

Distributed Information Storage and Collection for WSNs

Christine Jardak[†], Evgeny Osipov[‡], Petri Mähönen[†]

[†]Department of Wireless Networks

RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
{cja, pma}@mobnets.rwth-aachen.de

[‡]Department of Computer Science and
Electrical Engineering

LTU, Luleå University of Technology
Campus Porsön, S-971 87 Luleå, Sweden
eao@ltu.se

Abstract—Distributed data storage is an important component of wireless sensor networks, which protects the mission critical information from unexpected node failures or malicious destruction of parts of the network. In this paper we present DISC, a protocol for distributed information storage and collection. The two major mechanisms in DISC which make our solution distinct from the related approaches are probabilistic choice of storing nodes and a search engine based on the usage of Bloom filters. In comparison to the deterministic choice of the backup node, the random selection strategy makes it virtually impossible for an attacker to determine and destroy the exact node keeping a particular piece of information. The usage of Bloom filters in the information search engine makes the navigation to a specific data fast and efficient. We show that with DISC the amount of recovered information is more than two times higher than that in deterministic storage schemes.

I. INTRODUCTION

Distributed data storage is an essential component of many wireless sensor networks (WSNs). By temporarily keeping the data inside a network, the information in this kind of WSN will be more resilient against malicious or catastrophic physical damage of parts of the network. Assuming a mission critical application of a WSN, the owner of the network wants not only to receive the measured data from a particular area of interest in real time, but wants also to retrieve, at least partially, the information measured in one or more physically destroyed areas. A natural solution to achieve this goal is to backup the data in geographically distant nodes.

In this article, we consider a question of choosing a node for logging the critical sensor data. In most of the current systems for distributed data storage, the backup node is selected deterministically. In the simplest case the archiving functionality is assigned to a cluster head node as in [1]. In TinyPEDS [2] the information is also stored in a cluster head node of a neighboring cluster in order to increase the network robustness. In the later scheme the target backup cluster is chosen deterministically based on strictly structured cluster identifiers.

An obvious advantage of the deterministic selection of the backup node is the simplicity of implementation. Normally,

the protocol for retrieving the data from the network is simple due to the predefined position of the backup nodes. However, the determinism of location of the backup and the source nodes makes the sensor network vulnerable to malicious destruction of both. Potentially, this leads to a total loss of sensor data in WSN applications where the presence of historical information is essential to reconstruct the scene and take appropriate actions. Examples of such mission critical applications are WSNs for monitoring and controlling the situation on highways and in tunnels [3]; WSNs for detecting weapons of mass destruction in homeland security scenarios [4]; and WSNs for surveillance missions used by police and military, e.g [5] [6].

In this article we present DISC, a Distributed Information Storage and Collection protocol for wireless sensor networks which addresses above threats. With DISC, the backup node is selected randomly. We use Bloom filters to represent the stored information inside the network. This technique makes the procedure of searching and collecting the data fast and efficient. Although Bloom filters have been previously used in the WSN context, see e.g. [7] and [8] an original application of this technique in DISC context makes our protocol a unique contribution.

The remainder of this paper is organized as follows. We describe the protocol design and outline the major principles of our approach in Section II. After that we discuss the existing work for distributed data storage in Section III. Section IV is the main section where we describe the concept of Bloom filters and present the protocol architecture. We demonstrate illustrative performance of DISC in Section V. We conclude the discussion with Section VI.

II. PROTOCOL DESIGN SPACE

Before proceeding further with the outline of the DISC architecture we introduce the considered network model and several assumptions essential for the protocol design.

A. Network model and assumptions

A wireless sensor network (WSN) illustrated in Figure 1 is formed by N sensor nodes arbitrarily located in a monitored

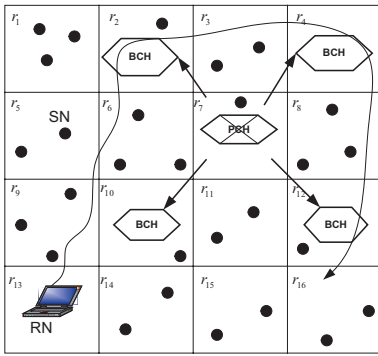


Fig. 1. The geographical query targeting the BCH nodes of the destroyed region X.

area. The area is divided into regions and we assume that each region is assigned an identifier denoted as r_i in the figure. Note that we do not place any specific assumptions on the structure of the regions' IDs. These identifiers can be assigned either manually or by means of any existing ID configuration protocol. Each sensor node has the following pieces of location information:

- Its own position in a local or global coordinate system.
- The identifier of a region where the node is located.
- The identifier of the neighboring regions.

We assume a hierarchical architecture of the sensor network. The network is logically divided into clusters, where in each cluster at least one node is elected as a cluster head. In order to achieve this, sensor nodes execute a low-energy cluster formation algorithm such as PANEL [9]. Without loss of generality we assume that a logical cluster of nodes coincides with the respective geographical region where it is formed. Further on we will use the terms *cluster* and *region* interchangeably. Consider an arbitrary chosen cluster (in Figure 1 the cluster with $ID = r_7$). We will refer the cluster head node in this region as to the *primary cluster head* (PCH). Correspondingly we will refer the cluster head nodes in the neighboring regions as to *backup cluster heads* (BCHs).

