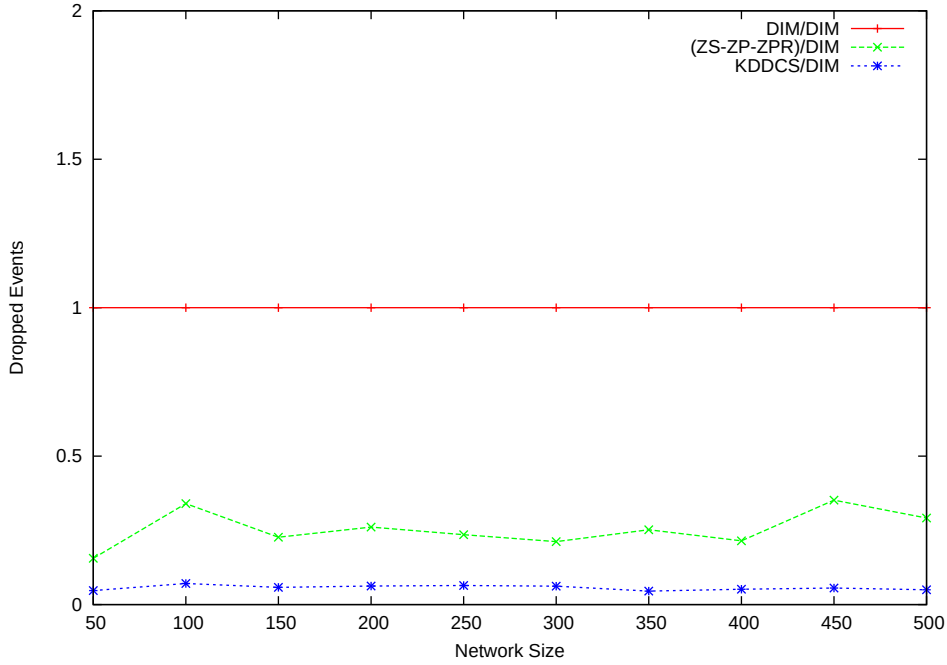


(a) Dead Nodes for a 60% Single Mixed Hotspot

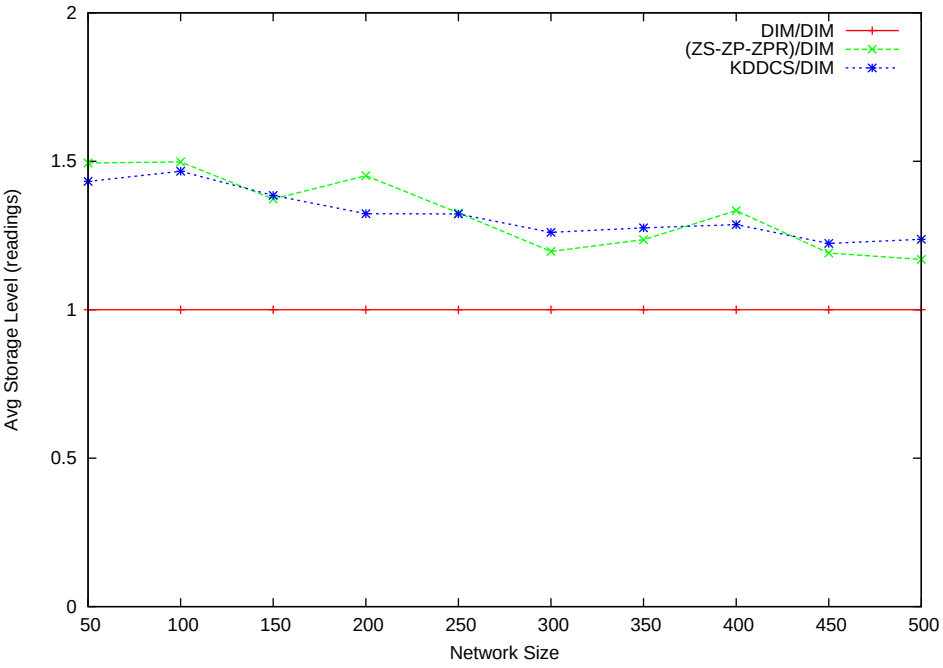


(b) Average Node Energy for a 60% Single Mixed Hotspot

Figure 80: KDDCS: Energy Consumption Graphs vs Single Mixed Hotspots



(a) Dropped Events for 80% Multiple Mixed Hotspots



(b) Average Node Storage for 80% Multiple Mixed Hotspots

Figure 81: KDDCS: QoD Graphs for Multiple Mixed Hotspots

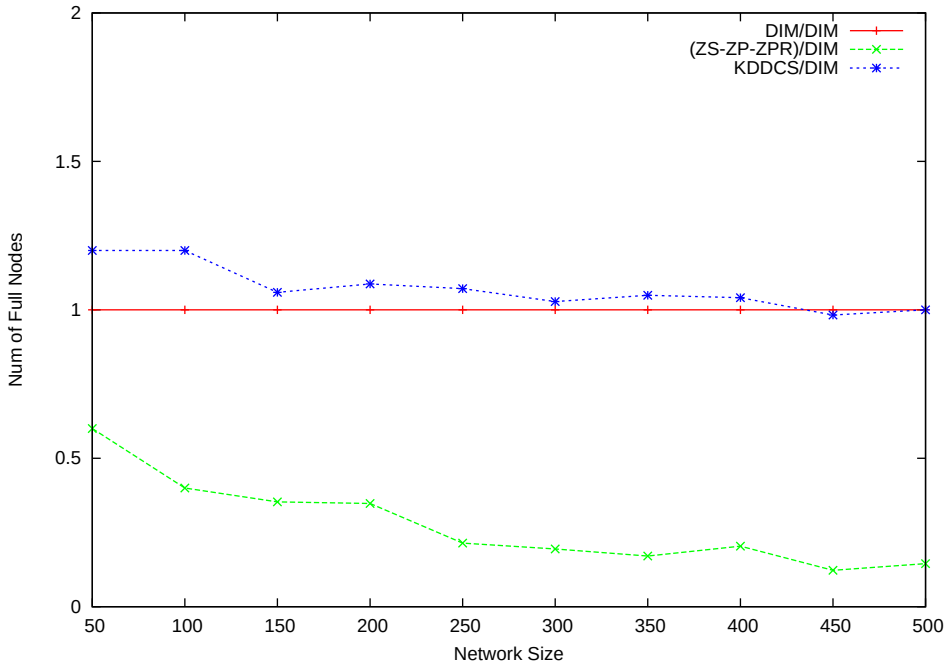


Figure 82: KDDCS: Number of Full Nodes for 60% Multiple Mixed Hotspot

scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The first figure shows that KDDCS highly outperforms the DIM scheme and slightly improves the performance of the ZS/ZP/ZPR scheme in terms of dropped events. KDDCS improves DIM's QoD by around 95%. The important observation in this figure is how much KDDCS scales with the network size. The second figure shows that KDDCS experiences the highest average node storage among the three schemes. KDDCS is 50% better than DIM in terms of average node storage. This is an important indication that KDDCS exhibits a higher data persistence as compared to the other two schemes. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