Apart from the sensor nodes in our network model we specify a *reader node* denoted RN in Figure 1. This is a user operated device, which collects and processes the measured data from the sensor network. We assume that the RN has virtually unlimited memory and energy resources in comparison to the sensor nodes. We do not specify further requirements for the reader device. It can be either mobile, carried by a human or vehicle or static, stationary mounted at a specific location.

The RN allows a user to form location oriented data queries using a high level SQL-like language such as in [10]. Since specification of such language is beyond the scope for this paper we do not further discuss this issue. We also assume that the user's query converted to a binary format is propagated towards a particular geographical region using a geographical routing protocol such as DREAM [11] or GPSR [12].

Finally, we assume that the life time of the WSN is divided into epochs of duration τ . The time origin, the start of each epoch and the consequent epoch enumeration (e_1, e_2, e_3, \dots) is network-wide synchronized by means of existing time synchronization protocols as shown in [13]. This allows a user to uniquely address a particular period of time by specifying the sequence number of a particular epoch (e_i).

B. Solution outline

In a nutshell the DISC protocol consists of two major functional parts: The data dissemination and the data collection. The former part is executed at the PCH and BCH nodes. The later is executed in all nodes that were BCH at some point in time as well as the reader node.

The logic of the data dissemination part of DISC is straightforward. Sensor nodes periodically report the measured data to the PCH, which in turn after a possible aggregation sends the data to one of the neighboring BCHs¹. The distinctive property of DISC which makes it different from other approaches is a random selection of the backup cluster head.

After computing the aggregated data at the PCH, we create a unique descriptor which includes the time epoch of the data aggregation, the region identifier of the aggregator and the type of data. In DISC the descriptor that we call a *trace* is a Bloom filter with one element obtained from the above stated parameters. The PCH then sends one or more data items along with the corresponding traces to a randomly chosen BCH. For the memory balancing purpose and in order to increase the randomness of the data location, BCH nodes do not store the received data locally but distribute it among the sensor nodes in their clusters. The role of the BCH is to keep the routing state what we call a *trace-table*, which store an identifier of the node keeping a particular piece of information.

The data collection part of DISC is activated when there is a need to backtrack the information from a physically destroyed region within a particular period of time. Firstly, the reader creates a high level query specifying the identifier of the region of interest, the epoch when the information is created and the data type. A *trace creation engine* of DISC takes this query as an input string for several hash functions and creates a set of binary traces as an output. For efficient communication, groups of traces are combined into a set of Bloom filters with several elements each forming the request message. The geographic routing protocol then carries the request to former and current BCH nodes of the region of interest. At BCHs the entries of the *trace-table* are tested for a membership in the received Bloom filter. When a trace entry matches the Bloom filter the node that keeps the actual data is commanded to send it to the reader device.

III. RELATED WORK

The current approaches for storing and retrieving data in WSN follow either a *local storage* model, where the measured

¹Note that the information from sensor nodes to the cluster head and between the cluster heads could be transmitted either in a plain text or in an encrypted form as it is done in [2].

data is stored locally on each sensor node or a *remote storage* model where the data is stored on nodes different from the one that produced it. In local storage systems each sensor generates and stores its own readings locally. A distributed data base TinyDB [10] and a publish-subscribe Directed Diffusion architecture [14] are the major representatives for the first class of the systems. While being very easy and intuitive to use for an end user these systems are vulnerable to malicious destruction of critical information.

In approaches following the remote storage model the sensor readings generated by a particular node are stored on distant nodes. In Data-Centric Storage (DCS) systems such as [15] and [16] some sensor nodes in different regions of the network are allocated to store data of a specific type. A sensor node generating data of a particular type (i.e. temperature) hashes it to the network region responsible for storing this type of information. The knowledge of the location for the particular type of information is known in all WSN nodes therefore the routing mechanisms for queries are simple. However, the behavior of the DCS routing is not sufficiently studied when the hashing criterion is the time of taking the measurements. In this case there is no determinism in selection of the backup node and the complexity of the routing increases.

An approach that uses a nondeterministic distributed storage model is Hood [17], a neighborhood programming abstraction. In Hood each sensor node can identify a subset of nodes within its radio coverage range and use them to share its state information. The attributes describing the state and their values are broadcasted and the receiving nodes after filtering cache them locally. Hood provides a good abstraction for a variety of distributed sensor applications, however, its application to the distributed data storage is not clear. The data sharing is limited to the scope of the local neighborhood, which potentially makes the storage vulnerable to large scale network destructions. The scalability of Hood in the case of more than one radio-hop neighborhoods is also not stringent. While in principle the neighborhood abstraction is not limited to the direct neighbors, the usage of broadcast to distribute the node's state potentially makes the application of Hood in multihop wireless environment inefficient.

One approach which in the first place motivated the development of DISC is TinyPEDS [2], a hierarchical secure distributed data storage solution. The measured information is transmitted from sensor nodes to the cluster head in an encrypted form. The cluster head then stores the data aggregated during a predefined period of time in a cluster head of the neighboring cluster. The routing mechanism to a cluster head for storing the information and from a sink node to this region is deterministic in TinyPEDS. It is based on strictly structured identifiers for geographical regions: The monitored area is logically divided into regions; All regions are assigned a unique incremental integer identifier; The data from region with $ID = i$ is stored in region with $ID = i + 1$. With DISC we intend to break this determinism of choosing the storage node additionally protecting the data from malicious destruction of the backup regions.

IV. DISC PROTOCOL ARCHITECTURE

Before proceeding further with the description of the DISC functionality we introduce the Bloom filters technique, which forms the core of the data description and the routing mechanism in our protocol.

A. Bloom filters

Bloom filter is a method for representing a set of n elements $A = \{a_1, \dots, a_n\}$ which allows a fast test on a membership of an arbitrary element a' inside A . Each element in the original set A is mapped into a set of m -bit vectors V , further denoted as *hash vectors* or *h-vectors*, of the same cardinality by using the following procedure. Consider an arbitrary element $a_i \in A$. In the corresponding bit vector v_i we initialize all bit positions with zeros. Let us choose k different hash functions which maps an element from A into a number $p_j = h_j(a_i), p_j \in (0, m-1), j = 1..k, i = 1..n$. In h-vector v_i we set positions p_j to 1. Bloom filter b representing set A is a linear combination of vectors $v_i, \forall i$.

In order to test the membership of an element a' inside A we first construct a hash-vector v' from a' , $v' = f(h_i(a'))$, $i = 1..k$, by the procedure described above and do a bitwise AND with the Bloom filter b . If $v' \& b \neq v'$ then a' certainly does not belong to A . In the case of equality a' is considered as a member of A with a probability of *false positive* computed as

$$P_f = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \cong \left(1 - e^{-\frac{kn}{m}}\right)^k. \quad (1)$$

Note that equation (1) above also regulates the choice of parameters k, m and n in order to upperbound the probability of the false positive. Given the ratio $\frac{m}{n}$, the optimal choice of k is given in [18] as $k = \lceil \frac{m}{n} \ln(2) \rceil$. For example, given $\frac{m}{n} = 4$, the optimal value $k = 3$. In DISC we use $k = 3, m = 16$ and $n = 3$ as a tradeoff between fast computation of h-vector computation on sensor nodes and sufficiently low probability of the false positive, which is with parameters above equals 9.2%.

The most notable application of Bloom filters in the Internet is their usage in peer-to-peer networks to probabilistically locate replicas. In [19], [20], [21] and [22] the authors propose that each peer stores two types of filters, a local filter (LF) computed from attributes of local objects and remote filters (RF) for attributes of objects stored in remote peers. When a peer receives a query for a particular set of attributes in the form of a bit-vector, it searches its LF then the RF filters in order to decide whether the requested information is stored locally or the request should be further routed.

There are several application of Bloom filters in wireless sensor networks. The most representative approaches are Mobeyes [7] and hierarchical clustering scheme [8]. The authors of Mobeyes suggest a system for monitoring road environment. In their solution vehicles equipped with wireless sensors monitor the condition and situation on the road. The sensed data is stored locally and periodically transmitted to neighboring vehicles using summaries. The summaries

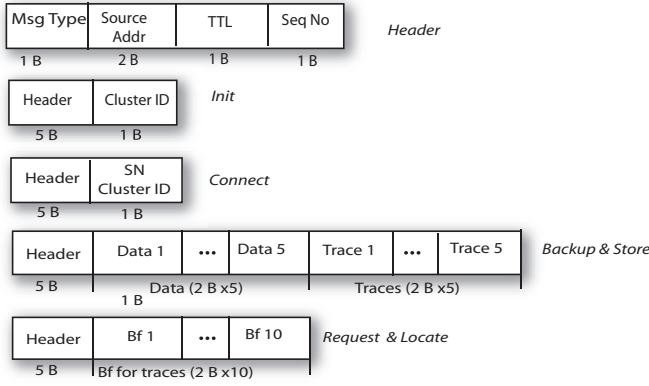


Fig. 2. Format of DISC messages.

are in essence hash-vectors described above computed from the timestamp of taking the measurements and the position coordinates. A mobile agent (a police car) interested in reconstructing the situation at a particular moment in time opportunistically gathers summaries from the vehicles. It uses Bloom filters to represent its set of already collected and still valid summary packets, forms request messages and broadcasts them. Upon receiving the request each vehicle performs the membership test of its stored summaries to the received Bloom filter and sends a list of items missing at the mobile agent.

The authors of [8] propose a hierarchical clustering which facilitates data aggregation and data discovery. The protocol organizes the network in a tree logical topology. Each node maintains Bloom filters which represent its own information and the information that stored in all its descendants. When the root node wants to determine which sensor has generated certain data it creates a h-vector computed from the data description. The query is navigated down the tree to the target node by performing the membership test of the h-vector inside Bloom filters of intermediate nodes. The query is transmitted only by those nodes where the result of the membership test was positive.

In comparison to the above applications of Bloom filters, in DISC we use this technique both to locate an atomic piece of information and for more complex *range* requests, where we search for a set of data satisfying certain condition.

B. DISC protocol description

DISC uses four types of messages to initialize the system, form the primary backup cluster, discover the backup aggregators, and disseminate the backup information. It also uses two types of messages for querying the BCHs to locate the archived data. The format of DISC messages is shown in Figure 2.

- *Init* (6 bytes): This message is used to announce the presence of newly elected cluster head node within the cluster.
- *Connect* (6 bytes): This message is used by sensor nodes in a cluster to connect to the primary cluster head.