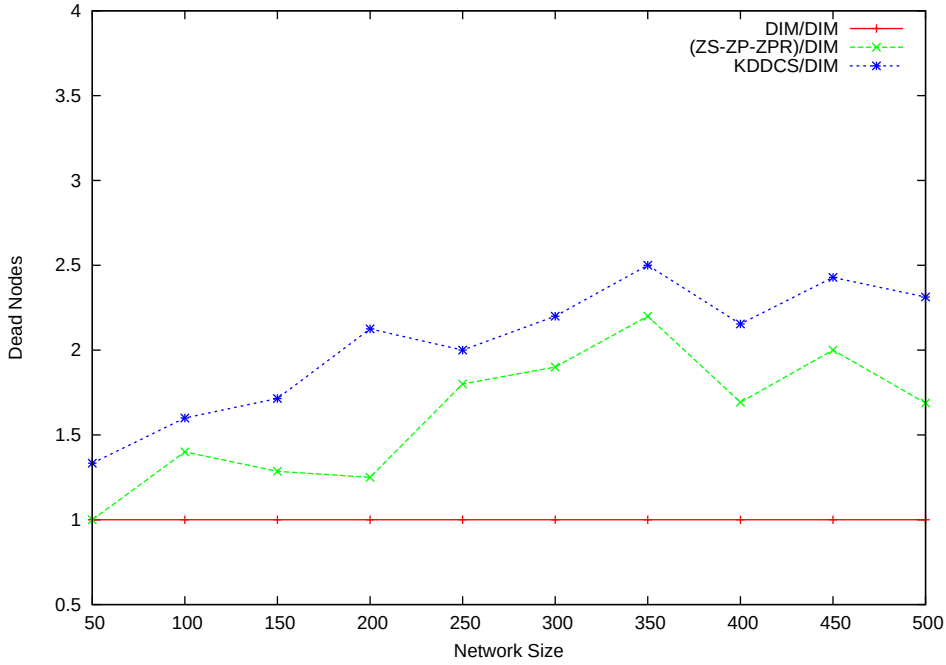
R2. Load Balancing: Figure 82 shows the number of full nodes of the three schemes for 60% multiple mixed hotspots. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The figure shows the high load balancing that KDDCS achieves compared to ZS/ZP/ZPR in terms of

highly increasing the number of full nodes. This is due to engaging a lot of nodes in storing the hotspot data. As the number continues to grow, many of these nodes reach their full capacity. Though the number of full nodes of KDDCS are almost the same as that of DIM for the different network sizes, it should be noted that the geographic distribution of these nodes is completely different between the two schemes. This is because the full nodes in the DIM scheme are those falling in the original hotspot area, while the full nodes in the KDDCS scheme are a subset of the nodes falling in the larger hotspot area (achieved after the repetitive tree rebalancing processes). We achieved similar results for hotspots of sizes ranging from 50% to 100%.

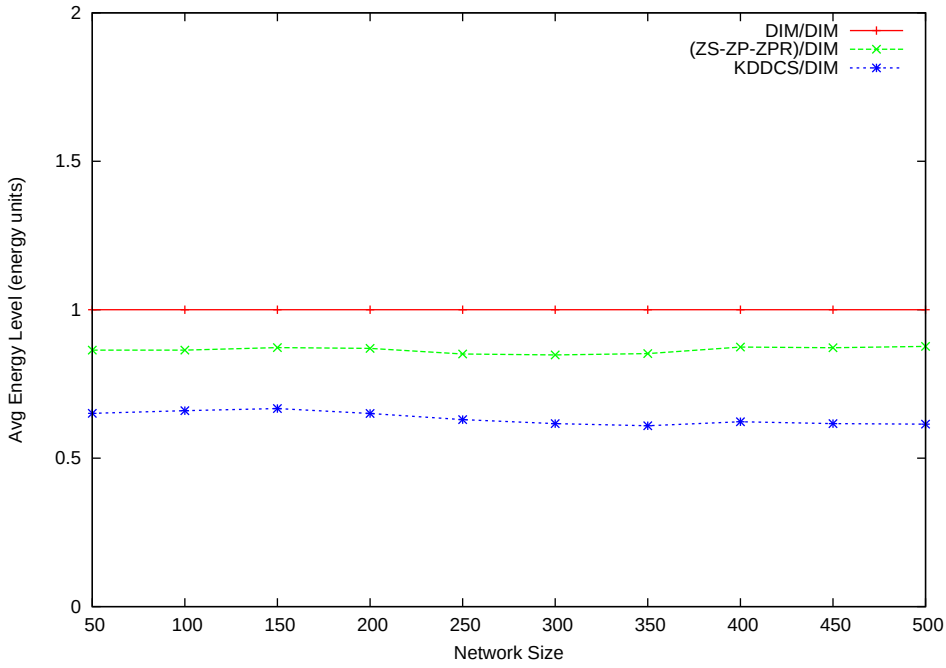
R3. Energy Consumption: Figure 83(a) and 83(b) compare the dead nodes and the average node energy of the three schemes for 60% multiple mixed hotspots. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The first figure shows that KDDCS at least doubles the number of dead nodes compared to DIM. However, the number of dead nodes for DIM is at most 3% of the network size (for all network sizes). This means that the performance degradation imposed by KDDCS in terms of additional dead nodes is relatively low (around 3% of the network size). The second figure shows that KDDCS continues to load-balance energy consumption across a larger number of network nodes and thus decreases the average node energy. Basically, the energy consumption overhead imposed by KDDCS is around 35% more than that of DIM. Both figures show that KDDCS incurs a moderate energy consumption overhead compared to the other two schemes in the case of multiple mixed hotspots. We achieved similar results for hotspots of sizes ranging from 50% to 100%.

Overall, KDDCS achieves very good QoD performance improvements of around 95% over DIM with a moderate energy consumption overhead of around 35%.

5.8.1.4 Moving Mixed Hotspots The following three results compare the KDDCS performance to those of DIM and ZS/ZP/ZPR for moving mixed hotspots. We conduct our



(a) Dead Nodes for 60% Multiple Mixed Hotspots



(b) Average Node Energy for 60% Multiple Mixed Hotspots

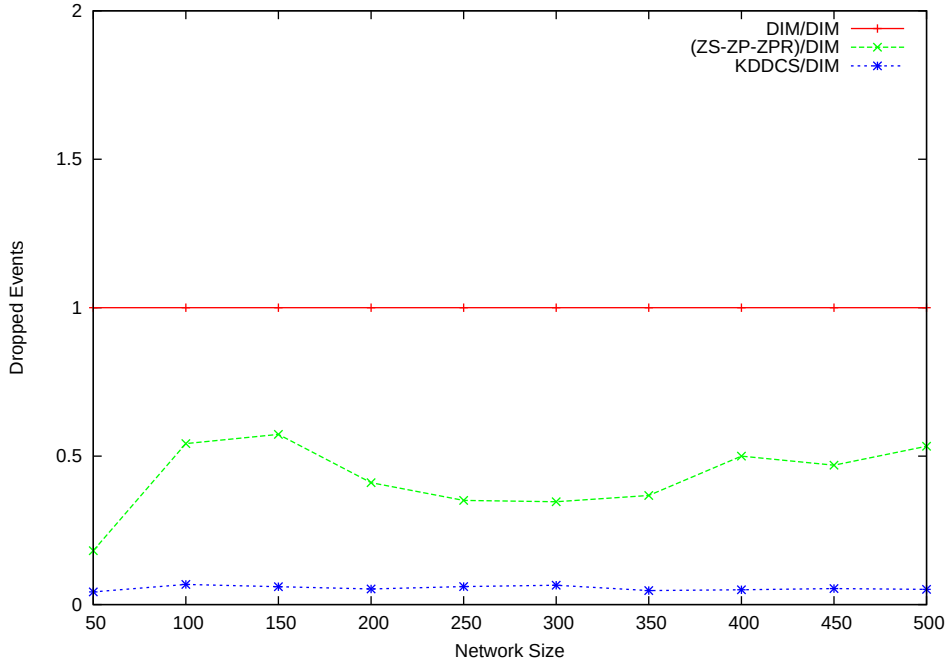
Figure 83: KDDCS: Energy Consumption Graphs vs Multiple Mixed Hotspots

study in terms of QoD (R1), Load Balancing (R2), and energy consumption (R3).