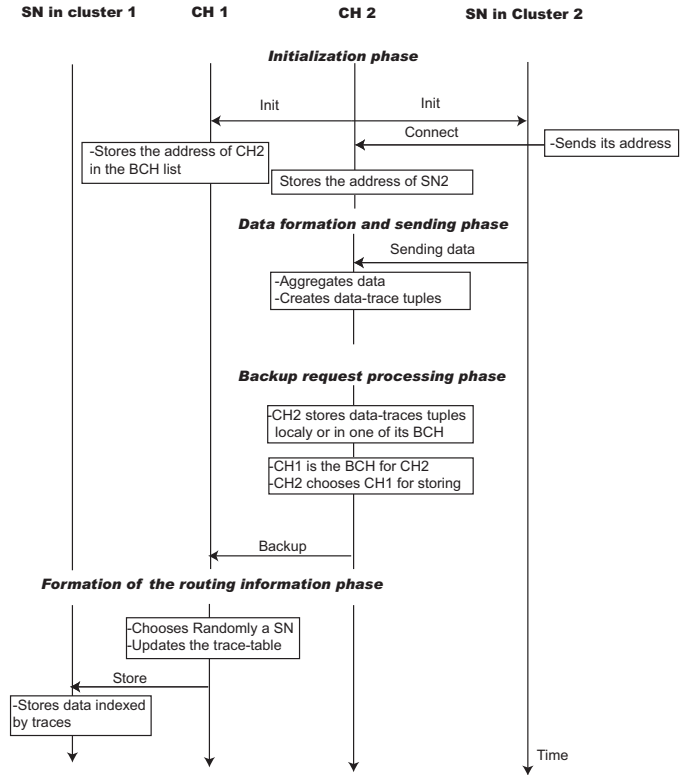


Fig. 3. Data dissemination part of DISC.

- *Backup* (25 bytes): This message is used to request the chosen BCH node to backup the data².
- *Store* (25 bytes): This message is used to request sensor nodes in the chosen backup cluster to store the data from the primary cluster.
- *Request* (minimum 7 bytes, maximum 25 bytes): This message is issued by the reader device to query BCH nodes for stored information.
- *Locate* (minimum 7 bytes, maximum 25 bytes): This message is sent from the BCH to the sensor node to retrieve the requested data.

The DISC protocol consists of two major functional parts: the data dissemination and the data collection. The former part is executed at PCH and BCH nodes. The latter is executed in all nodes that were BCH at some point in time as well as the reader node. Below we describe the details of each part.

1) *Data dissemination part*: The data dissemination part of DISC is shown on the message flow chart in Figure 3. It consists of four phases: The initialization, the data formation and sending, the backup request processing in the primary cluster and the formation of the routing information in the backup cluster head.

a) *Initialization phase*: During the initialization phase the sensor nodes learn about the corresponding primary cluster

²The resolution of the ADC (Analog to Digital Converter) in Crossbow TelosB motes is 12 bits, therefore we allocate 2 bytes for the actual data in the packet.

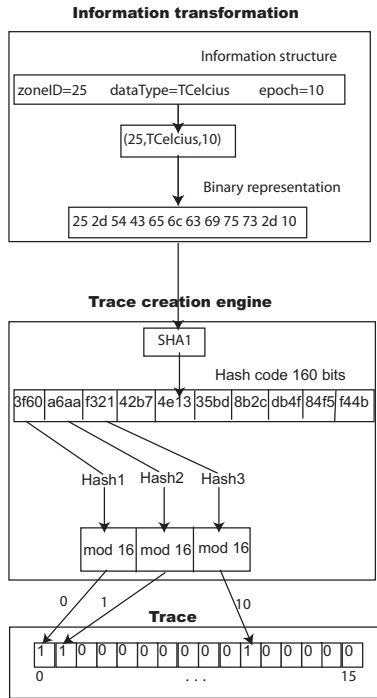


Fig. 4. The trace formation procedure.

head node and cluster head nodes learn about the presence of each other. For this purpose the newly elected CH node broadcasts the *Init* message containing (a) its address; (b) the incrementally increasing sequence number; (c) TTL, the time to live; and (d) the identifier of the cluster where the node is elected. Note that in general cluster heads in the neighboring clusters can be located several hops away from each other. In this case the *Init* message should be re-broadcasted by intermediate nodes in a given range specified by the TTL field.

The *Init* message can be received by (a) sensor nodes in the own cluster; (b) sensor nodes in the neighboring clusters; and (c) the CH nodes in the neighboring clusters. Upon reception of the message these nodes perform different actions. The sensor nodes in the primary cluster reply with the *Connect* message, notifying the PCH about their addresses. The sensor nodes in the neighboring clusters re-broadcast this message until it reaches the cluster head or TTL expires. The CH nodes in the neighboring clusters store the address and the cluster identity information in a list of backup cluster heads. Duplicates of *Init* messages with the same sequence number are discarded by all nodes.

b) Data formation and sending phase: The data formation and sending phase is executed at primary cluster head nodes. During each epoch sensor nodes send the measured information to the primary cluster head. The information can be handled by PCH nodes in two ways. The primary cluster head may backup either raw unprocessed data or perform an additional processing and aggregate data before sending it to the backup cluster. In both cases the PCH node computes a meta-data corresponding to a particular chunk of the backup