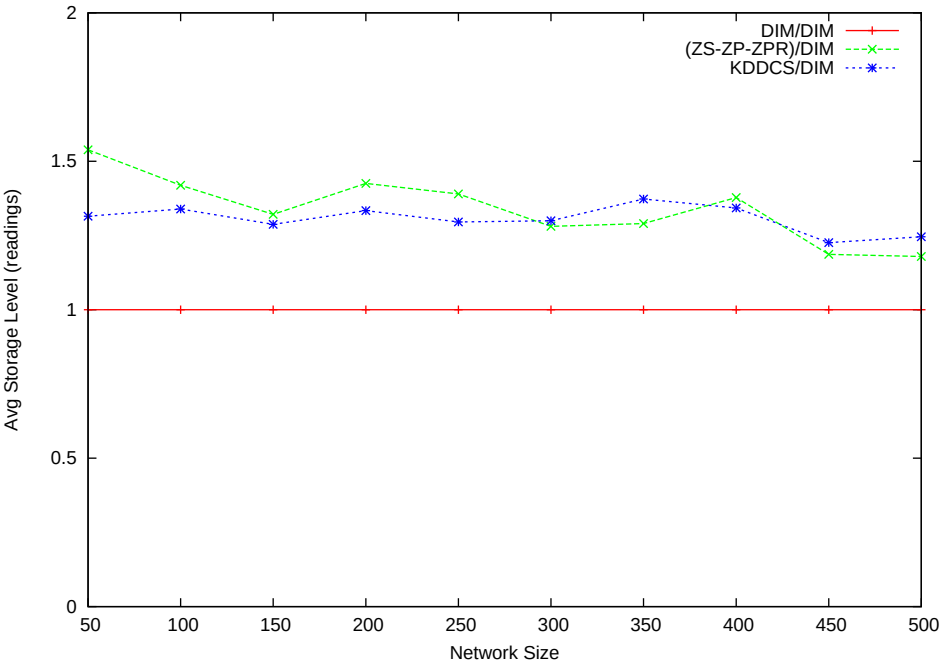
R1. QoD: Figures 84(a) and 84(b) compare the performance of the three schemes in terms of dropped events and average node storage for an 80% moving mixed hotspot. Recall that we use the number of dropped events as a direct indication of the QoD of any scheme. A scheme dropping less readings is achieving a better data persistence. This helps such a scheme to provide more accurate and complete query answers. Consequently, this would improve the scheme's QoD. The first figure shows that KDDCS improves QoD by around 95% over DIM. The second figure shows that KDDCS achieves a 30% higher average node storage than DIM. This shows that KDDCS highly increases the data persistence as compared to the two other schemes. We achieved similar results for hotspots of sizes up to 80%.

R2. Load Balancing: Figure 85 shows the number of full nodes of the three schemes for an 80% moving mixed hotspot. Recall that a full node is a one storing the maximum storage capacity. The number of full nodes represents an indication on the capability of any scheme to load balance the hotspot data among a larger number of sensor nodes. The figure shows that KDDCS multiplies the number of full nodes between 4 and 5 times compared to DIM. This shows the high load balancing gain that KDDCS achieves compared to the other two schemes by highly increasing the number of nodes participating in the storage of the hotspot data and consequently increasing the overall number of full nodes in the network. The important observation is that the number of full nodes for KDDCS is linear in the network size while it is almost constant for the other two schemes. We achieved similar results for hotspots of sizes up to 80%.

R3. Energy Consumption: Figure 86(a) and 86(b) compare the dead nodes and the average node energy of the three schemes for 80% and 60% moving mixed hotspot, respectively. Recall that we measure the average node energy in terms of energy units, where the unit is the amount of energy needed to send (or receive) one sensor reading. We use the average node energy as an indication for the energy consumption overhead imposed by a scheme on each node in the sensor network. On the other hand, node deaths give an important indication on the QoS improvement that any scheme achieves. The figures show that KDDCS incurs a moderate energy consumption overhead in terms of a moderate increase in the number of dead nodes (ranging from twice to two and half times that of



(a) Dropped Events for an 80% Moving Mixed Hotspot



(b) Average Node Storage for an 80% Moving Mixed Hotspot

Figure 84: KDDCS: QoD Graphs vs Moving Mixed Hotspots

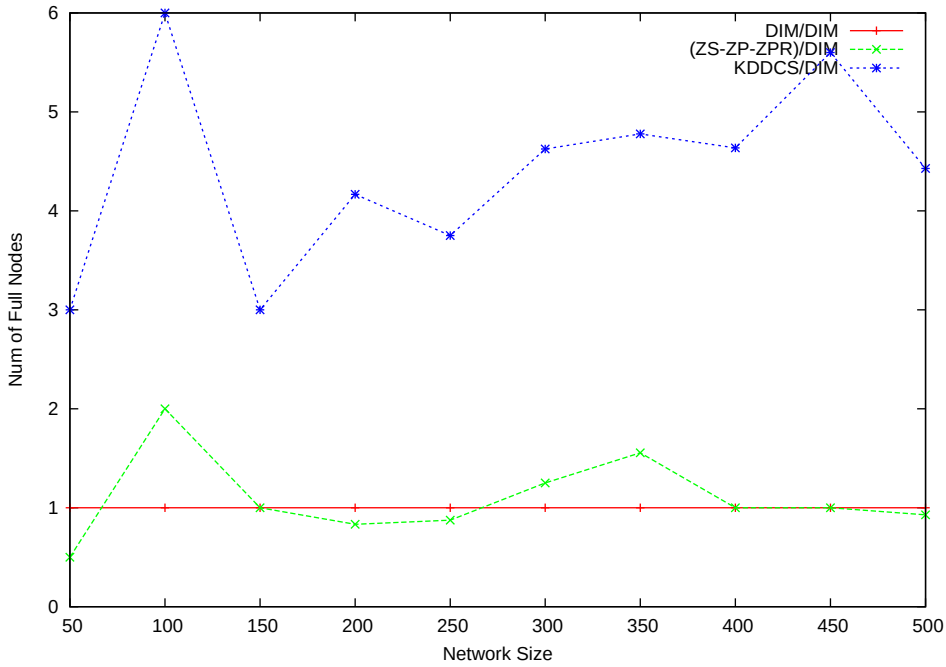
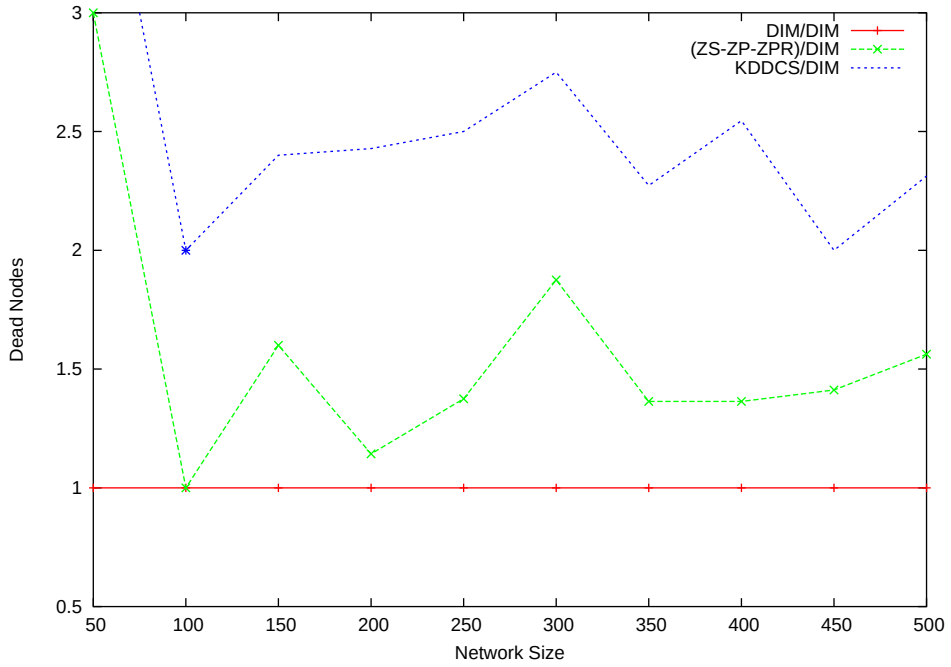


Figure 85: KDDCS: Number of Full Nodes for 80% Moving Mixed Hotspots

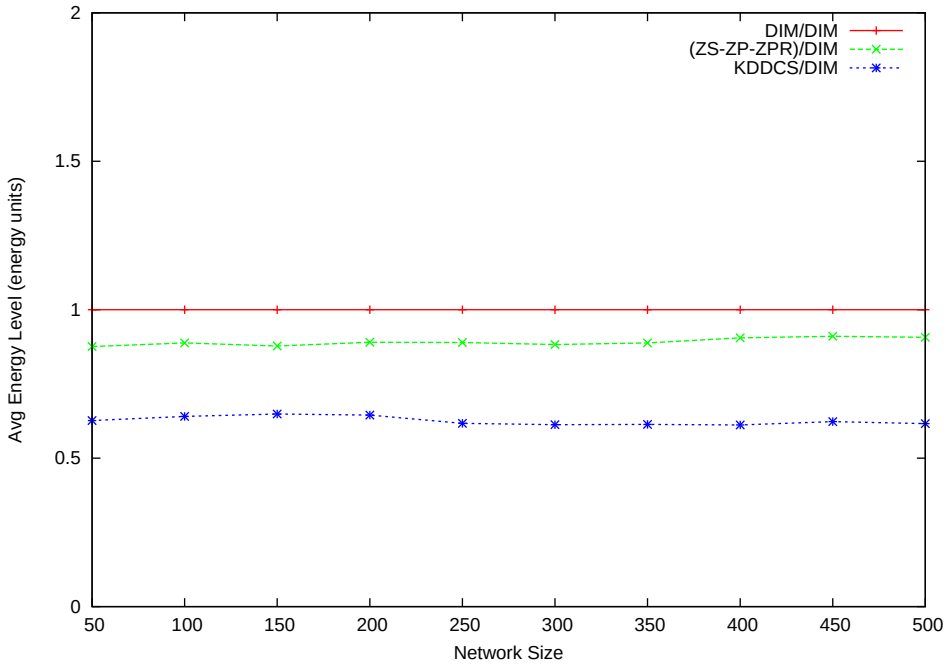
DIM) and a decrease in the average node energy by around 35%. As for the number of dead nodes of DIM, it is at most 3% of the network size (for all network sizes). Thus, KDDCS degrades the performance (in terms of dead nodes) by at most 4.5% of the network size. This can safely be considered as a small to moderate overhead depending on the network size. The important observation about the second figure is that KDDCS continues to load-balance energy consumption across the different sensor nodes of the network. This results in decreasing the average node energy of the sensor nodes across the network. We achieved similar results for hotspots of sizes up to 80%.

In conclusion, KDDCS achieves a high QoD performance gain of around 95% over DIM at the cost of introducing a moderate energy consumption overhead on the sensor network of around 35%.

5.8.1.5 Discussion In this section, we studied the KDDCS performance for single, multiple, and moving mixed hotspots. Overall, KDDCS is highly able to cope with mixed



(a) Dead Nodes for an 80% Moving Mixed Hotspot



(b) Average Node Energy for a 60% Moving Mixed Hotspot

Figure 86: KDDCS: Energy Consumption Graphs vs Moving Mixed Hotspots

Table 8: KDDCS Performance (Relative to DIM) for Mixed Hotspots

Hotspot Type	QoD Improvements	QoS Overheads
Single Static Hotspots	75%	18%
Multiple Static Hotspots	95%	29%
Moving Hotspots	95%	35%

hotspots of different types and sizes. KDDCS scores good QoD achievements for the different hotspot types and sizes while introducing a tolerable energy consumption overhead on the sensor nodes across the network. For single hotspots, QoD improvements are around 75% over DIM while energy consumption overheads are 18% over DIM. For multiple static hotspots, QoD improvements are around 95% over DIM while energy consumption overheads are around 29% DIM. For moving hotspots, QoD improvements are around 95% over DIM while energy consumption overheads are 35% over DIM. Unlike ZS/ZP/ZPR, the benefits of KDDCS becomes quite obvious for multiple and moving mixed hotspots. Table 8 summarizes the KDDCS performance for mixed hotspots.

5.9 KDDCS ROBUSTNESS TO PACKET LOSS

To make our algorithms resilient to packet loss, a simple ACK scheme can be adopted. In this scheme, ACKs are requested for reading insertions as well as for queries. For insertions, the scheme is obvious as the ACK would be required from the corresponding storage node (as well as the replica node if replication is applied). Considering queries, the query issuer receives ACKs from all the nodes responsible for storing the sub-zones of the queried zone. In case one of the ACKs is not received, the query can be resent to that node or to its replica.

5.10 LEARNED LESSONS FROM THE EXPERIMENTAL EVALUATION

In this section, we will discuss some of the insights than can be taken from the experimental evaluation of KDDCS scheme. Our main goal is to study the compare the benefits of implementing our scheme to those of implementing the ZS/ZP/ZPR scheme and also compare the implementation overheads of both schemes. Also, we would like to determine the network and hotspot settings fitting for each of the schemes.

Before discussing the learned lessons out of our experimental evaluation, we summarize our experimental results in the following points:

1. KDDCS scores good QoD achievements for the different hotspot types and sizes while introducing a tolerable energy consumption overhead on the sensor nodes across the network.
2. KDDCS energy consumption performance is almost the same as that of DIM in the case of uniform loads.
3. For single hotspots, QoD improvements are around 75% for single storage hotspots, 13% for single query hotspots, and 75% for single mixed hotspots while energy consumption overheads are 1%, 2.5%, and 18% respectively.
4. For multiple static hotspots, QoD improvements are around 25% for multiple storage hotspots, 87% for multiple query hotspots, and 95% for multiple mixed hotspots while energy consumption overheads are 1%, 15%, and 35% respectively.
5. For multiple dynamic (moving hotspots), QoD improvements are around 60% for moving storage hotspots, 35% for moving query hotspots, and 95% for moving mixed hotspots while energy consumption overheads are 13%, 3%, and 35% respectively.
6. The performance improvements of KDDCS scale proportionally with the network size.

Table 9 summarizes the KDDCS performance for the different hotspot types.

In general, experiments have shown that the KDDCS achievements in terms of improving the QoD by far exceed those of ZS/ZP/ZPR when implemented on top of DIM. This is basically valid for most hotspot types and sizes. Furthermore, KDDCS showed very good

Table 9: KDDCS Performance (Relative to DIM) for the Different Hotspot Types

Hotspot Type		QoD Improvements	QoS Overheads
Storage Hotspots	Single Static Hotspots	75%	1%
	Multiple Static Hotspots	25%	1%
	Moving Hotspots	60%	13%
Query Hotspots	Single Static Hotspots	13%	2.5%
	Multiple Static Hotspots	87%	15%
	Moving Hotspots	35%	3%
Mixed Hotspots	Single Static Hotspots	75%	18%
	Multiple Static Hotspots	95%	35%
	Moving Hotspots	95%	35%

results in load balancing multiple hotspots, both static and dynamic, unlike ZS/ZP/ZPR whose achievements were limited for such hotspot types.

Experiments have also shown that the energy consumption overhead of KDDCS is fairly varying for the different hotspot types. In the case of single hotspots, KDDCS does not impose a remarkable energy consumption overhead on the sensor network. In fact, the KDDCS energy consumption overhead is in the same range of that imposed by ZS/ZP/ZPR. It is better for single storage and single query hotspots and a little bit higher for the case of multiple mixed hotspots. The case is a little bit different for multiple hotspots (mainly multiple query and multiple mixed hotspots) as the energy consumption overheads imposed by KDDCS is a little bit higher than those of ZS/ZP/ZPR. However, it should be noted that we test the worst case scenario for KDDCS, in which KDDCS starts with a uniform assignment of zones to sensor nodes. This should not be the version to be used in case large-scale hotspots are expected to arise in pre-known attribute ranges (known prior to the network operation). Starting with a more skewed assignment of zone responsibilities to sensor nodes would highly reduce the energy consumption overhead imposed by KDDCS as it will reduce the scale on which KDTR is applied at the early phases of the network operation. Our

experimental results shows that using a pre-known distribution could improve the KDDCS energy consumption overheads by large amounts that could reach to 80% in some cases.

Another important observation is the effect of the value of the h threshold on the KDDCS performance. Though we tested a range of values for the rebalancing ratio h, our results showed that changing the value of h did not have a major effect on the KDDCS performance. This can be explained by the type of hotspots that we simulated. As our hotspots usually fall in relatively small subranges of the possible attribute ranges, satisfying the rebalancing criteria of the KDTR algorithm only requires a small h value in most of the cases. Our experiments showed that this results in almost no difference in performance between h values ranging from 1.5 to 3. The effect of h should be more obvious in the cases where the skewness of the hotspot distribution follows a more smooth distribution, e.g., a normal distribution with relatively large variance.

We now discuss the settings in which KDDCS excelled the most. As for network sizes, the KDDCS performance was similar to that of DIM for small networks (less than 100 nodes). The KDDCS advantage is higher for larger networks with the improvements increasing with the network size. As for hotspot sizes, KDDCS, unlike ZS/ZP/ZPR, had the ability to deal with hotspots of different sizes. In fact, the actual hotspot size does not really matter as KDDCS deals with the hotspots very early during their formation. This is why KDDCS can be considered able to avoid hotspots of different sizes. Concerning sensor network applications, KDDCS is best suited for investigation sensor networks where the value of the sensor data is relatively higher than that of the sensor energy.

In general, KDDCS is a load balancing scheme that is able to cope with all types of hotspots and network sizes. Its application is very encouraged for networks that are expected to frequently face hotspots of different types, lengths, and sizes.