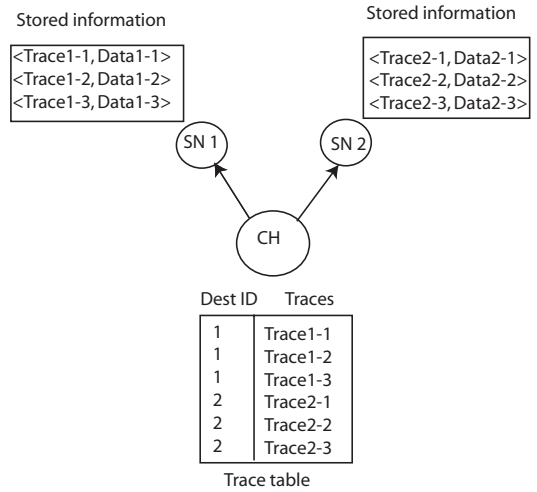


Fig. 5. The trace table of a cluster head having two SNs.

information. The meta-data that we call *trace* is used to index the backup information for the purpose of further retrieval during the data collection phase. Figure 4 shows an example of the trace creation procedure. There the temperature in Celsius is measured in cluster with ID equals 25 and aggregated in epoch 10. The attribute tuple $\langle zoneID, dataType, epoch \rangle$ is converted into a binary form and fed into the trace creation engine. The trace creation engine executes three steps. Firstly, it computes hash of the attributes using SHA1 hash function. As stated in Section IV-A in order to obtain a hash vector several independent hash functions shall be computed. As we show in the next section execution of SHA1 function on sensor nodes is computationally expensive. In order to minimize the processing time during computation of traces we use the following solution. Due to the pseudo-randomness of the SHA1 output it is a common approach to consider subparts of the output as results of several independent hash functions. Therefore, as a second step the resulting 160 bits hash is divided into ten chunks of two bytes each which we consider as outputs of ten hash functions. Finally, we take first three two bytes chunks and compute a hash-vector corresponding to the initial attributes as described in Section IV-A. The result of the data formation phase is a tuple $\langle data, trace \rangle$ which the PCH node either stores locally or sends to one of the backup cluster heads as described below.

c) Backup request processing phase: The backup request processing phase is executed at PCH nodes in the case when it probabilistically decides that the information should be stored on a remote node. Firstly, the PCH randomly chooses a backup cluster head from its BCH list. Secondly, it sends a *Backup* message containing the data-trace tuples obtained during the previous phase.

d) Formation of the routing information phase: This phase is executed at BCH nodes after receiving a *Backup* message. In the same fashion as PCHs, the BCH may store

TABLE I
MEMORY FOOTPRINT AND RUNTIME PERFORMANCE PARAMETERS.

Parameter	Value
Code size in ROM	3700 B
Variables and structures in RAM	800 B
Size of an entry in trace-table	2 B
Size of a storage entry in a sensor node	4 B
Code size for SHA1 in ROM	1800 B
Code size for RS-Hash in ROM	162 B
Trace generation time using SHA1	11.23 ms
Trace generation time using RS-Hash	0.64 ms

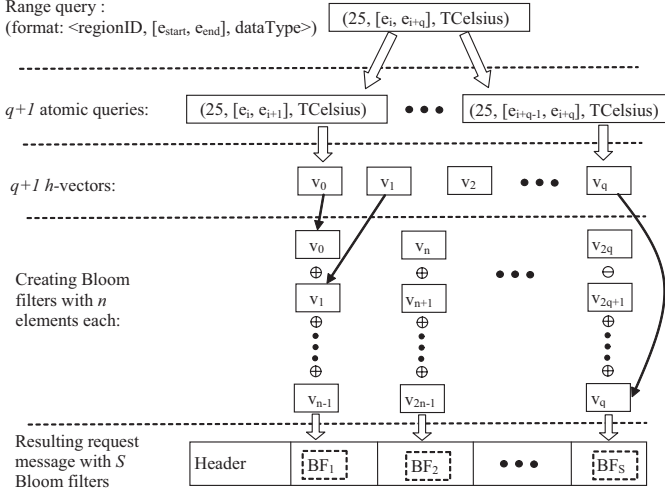


Fig. 6. Creating a Request message.

data either locally or on one of the sensor nodes inside its cluster. If BCH decides that the information should be stored on a remote node, it randomly selects a sensor node from its cluster and sends a *Store* message containing the data to be archived along with the corresponding traces. The address of the sensor node that stores data with a specific trace is inserted into a *trace-table*, which structure is shown in Figure 5. In order to conserve the memory of sensor nodes, we make the routing state for a specific data item in BCH soft, expiring it after time T which is an adjustable parameter of DISC. The same is valid for sensor nodes that keep the actual data.

2) *Data collection part*: The data collection part is divided into two phases: The formation of queries and the location of the stored data.

a) *Formation of queries*: The query formation phase is executed in the reader device when a user decides to locate data of a certain type taken in a specific region during particular time interval. The user specifies the query as $\langle regionID, [e_{start}, e_{end}], dataType \rangle$ tuple. There $regionID$ is an identifier of the target region, $[e_{start}, e_{end}]$ is the time interval when the data of interest was created and $dataType$ is the type of requested information. According to our assumption the user specifies the time interval in terms of *epoch*'s sequence number. The procedure of forming a *Request* message with a range query for the temperature information in region 25 during $q+1$ epochs $[e_i, e_{i+q}]$ is shown in Figure 6. Firstly, the initial query is splitted into $q+1$ atomic queries, each corresponding to the data stored during a particular epoch. Secondly, $q+1$ hash vectors are computed as described in Section IV-A for each atomic query. Then, the resulting sequence of h-vectors is split into S groups of n h-vectors each. Finally, by doing bitwise *or* operation within each group we compute S Bloom filters which we record in the *Request* message. The dimensioning of parameter n is described in

Section IV-A. In the current implementation we limit the number of Bloom filters per request message by 10 entries which makes the size of the *Request* message payload bounded between two and 25 bytes. We conjecture that this maximum number of Bloom filters in the request message is a good compromise between the maximum time range for the user's query and the overhead caused by propagation of the message in the network.

b) *Routing phase*: The routing phase of DISC is executed at BCH nodes upon receiving the *Request* message from the reader device. The delivery of this message to the vicinity of the target region is done by means of geographic routing which is beyond the scope for this paper. Here we describe the routing actions taken by BCHs located in the geographical proximity of the target region. Firstly, the BCH node tests the membership of h-vectors stored in the local trace-table inside each Bloom filter of the request message as described in Section IV-A. If a specific h-vector matches one of the Bloom filters a *Locate* message is created. The BCH writes the destination address of a node from the matched entry in the trace-table, puts the matched h-vector in the payload of the message and sends it to the node that actually stores the data. At this node the h-vector is used to locate the requested data by a direct search and send the requested data back to the BCH.

V. IMPLEMENTATION DETAILS AND ILLUSTRATIVE PERFORMANCE OF DISC

The DISC protocol is currently implemented in TinyOS³ version 1.x. In this section we present the memory footprint of the protocol and selected run time performance parameters. We also demonstrate illustrative properties of DISC's backup node selection strategy which make it substantially more robust to malicious destruction of large parts of the network than the approaches using deterministic node selection.

A. Memory footprint and run-time performance

In Table I we show the technical parameters of DISC implementation compiled for MSP430 microcontroller of TelosB motes. The amount of ROM in bytes consumed by DISC's modules for both mechanisms of data dissemination and data collection is 3.7 kB. For a TelosB with 48 kB ROM DISC's code occupy 7.8% of the memory.

³Online. Available: <http://www.tinyos.net>

The maximal RAM consumption for the protocol excluding the size of the trace-table and the memory space used for storing traces and data on a sensor node is 800 B, which is 8% of 10 kB RAM memory of TelosB. The RAM consumed by the trace-table in the current implementation is proportional to the product $n \cdot n_b$, where n is the number of nodes in a cluster and n_b is the maximal number of backup entries which is an adjustable parameter of DISC. In the future development of DISC we will consider moving the trace-table into the external flash memory with dynamic loading of the needed parts. The actual backup data on sensor nodes are stored in the external flash memory.

Computing traces is the most processor resource demanding part of DISC implementation since it requires computation of hash functions. In our implementation we considered two hash functions: SHA1 [23] and RS-Hash [24]. As it is visible from the table the code complexity of SHA1 is 11 times higher than that of RS-Hash. As is shown in the table, the processing time for RS-Hash is 17 times smaller than that for SHA1.

B. Fraction of recovered information

In this section we demonstrate rather intuitive, but nevertheless illustrative properties of DISC that show an advantage of the probabilistic choice of backup nodes in comparison to the deterministic approaches. We analyze the fraction of recovered information under different types of node failures. Firstly, we consider the case of node failures in a single cluster, then we extend our analysis to address node failures in several clusters. For both cases denote N_x , the number of backup clusters for cluster x , N_c the number of nodes in remote cluster c , I_c the fraction of information stored in cluster c and I_c^r the fraction of all information recovered after node failures. Obviously, depending on the strategy of choosing the backup cluster:

$$I_c = \begin{cases} \frac{1}{N_c}, & \text{DISC choice,} \\ 1, & \text{Deterministic choice.} \end{cases}$$

1) *Node failures in a single cluster:* The fraction of the recovered information after node failure in single cluster c is

$$I_c^r = 1 - (I_c \cdot F_f^i) \quad (2)$$

In (2) F_f^i is the fraction of failed nodes that stored the information. In order to compute F_f^i we need to consider a conditional probability $P(NDown|HasD)$ of a node being down (event $NDown$) given that it has the backup data (event $HasD$). Due to independency of the events for nodes failures and storing the information $P(NDown|HasD)$ equals the probability of the node failure p_f . Now, $F_f^i = F_f^c P(NDown|HasD) = F_f^c p_f$ where F_f^c is the fraction of failed nodes over all nodes in cluster c . Finally substituting F_f^i to (2) we obtain:

$$I_c^r = 1 - (I_c \cdot F_f^c \cdot p_f) \quad (3)$$

Figure 7 shows of the recovered information depending on the fraction of failed nodes in the cluster in the case of malicious destruction of nodes, in this case nodes die with

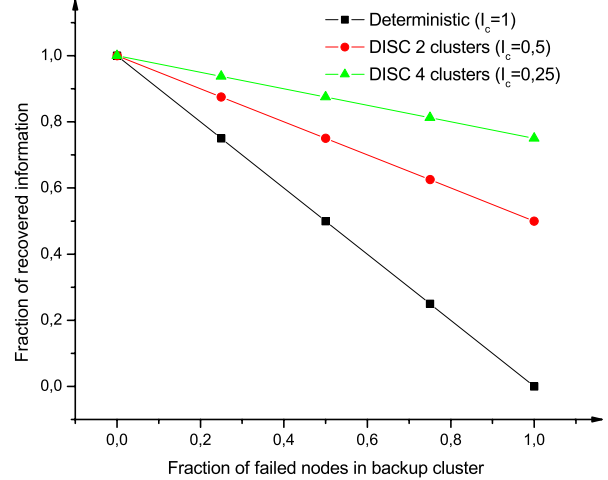


Fig. 7. The fraction of recovered information per cluster in case of malicious destruction of nodes ($p_f = 1$).