5.11 SUMMARY

In this chapter, we presented KDDCS, a globally load-balanced DCS scheme whose main design goal is to avoid the formation of hotspots of the different types, including storage,

query and mixed hotspots. KDDCS avoids hotspots, i.e., successfully deals with them in their early stages of formation, by continuously load balancing the underlying k-d tree. Furthermore, we proposed a new routing algorithm, namely the *Logical Stateless Routing*, for routing events from the generating sensors to the storage sensors. LSR is competitive with the popular GPSR routing. Our experimental evaluation has confirmed that our proposed KDDCS both increases the QoD in the different hotspot cases without adding a large energy consumption overhead on the sensor network. KDDCS does not depend on the storage or the energy capacity distribution in the sensor network.

6.0 CONCLUSIONS

In this chapter, we conclude our dissertation by briefly presenting the summary of our contributions. We also highlight our future work, including both the possible extensions to our thesis work as well as the future research directions related to our work.

6.1 SUMMARY OF CONTRIBUTIONS

In this dissertation, we address the hotspot problems in the Data-Centric Storage (DCS) sensor network model by means of *load balancing*. The thesis identifies two important problems in sensor networks implementing DCS, namely *storage hotspots* and *query hotspots*. Both problems represent different forms of data skewness in the sensor network. Storage hotspots arise because of the skewness of the distribution of the sensor reading values. This skewness results in assigning a large amount of sensor readings for storage to a small amount of sensor nodes. Query hotspots represent a skewness in data popularity where a large percentage of the query load ask for the data stored in a small subset of the sensor nodes. Both problems primarily downgrade the Quality of Data (QoD) of the sensor network with a secondary negative effect on the Quality of Service (QoS) of the sensor network.

Through the design, development, and evaluation of a set of simple easy-to-implement distributed load balancing schemes, the thesis is supported. The key contribution of this thesis is the concept of data migration and using it in load balancing both storage and query hotspots. The usage of the data migration concept comes in two types of schemes: local and global. Local schemes act as hotspot detection and decomposition schemes. Once a hotspot is detected by any of these schemes, a portion of the hotspot data is migrated away from the

hotspot area as a local remedy without disturbing the underlying index structure of the DCS scheme. In this direction, we present three schemes to deal with storage, query, and mixed hotspots. These schemes are Zone Sharing (ZS), Zone Partitioning/Zone Partial Replication, and ZS/ZP/ZPR (resulting from the mix of the first two schemes), respectively. Our local schemes are highly characterized by their low implementation and messaging overheads.

The second type of schemes that we present in this thesis are global schemes. We mainly present the K-D tree based Data-Centric Storage (KDDCS) scheme to globally load-balance all types of hotspots, including storage, query, and mixed hotspots. Load balancing in KDDCS is based on defining and distributively solving a theoretical problem that we call the *Weighted Split Median* problem. Continuously solving the problem in a distributive manner guarantees that the different subtrees of the k-d tree will remain balanced, to a constant factor. One advantage for our schemes, both local and global, is that they work well for all networks, regardless of the distribution of node storage capacities.

Experimental evaluation, through extensive simulations, showed the efficiency of our schemes to deal with hotspots of different types. Due to their heuristic nature, local schemes are best suited for decomposing single isolated static hotspots of small to medium sizes. Their implementation on top of the DIM scheme achieves moderate QoD improvements while imposing a slight energy consumption overhead on the sensor network. On the other hand, global schemes are able to avoid the formation of hotspots, i.e., load-balance the hotspot very early in its formation. Experiments show the KDDCS capability to effectively cope with all hotspot settings, including single and multiple hotspots (both static and dynamic), as well as all hotspot sizes. KDDCS achieves high QoD improvements for all hotspot types at the cost of introducing a moderate energy consumption overhead on the sensor network.

The contributions of our thesis are both *theoretical* and *practical*. The main theoretical contribution of this thesis lies in presenting a new k-d tree distributed rebalancing algorithm, namely the K-D Tree Rebalancing (KDTR) algorithm, based on defining and distributively solving the *Weighted Split Median* problem. The running time of the KDTR algorithm 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. Although presented in the context of sensor networks, KDTR can easily be used

in load balancing k-d trees used in general computer systems and networks, e.g. distributed systems, peer-to-peer networks [49], and mobile ad-hoc networks.

On the practical level, the schemes presented in our thesis work offer important guidelines and easy-to-implement tools to help network designers and operators in successfully dealing with the hotspot problem according to its expected severity. We highlight the practical contributions of our thesis as well as the usage of each of them in the following points:

- Local hotspot detection and decomposition schemes developed in our thesis, i.e., ZS, ZP/ZPR, and ZS/ZP/ZPR, are recommended for sensor networks with low hotspot formation probability, low data urgency, long expected network lifetime, and with the energy value relatively higher than that of data, e.g., sensor networks for regular monitoring applications.
- The KDDCS developed in this thesis is recommended for sensor networks with high hotspot formation probability, high data urgency, short expected network lifetime, and relatively low energy value (compared to data), e.g., sensor networks for disaster management. KDDCS is considered an important step towards developing a robust hotspot-free DCS scheme.
- In addition to load balancing, the KDTR algorithm allows KDDCS to naturally take care of node deaths through continuous load balancing. Thus, KDTR acts as a first level of fault tolerance for KDDCS. This advantage can be exploited when exporting KDTR to other contexts, e.g., peer-to-peer networks.
- As part of the KDDCS scheme, we developed a new virtual routing scheme, namely the Logical Stateless Routing (LSR) scheme, for the routing of events from their generating sensors to their storage sensors in the KDDCS scheme. LSR is competitive with the GPSR routing scheme. Due to the continuous load balancing in KDDCS, LSR has the potential of following different routing paths for identical requests arising in different points in time. LSR can easily be used in general networking applications, both wired and wireless.

Finally, in the context of our disaster management application, our recommendation is to adopt the KDDCS scheme as the solution for hotspots of all types. KDDCS provides the

best possible trade between QoD and QoS when the quality and timeliness of information needed to determine how to save lives is more important than the energies and lifetimes of sensor nodes.

6.2 FUTURE WORK

In this section, we highlight possible experimental studies that we plan to conduct for the different schemes presented in this dissertation in the near future. The main goal of these experiments is to further study the benefits of our schemes when applied in new contexts, namely on sensor network testbeds, on heterogeneous sensor networks, and on sensor networks experiencing failures, respectively. In the second part of this section, we discuss some of the future research directions that are complementary to our thesis work.

6.2.1 Future Experimental Studies of the Thesis Schemes

6.2.1.1 Experimenting on Sensor Network Testbeds To fully test the performance of our schemes in real-world settings, we plan to implement them on one of the currently available wireless sensor network testbeds, e.g., MoteLab [39]. We will also try to find real traces for the different experiments conducted in this dissertation.

6.2.1.2 Experimenting on Heterogeneous Networks When evaluating the local schemes presented in this thesis, our experimental evaluation assumed a homogeneous network of sensor nodes where the storage responsibility is shared among all nodes. Depending on the sensor types presented in Chapter 3, our assumed sensor network can be either a network of motes or a network of microservers. However, a typical real-world sensor network can consist of both, microservers and motes. The possible new features of heterogeneous networks are as follows. First, the storage responsibility in heterogeneous networks can solely assigned to microservers. Second, the energy capacities are not the same among all nodes in this network type. Instead, possibly rechargeable microservers have much more available

energy than that of motes. It is clear that our schemes can easily be applied in heterogeneous sensor networks with no major changes needed. In this section, we describe how our schemes can be applied in heterogeneous networks and highlight the changes needed for our schemes to be fit for such networks.

ZS/ZP/ZPR for Heterogeneous Networks: In this section, we present the changes needed for the ZS/ZP/ZPR scheme to be applied in heterogeneous networks. This includes the changes applied to both the ZS and the ZP/ZPR schemes to be separately or collectively applied in heterogeneous networks. The fact that microservers are the ones responsible for storage makes heterogeneous networks look like overlay networks where a backbone of microservers is available in the network and responsible for storing readings and answering queries while the rest of the network is composed of motes responsible for sensing information. Furthermore, the microservers should be able to communicate among each other using the 802.11 radio rather than using the 802.5 one, thus, without involving motes. We use these two important facts in our discussion below.