probability $p_f = 1$. The bottom curve in the figure shows the recovered information in the case of deterministic choice of the backup cluster. In this case all information from cluster x is stored in a single remote cluster c ($I_c = 1$). This situation is, for example, the case of the backup strategy in TinyPEDS. In the case of probabilistic choice of the backup region the losses in a single cluster result in a loss of only a part of information that was stored in that cluster. As Figure 7 shows when increasing the number of backup clusters in DISC the amount of the recovered information increases. Note that cases where $p_f < 1$ correspond to a normal operation of the network where nodes go down by natural reasons (battery exhaustion, hardware failures etc.) during network's life time. In these cases the graphs for the fraction of recovered information have similar shapes as those in Figure 7 with higher figures for the recovered information, we omit them due to a limited space of the paper. Although curves both for deterministic and probabilistic choice of the backup node show similar dynamics of recovering the information, it is clear that probabilistic strategy allows recovering of more than two times more information. We conjecture that overall it is more favorable than node selection strategies of deterministic approaches.

2) *Node failures in several clusters:* In the above analysis we considered the case when node failures occur within a single cluster since this case is common for both deterministic strategies for backup node selection and the probabilistic strategy of DISC. In DISC, however, the analysis should be extended considering node failures in all clusters that our protocol chooses for storing the data. The formal extension of the analysis is trivial, therefore we present only the resulting expression for the fraction of recovered information (I^r).

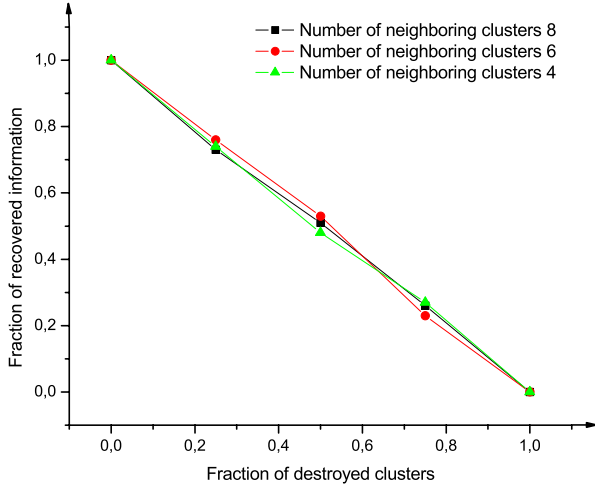


Fig. 8. The fraction of recovered information in the case of total clusters destruction.

$$I^r = 1 - \sum_{j=1}^{j=N_x} (I_c \cdot F_f^j \cdot p_j) \quad (4)$$

In (4) F_f^i is the fraction of failed nodes in cluster i over all nodes in this cluster; p_i is the probability of node failure in cluster i .

This time we demonstrate this behavior by simulations. We use TOSSIM⁴, the discrete event simulator for TinyOS to simulate the following scenario. We use a topology as shown in Figure 1 with 100 nodes uniformly distributed over all clusters. We perform three experiments with 4, 6 and 8 available backup clusters for the chosen primary cluster. This implies $I_c = 0.25, 0.16, 0.125$ in (4). In each experiment we subsequently put clusters down, i.e. all nodes in a particular cluster fail due to malicious destruction. That is for the failed cluster j in (4), $p_j = 1$, $F_f^j = 1$ while for the online cluster $p_j = 0$, $F_f^j = 0$, which corresponds to a perfectly functioning nodes in the beginning of their life time. We study the case where the PCH node stores 100 readings in its neighboring regions. In each step after shutting down the next cluster we query the network for all initially stored data. As a result we record the fraction of the available information corresponding to the fraction of the offline backup clusters. In TOSSIM we use the default settings for the radio transmission model and set the noise level in each sensor node so that only nodes inside one cluster can communicate with their primary cluster heads. We also do not model multihop transmissions between the cluster head nodes in the neighboring clusters, since our goal is to study the qualitative behavior of DISC. For this purpose we set the noise level in all cluster heads so that they appear within one radio hop from each other.

⁴Online. Available: <http://www.cs.berkeley.edu/~pal/research/tossim.html>

As for the DISC parameters, we use three hash functions to compute traces, the number of traces in Bloom filters of query messages is three. We repeat each experiment 20 times randomly seeding the simulator for each run. Figure 8 shows mean values for the fraction of the recovered information in all three experiments. The obtained results are intuitive and in accordance to (4). The fraction of the recovered information with DISC is inversely proportional to the fraction of shutted down clusters. Overall the randomness of the backup node selection in DISC makes it virtually impossible to completely destroy the stored information for a single region unless totally destroying its entire surrounding.

VI. CONCLUSIONS

In this paper we considered a problem of probabilistic selection of a backup node for distributed data storage in wireless sensor networks. We presented DISC, the protocol for distributed information storage and collection. Our protocol uses Bloom filters technique to compactly represent description of several data items inside a single query messages. The usage of Bloom filters also makes the search for a particular piece of data inside sensor nodes fast and efficient. Our major conclusion is that the probabilistic strategy of choosing the backup node makes the wireless sensor network substantially more robust against malicious destruction of its large parts. This we demonstrated by simulations and formal analysis. On the other hand it is interesting to investigate a general tradeoff between the level of robustness, the complexity of protocols using different backup strategies and energy consumption due to transmission overhead created by protocols' messages. We leave the development of this issue for our future work.

VII. ACKNOWLEDGMENTS

We would like to thank European Union (UbiSec&Sens project) and RWTH Aachen University for providing financial support for our work.

REFERENCES

- [1] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman, "Collaborative storage management in sensor networks," *International Journal of Ad Hoc Ubiquitous computing (IJAHUC)*, vol. 1, no. 1/2, pp. 47–58, 2005.
- [2] J. Girao, D. Westhof, E. Mykletum, and T. Araki, "Tiny persistent encrypted data storage in asynchronous wireless sensor networks," *Elsevier on Ad Hoc Networks*, vol. 5, no. 7, pp. 1073–1089, September 2007.
- [3] T. T. Hsieh, "Using sensor networks for highway and traffic applications," *IEEE Potentials*, vol. 23, no. 2, pp. 13–16, April 2004.
- [4] K. B. Lee and M. E. Reichardt, "Open standards for homeland security sensor networks," *IEEE, Instrumentation and Measurement Magazine*, vol. 8, no. 5, pp. 14–21, December 2005.
- [5] T. He, S. Krishnamurthy, J. A. Stankovic, and T. Abdelzaher, "energy-efficient surveillance system using wireless sensor networks," in *Proc. ACM, MobiSYS'06*, Uppsala-Sweden, June 2006.
- [6] T. Bokareva, W. Hu, S. Kanhere, B. Ristic, N. Gordon, T. Bessell, M. Rutten, and S. Jha, "Wireless sensor networks for battlefield surveillance," in *Proc. of the Land Warfare Conference*, October 2006.
- [7] U. Lee, E. Magistretti, B. Zhou, M. Grella, P. Bellavista, and A. Corrado, "Mobeyes: Smart mobs for urban monitoring with vehicular sensor networks," *IEEE Wireless Communications Magazine*, vol. 13, no. 5, pp. 52–57, September 2006.

- [8] P. Hebden and A.R. Pearce, "Bloom filters for data aggregation and discovery: a hierarchical clustering approach," in *Proc. of the IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing Conference (ISSNIP)*, Melbourne-Australia, 2005.
- [9] L. Buttyan and P. Schaffer, "Panel: Position-based aggregator node election in wireless sensor networks," in *Proc. of the 4th IEEE International Conference on Mobile Ad-Hoc and Sensor Systems*, Pisa-Italy, October 2007(accepted).
- [10] S. R. Maden, M. J. Franklin, and J. M. Hellerstein, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Transactions on Data Base Systems*, vol. 30, no. 1, pp. 122–173, March 2005.
- [11] S. Basagni, I. Chlamtac, and V. R. Syrotiuk, "A distance routing effect algorithm for mobility (DREAM)," in *Proc. ACM, MobiCom*, Dallas-Texas-USA, October 1998, pp. 76–84.
- [12] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. ACM, MobiCom'00*, Boston-Massachusetts-USA, October 2000, pp. 243–354.
- [13] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *Proc. IEEE, SenSys'04*, Baltimore-USA, November 2004.
- [14] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," in *Proc. ACM, MobiCom'00*, Boston-Massachusetts-USA, August 2000, pp. 56–67.
- [15] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, and S. Shenker, "Data-centric storage in sensornets," in *Proc. of the 1st ACM SIGCOMM Workshop on Hot Topics in Networks*, Pittsburgh-USA, February 2002.
- [16] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with GHT, a geographic hash table," *ACM Mobile Networks and Applications (MONET'03)*, vol. 8, no. 4, pp. 427–442, August 2003.
- [17] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler, "HOOD: A neighborhood abstraction for sensor networks," in *Proc. ACM, MobiSys'04*, Boston-Massachusetts-USA, June 2004.
- [18] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: a survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.
- [19] T. Repantis and V. Kalogeraki, "Data dissemination in mobile peer-to-peer networks," in *Proc. of the 6th IEEE International Conference on Mobile Data Management (MDM'05)*, Ayia Napa-Cyprus, May 2005.
- [20] G. Koloniari, Y. Petrakis, and E. Pitoura, *Databases, information systems, and peer-to-peer computing*, Springer-Verlag, 2004.
- [21] S. C.Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *Proc. IEEE, Infocom'02*, New York-USA, 2002.
- [22] A. Mohan and V. Kalogeraki, "Speculative routing and update propagation: A kundali centric approach," in *Proc. of the IEEE International Conference on Communications (ICC'03)*, May 2003, vol. 1, pp. 343–347.
- [23] D. Eastlake and P. Jones, "Secure hash algorithm 1 (SHA1)," IETF RFC 3174, September 2001.
- [24] S. M. J. Rizvi, M. Hussain, and N. Qaiser, "Comparison of hash table verses lexical transducer based implementations of urdu lexicon," in *Proc. of the Engineering, sciences and technology, student conference*, December 2004.