We start by discussing the changes needed to be applied to the DIM scheme for handling heterogeneous networks. At the beginning of the network operation, each microserver needs to know its direct neighbors. Furthermore, the storage responsibility of the attribute ranges should be partitioned among microservers only. Thus, the address assignment of the DIM scheme is only applied among microservers. At the end of this process, each microserver has a bit-code address representing its zone responsibility.

To route events from motes to the corresponding microservers, each mote needs to be first associated with one of the zones. Thus, at the end of the address assignment phase, each microserver initiates a broadcast message notifying surrounding motes with its address. Each mote associates itself with the closest microserver to its geographical position. To route any event, the mote uses its address as well as the attribute values of the event to determine the correct direction to which the event should be routed exactly as in the DIM scheme.

We now describe how to apply our schemes to this modified DIM version. Periodically, each microserver compares its storage with its direct neighboring microservers. In case a hotspot arises, it is dealt with using one of our local schemes. Each of the microservers involved in a ZS or ZP then updates the SZLs or TZLs of its neighboring microservers as

well as those of its neighboring motes. To route an event, a node first checks its zones' list to determine whether the event's storage responsibility has been changed or no. If so, the event is routed to its new storage sensor. Otherwise, it is routed to its original one.

In the future, we plan to experimentally study the performance of ZS/ZP/ZPR on heterogeneous networks. It is important to note that this modified version of our ZS/ZP/ZPR scheme relies on the important facts that microservers have much more available storage and energy than motes and that event relocations will not involve any motes. However, in case these assumptions are not valid, the energy consumption load imposed on motes due to the event relocation process applied by our schemes may become much more dangerous on both the QoD and QoS of the network compared to the benefits achieved from applying our schemes. In such a case, living with the dangers of hotspots may be a better option.

KDDCS for Heterogeneous Networks: Our KDDCS experimental evaluation assumed a homogeneous network of sensor nodes where the storage responsibility is shared among all nodes. To handle heterogeneous networks, we follow the exact procedure used in the previous subsection to adapt our local hotspot detection and decomposition schemes to heterogeneous networks composed of a mix of microservers and motes. Our underlying assumption is that microservers are the only nodes responsible for storage and should be able to communicate among each other using the 802.11 radio rather than using the 802.5 one, thus, without involving motes. Thus, event relocations among microservers will not involve any motes. As the microservers' storage capacities are much higher than those of motes, this assumption seems mandatory for using the data migration concept in heterogeneous networks. Otherwise, applying the KDTR will considerably affect the QoS of the network as it will add a tremendous relative energy consumption burden on motes. We describe how KDDCS will work.

At the beginning of the network operation, each microserver needs to know its direct neighbors. Furthermore, the storage responsibility of the attribute range should be partitioned among microservers only. Thus, the KDDCS logical address assignment is only applied among microservers. At the end of this process, each microserver has a bit-code address representing its zone responsibility. To route events from motes to the corresponding sensors, each mote needs to be first associated with one of the zones. Thus, at the end

of the address assignment phase, each microserver initiates a broadcast message notifying surrounding motes with its address. Each mote associates itself with the closest microserver to its geographical position. Once a mote associates itself with a specific microserver, it inherits the full tree path of this microserver and stores all the pertinent information stored by this microserver. To hash and route any event, the mote uses its pertinent information as well as the attribute values of the event to determine the correct direction to which the event should be routed exactly as in the original KDDCS scheme.

Periodically, the KDTR algorithm is applied among microservers. In case a tree rebalancing takes place, each of the microservers of the rebalanced tree broadcasts its changed set of bisectors. Whenever a mote associated to one of these microservers listens to the broadcast message of this microserver, it updates its pertinent information with the new values. Hence, the pertinent information of all motes are kept up-to-date with those of the microservers at the end of the KDTR algorithm. This ensures the correct hashing and routing of events to their corresponding storage sensors.

In the future, we plan to experimentally study the performance of KDDCS on heterogeneous networks.

6.2.1.3 Experimenting on Sensor Networks Experiencing Failures Any sensor network can be affected by two major types of failure: node failures and packet failures. Node failures mainly occur due to lack of energy, while the main reason of packet failures are geographical obstacles or atmospheric variations.

Studying the Effect of Failures on ZS/ZP/ZPR: To understand the effect of failures on ZS/ZP/ZPR, we first discuss the techniques used by DIM to handle different failure types. As for packet failures, DIM deals with them by simply requiring ACKs for the different events (insertions and queries) and resending the packets corresponding to the missing ACKs when ACKs do not reach the sender after a deadline. To cope with node failures, DIM implements two types of replication. The first type is *local replication*, which consists of replicating each zone in the node responsible for the one's complement of its bit-code. This replication deals with random node failures. The second type is mirror replication. It consists of replicating each event in its backup node which is the node that takes the responsibility of the event

zone in case the current node responsible for the zone fails. This replication is for resilience to concurrent failures of geographically contiguous nodes. In general, replication increases the effect of a hotspot as it means that any storage and/or query hotspot arising in a given zone z will arise in more than one geographical locations depending on the replication level applied by DIM.

We now move on to the effect of failures on our local hotspot detection and decomposition schemes. As our schemes mainly focus on improving the QoS and QoD of DIM, they can be affected by node failures much more than being affected by packet failures. This is simply because the failure of a node causes the loss of its storage. In case this node already falls in a hotspot area, its failure wastes the benefits of the hotspot decomposition in case this node was involved in a ZS process, a ZP process, or a ZPR process. Additionally, the failure of this node decreases the number of available nodes for sharing the hotspot load, thus, increases the dangers of hotspots and reduces the ability of our schemes to decompose such hotspots. As for packet failures, the addition of our schemes on top of DIM does not affect the DIM's ACK resending process. Additionally, the existence of packet failures does not affect the performance of our schemes against hotspots compared the basic DIM scheme experiencing the same packet failures.

To cope with node failures, our schemes can take benefit of the DIM's replication schemes. Whenever a hotspot arises in a given zone z, DIM's replication causes the hotspot to arise in the node responsible for storing z as well as the nodes responsible for storing the replicas of z. Though this replication can cope with node failures, its implementation causes the bad effects of hotspots to be multiplied. Our schemes can deal with this increased negative effect of hotspots by applying the zone sharing/partitioning to the original storage node as well as to the replica nodes. In such a case, the SZLs and/or the TZLs are updated in the neighbors of the original node as well as in the neighbors of its replicas. Whenever a reading is inserted in a shared (or partitioned) zone, it goes to the new primary node storing the zone (the donor, the migrator, or the receiver, depending on the zone address). Whenever a query is issued, it can be answered by the closer node to the query issuer, either the new primary zone copy or the new replica nodes. Thus, applying our schemes to the replica nodes keeps the replication positive effect of successfully dealing with node failures while reducing the

multiplied hotspot burden caused by replication. In the future, we plan to experimentally study this node failure handling technique and its effect on the performance of our local schemes.

Studying KDDCS Resilience to Node Failures: Our KDDCS scheme implicitly assumed a failure-free network. This assumption is not completely realistic in the context of sensor networks as failures can occur due to different physical reasons, e.g. energy consumption. A failure of a node can cause the loss of its data which harms the QoD of the whole scheme. In this section, we present a simple replication scheme that makes KDDCS resilient to node failures. We describe our replication scheme below.

Our scheme is based on the idea of storing any reading in a node whose bit-code is a function of the original reading bit-code. As the bit-code of any reading in our scheme is dynamic and formed in a hop-by-hop manner based on the value of the bisectors in the intermediate tree nodes, the replica node cannot be obtained based on applying this function of the reading bit-code (as DIM for example applies a 1's complement on the static bit-code of each reading). Instead, our replication scheme uses the fact that the bounds of the attribute ranges are known to all nodes. Based on this piece of information, it is easy to get the complement of each reading corresponding to the reading with complement attributes to original attributes. Hence, the original reading is sent for storage in the node responsible for storing this complement reading.

Using this simple replication scheme, KDDCS can handle negative effects resulting from node failures. Once a query is issued to a failed node, the query can be redirected to the node(s) responsible for storing the failed node's attribute range. In the future, we plan to experimentally study this node failure handling technique and its effect on the performance of our KDDCS scheme.

6.2.2 Future Directions

We now move on to discuss some of the research directions that complement the work presented in this thesis and that we plan to work on in the near future. We also highlight the initial results we have for some of the studies we already pursued in these directions.

We believe that these studies, together with our thesis work, will have the great effect to pave the road towards building a robust in-network storage scheme that fits a variety of applications/workloads.

6.2.2.1 Load Balancing Query Hotspots Using Query Semantics In this thesis, we concentrated on decomposing query hotspots through data migration associated by local or global rebalancing of the DCS index structure. This can be considered as one way for solving the problem. Another technique for solving the problem can be through the usage of the query semantics of the different queries forming the hotspot.

In a preliminary work [6], we presented three *content-based* schemes to detect and decompose query hotspots in sensor networks. We assume the underlying sensor network implements a point-to-point routing scheme and is accessible by multiple base stations (either static or dynamic). A query is issued from a base station to a nearby sensor. This *query issuer* sensor is then responsible for forwarding the query to the *data source(s)*, the sensor(s) addressed by the query. Results are then forwarded to the issuer that, in its turn, answers the base station(s).

To decompose query hotspots, our schemes avoid duplication in forwarding results of similar queries. Our proposed solution is composed of two phases: *local hotspot detection*, and *hotspot decomposition*. The hotspot detection is solely determined by each data source sensor. At its high level, a sensor keeps track of the recent queries it answered (or currently answering), together with the issuer(s) of each of these queries. The sensor detects a possible hotspot when two (or more) queries intersect. Based on the nature of the intersection and the location of the issuers (physical or logical), the hotspot decomposition phase consists of one of three solutions: *two-phase query processing*, *three-phase query processing*, and *query partitioning*.

Our first load-balancing scheme, two-phase query processing, is based on detecting that two issuers are asking for the same data simultaneously or within a small time duration. The idea is to avoid duplication in sending results by answering one of the issuers and asking it to forward the results to the second issuer. The selection of the issuer to send results to depends on the approximate positions of the two issuers with respect to the data source. This scheme

is beneficial when the distance between the two issuers is smaller than the distance between each one of them and the data source.

When the two issuer nodes and the data source form an equilateral triangle, our three-phase query processing scheme sends query results to an intermediate destination which then forwards received results to both issuers.

Our third load-balancing scheme, namely query partitioning, considers load-balancing when a query intersects with previous queries answered by the same data source. In general, two or more queries simultaneously addressing a data source may intersect among each other in part of their results, and at the same time, may intersect with one or more queries that recently addressed the same data source. In such a case, the data source detects the two types of intersections. For the first type of inter-query intersection, the data source applies either the two-phase or the three-phase query processing technique depending on the locations of the query issuers (as described in the previous two sections). For the second intersection type, the data source redirects the intersecting part of the query to be answered by the most recent issuer of that part. This can be done through sending a single packet to this issuer, x. Then, x acts like a new data source receiving a new query and processing it using data cached in its memory. In case the query result is to be sent to more than one issuer of the original query, x applies either two-phase or three-phase query processing. Otherwise, the result is forwarded to the only issuer requesting it. Assuming that a query is usually composed of more than one packet, this scheme decreases the load on data sources and maximizes the benefit of caching query results.

We show through extensive simulations, that the major advantages achieved by applying our schemes on top of geographic routing protocols are:

- Load balancing query hotspots and thus increasing the network lifetime and throughput.
- Improving the QoD offered by the sensor network.
- Maintaining a comparable level of QoS and real-time guarantees to that offered by the underlying routing protocol.

In general, this work can be considered as one step towards the full usage of the semantics of queries forming the query load in order to load-balance the query access frequencies among

the different sensor nodes in the sensor network. In the future, we're interested in developing schemes which take benefit of both, data migration and query semantics, in order to avoid the formation of query hotspots in DCS sensor networks.

6.2.2.2 Spatio-Temporal Data-Centric Storage for Real-Time Sensor Applications In our dissertation, we mainly focused on sensor networks whose query load is solely composed of range queries. However, geo-centric queries represent another major type of queries that will be heavily accessing sensor networks in the near future. It is widely believed that sensor networks will shortly consist of globally deployed sensors providing real-time geo-centric information to users. Particularly, users with mobile devices will issue ad-hoc queries usually from within, or nearby, the queried area. An example of sensor clusters is the *Bronx Zoo* cluster. In this application, motes are deployed in the Bronx Zoo for habitat monitoring [14,62,60,15]. The park visitors are allowed to use their mobile devices to query sensors for real-time information about animals, their behaviors, the climate they live in, etc.

It is clear that the in-network data-storage for this application has different requirements than those of our disaster management application due to the different nature of queries and their *geographic locality* that was not a characteristic of our ad-hoc queries. Load balancing continues to be an important goal to shoot for while building an in-network data-storage scheme for this application. Recall that, in a typical DCS scheme, a *readings-to-sensors* mapping function assigns the responsibility for storing the reading of any sensor to a storage-sensor based on the value of that reading. As our queries are geo-centric, we realize that reading-based indexing is not a good fit for our model as processing a geo-centric query will require flooding the whole cluster. In a recent work [7], we present the Spatio-Temporal Data-Centric Storage scheme (STDCS), a novel DCS scheme for real-time geo-centric sensor network applications. In STDCS, data indexing is based on the sensor locations rather than the reading values. Hence, STDCS presents a *sensors-to-sensors* mapping instead of the previous readings-to-sensors mappings. Our scheme embeds the sensor geographic locations into the leaves of a k-d tree [11] and assigns virtual addresses to sensors based on their positions in the k-d tree. The virtual address of each sensor is used as an input to the mapping function as to determine its storage-sensor. Any point-to-point routing scheme can

be then used to route readings to their storage-sensors, e.g. the Greedy Perimeter Stateless Routing protocol (GPSR) [32]. Query processing can be easily done locally and distributively by the sensors in a hop-by-hop manner.

Our major design goal for STDCS is *load-balancing*. Traffic skewness may easily occur in our applications due to the time-varying number of users and the hard task of expecting their behaviors at any point in time. The main skewness source in our model lies in query hotspots that may arise because of the difference in *popularity* between the readings of different sensor nodes due to the reading type, location, time, etc. To maintain load balancing, we present the novel concept of spatio-temporal data indexing, where the mapping of readings to their storage-sensors depends not only on the location of the generating sensor, but also on the generation time of the reading. Spatio-temporal indexing balances the load, in terms of query accesses, among sensors with no dependence on the query distribution imposed on the cluster. Additionally, we present a separate load balancing scheme to *adaptively* detect and decompose query hotspots. This scheme is based on the dynamic adjustment of the parameters of the spatio-temporal data indexing.

Through extensive experimental evaluation, we show that the advantages of STDCS are:

- Highly outperforming both local storage and spatial indexing when facing query hotspots.
- Minimizing load balancing overhead by adaptively adjusting the needed load balancing level based on the detected skewness level of the hotspot.

In summary, STDCS is different from previous proposals in two aspects. First, it is based on the novel idea of using a temporally evolving spatial indexing scheme to balance querying load among sensors. Furthermore, STDCS uses dynamic mechanisms for query hotspot detection and decomposition.

An important question to be asked is whether it is possible to build a unified in-network data-storage scheme that can efficiently process and answer both, range queries and geo-centric queries. Another important question is how to maintain load balancing for such a scheme (with its orthogonal purposes). We believe that building such a scheme will be considered as a huge step that would quickly put the concept of in-network data-storage into action in real-world sensor network deployments.

BIBLIOGRAPHY

[1] Crossbow mica2, 2003. http://www.xbow.com.

[2] *Monte Carlo Statistical Methods*. Springer-Verlag, 2004.

[3] Michele Albano, Stefano Chessa, Francesco Nidito, and Susanna Pelagatti. Data centric storage in non-uniform sensor networks. In *Proc. of INGRID*, 2007.

[4] Michele Albano, Stefano Chessa, Francesco Nidito, and Susanna Pelagatti. Q-NiGHT: Adding qos to data centric storage in non-uniform sensor networks. In *Proc. of MDM*, 2007.

[5] Mohamed Aly, Panos K. Chrysanthis, and Kirk Pruhs. Decomposing data-centric storage query hot-spots in sensor networks. In *Proc. of the* 3^{rd} *Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS)*, 2006.

[6] Mohamed Aly, Anandha Gopalan, and Adel Youssef. Load-balancing query hotspots for next-generation sensornets. In *Proc. of the* 50^{th} *IEEE Global Communications conference (GLOBECOM)*, 2007.

[7] Mohamed Aly, Anandha Gopalan, Jerry Zhao, and Adel Youssef. STDCS: Spatio-temporal data-centric storage for real-time sensor applications. In *Proc. of the* 5^{th} *IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2008.

[8] Mohamed Aly, Nicholas Morsillo, Panos K. Chrysanthis, and Kirk Pruhs. Zone Sharing: A hot-spots decomposition scheme for data-centric storage in sensor networks. In *Proc. of he* 2^{nd} *International VLDB Workshop on Data Management for Sensor Networks (DMSN)*, 2005.

[9] Mohamed Aly, Kirk Pruhs, and Panos K. Chrysanthis. KDDCS: A load-balanced in-network data-centric storage scheme in sensor network. In *Proc. of the* 15^{th} *ACM Conference on Information and Knowledge Management (CIKM)*, 2006.

[10] Seung Jun Baek and Gustavo de Veciana. A scalable model for energy load balancing in large-scale sensor networks. In *Proc. of 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2006.

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

[12] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. In *Proc. of MDM*, 2001.

[13] Qing Cao and Tarek Abdelzaher. A scalable logical coordinates framework for routing in wireless sensor networks. In *Proc. of RTSS*, 2004.

[14] UC Davis Wildlife Health Center. Southern california puma project, 2004. http://www.vetmed.ucdavis.edu/whc/scp/.

[15] Panos K. Chrysanthis and Alexandros Labrinidis. Final workshop report of the NSF workshop on data management for mobile sensor networks (MobiSensors), 2007. http://mobisensors.cs.pitt.edu/.

[16] Yanlei Diao, Deepak Ganesan, Gaurav Mathur, and Prashant Shenoy. Rethinking data management for storage centric sensor networks. In *Proc. of CIDR*, 2007.

[17] Stefan Dulman, Tim Nieberg, Jian Wu, and Paul Havinga. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. In *Proc. of WCNC*, 2003.

[18] Cheng Tien Ee, Sylvia Ratnasamy, and Scott Shenker. Practical data-centric storage. In *Proc. of NSDI*, 2006.

[19] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. 2004.

[20] Deborah Estrin. Reliability and storage in sensor networks. Technical report, 2005.

[21] Rodrigo Fonseca, Sylvia Ratnasamy, Jerry Zhao, Cheng Tien Ee, David Culler, Scott Shenker, and Ion Stoica. Beacon Vector Routing: Scalable point-to-point routing in wireless sensornets. In *Proc. of NSDI*, 2005.

[22] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5, 2001.

[23] Deepak Ganesan, Sylvia Ratnasamy, Hanbiao Wang, and Deborah Estrin. Coping with irregular spatio-temporal sampling in sensor networks. In *Proc. of HotNets-II*, 2003.

[24] Familiar Group. The Familiar project, 2003. http://familiar.handhelds.org/.

[25] Longjiang Guo, Yingshu Li, and Jianzhong Li. Event query processing based on data-centric storage in wireless sensor networks. In *Proc. of IEEE Globecom*, 2006.

[26] Jason Hill and David Culler. Mica: A wireless platform for deeply embedded networks. *Proc. of IEEE Micro*, 22(6):12–24, 2002.

[27] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proc. of ASPLOS-IX*, 2000.

[28] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking (TON)*, 11, February 2003.

[29] Intel. Intel Stargate, 2002. http://platformx.sourceforge.net.

[30] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. 2005.

[31] Brad Karp. Challenges in geographic routing: Sparse networks, obstacles, and traffic provisioning. In *Proc. of the DIMACS Workshop on Pervasive Networking*, 2001.

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

[33] Young-Jin Kim, Ramesh Govidan, Brad Karp, and Scott Shenker. On the pitfalls of geographic face routing. In *Proc. of DIALM-POMC*, 2005.

[34] Yongxuan Lai, Yufeng Wang, and Hong Chen. Energy-efficient robust data-centric storage in wireless sensor networks. In *Proc. of DMSN*, 2007.

[35] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *Proc. of Sensys*, 2003.

[36] Xin Li, Fang Bian, Ramesh Govidan, and Wei Hong. Rebalancing distributed data storage in sensor networks. Technical report, University of Southern California, 2005.

[37] Xin Li, Young Jin Kim, Ramesh Govidan, and Wei Hong. Multi-dimensional range queries in sensor networks. In *Proc. of ACM SenSys*, 2003.

[38] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. volume 36, pages 131–146, New York, NY, USA, 2002. ACM Press.

[39] the Electrical Engineering Maxwell Dworkin Laboratory and Harvard University Computer Science Department. MoteLab, 2005. http://motelab.eecs.harvard.edu/index.php.

[40] Katina Michael and Luke McCathie. The pros and cons of RFID in supply chain management. In *Proc. of the International Conference on Mobile Business (ICMB)*, 2005.

[41] Dust Networks. Blue Mote, 2003. http://www.dust-inc.com.

[42] James Newsome and Dawn Song. GEM: Graph embedding for routing and data centric storage in sensor networks without geographic information. In *Proc. of SenSys*, 2003.

[43] Seung-Jong Park, Ramanuja Vedantham, Raghupathy Sivakumar, and Ian F. Akyildiz. A scalable approach for reliable downstream data delivery in wireless sensor networks. In *Proc. of MobiHoc*, 2004.

[44] Tri Pham, Eun Jik Kim, and W. Melody Moh. On data aggregation quality and energy efficiency of wireless sensor network protocols. In *Proc. of BROADNETS*, 2004.

[45] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: enabling ultra-low power wireless research. In *Proc. of IPSN*, 2005.

[46] Ioan Raicu, Loren Schwiebert, Scott Fowler, and Sandeep K.S. Gupta. Local load balancing for globally efficient routing in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 1, 2005.

[47] Ananth Rao, Sylvia Ratnasamy, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *Proc. of Mobicom*, 2003.

[48] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govidan, and Scott Shenker. GHT: A geographic hash table for data-centric storage. In *Proc. of WSNA*, 2002.

[49] Tobias Scholl, Bernhard Bauer, Benjamin Gufler, Richard Kuntschke, Angelika Reiser, and Alfons Kemper. Scalable community-driven data sharing in e-science grids. *Future Gener. Comput. Syst.*, 25(3):290–300, 2009.

[50] Rahul C. Shah and Jan M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, 2002.

[51] Min Shao, Sencun Zhu, Wensheng Zhang, and Guohong Cao. pDCS: Security and privacy support for data-centric sensor networks. In *Proc. of Infocom*, 2007.

[52] Mohamed A. Sharaf, Jonathan Beaver, Alexandros Labrinidis, and Panos K. Chrysanthis. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *Proc. of MobiDE*, 2003.

[53] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govidan, and Deborah Estrin. Data-centric storage in sensornets. In *Proc. of HotNets-I*, 2002.

[54] Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *Commun. ACM*, 47(6):34–40, 2004.

[55] Infineon Technologies. EyesIFXv1 node, 2005. http://www.infineon.org.

[56] Ping Xia, Panos K. Chrysanthis, and Alexandros Labrinidis. Similarity-aware query processing in sensor networks. In *Proc. of WPDRTS*, 2006.

[57] Ting Yan, Tian He, and John A. Stankovic. Differentiated surveillance for sensor networks. In *Proc. of the 1st international conference on Embedded networked sensor systems (SenSys)*, pages 51–62, New York, NY, USA, 2003. ACM.

[58] Ting Yan, Tian He, and John A. Stankovic. Differentiated surveillance for sensor networks. In *Proc. of SenSys*, 2003.

[59] Yong Yao and Johannes Gehrke. Query processing for sensor networks. In *Proc. of CIDR 2003*, 2003.

[60] Demetrios Zeinalipour-Yazti, Panayiotis Andreou, Panos K. Chrysanthis, and George Samaras. SenseSwarm: a perimeter-based data acquisition framework for mobile sensor networks. In *Proc. of DMSN*, 2007.

[61] Demetrios Zeinalipour-Yazti, Song Lin, Vana Kalogeraki, Dimitrios Gunopulos, and Walid A. Najjar. Microhash: An efficient index structure for flash-based sensor devices. In *Proc. of USENIX FAST*, 2005.

[62] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. Hardware design experiences in ZebraNet. In *Proc. of SenSys*, 2004.