Monitoring Exact Top-K Values in Wireless Sensor Networks Using Dominating Set Trees

Baljeet Malhotra
Univ. of Alberta, Canada
baljeet@cs.ualberta.ca

Mario A. Nascimento
Univ. of Alberta, Canada
mn@cs.ualberta.ca

Ioanis Nikolaidis
Univ. of Alberta, Canada
yannis@cs.ualberta.ca

ABSTRACT

Top-k queries form an important class of aggregation queries in wireless sensor networks. Unlike previous proposals we consider the exact top-k values query problem, i.e., where one seeks to find exactly all k highest unique values in the network regardless of how many nodes report it. Previous proposals also did not pay due attention to the underlying logical tree topology used for data aggregation and forwarding. In this context, this paper presents two main contributions: (1) we propose the use of a particular tree topology, based on Dominating Sets, which is well suited to explore the network's physical topology for processing top-k queries efficiently; and (2) we propose EXTOK, a filtering-based algorithm for processing the exact top-k values query, we also prove its correctness and investigate its performance with respect to a number of parameters, including network link failures. In all examined cases, EXTOK performs consistently well while effectively exploiting the proposed logical tree topology and it is also resilient to link failures.

1. INTRODUCTION

A wireless sensor network (WSN) is a collection of sensor nodes that are equipped with one or more sensors, slow microprocessor, small memory, and radio transceiver. A WSN node's energy supply is limited and usually provided by batteries, making energy conservation a major issue, especially when batteries cannot be replaced or their energy cannot be replenished. There exist situations, e.g., querying for extreme behavior in an environment, where only the data of a few sensors are relevant to answering the query. For example, consider the example of sensors deployed for monitoring temperature in a forest. Clearly, the highest temperatures observed in the forest could be useful to determine risk of forest fires. Ideally, we would like to query only the nodes that observed the highest temperatures. A query relevant to this problem is the top-k query.

The precise semantics of the top-k query as considered in this paper satisfies the following requirements: (a) k is not restricted (but naturally cannot be more than the number of sensors in the network), (b) the query determines the *exact* k highest values (i.e., no approximations), (c) the query determines the *full set* of sensors that reported the k highest values, and (d) the query is executed periodically starting at some point in time and reporting values for an unbounded number of subsequent rounds (i.e., until the user wishes to terminate it). Existing proposals, discussed shortly, do not, at the same time, satisfy the requirements (a), (b), and (c).

Technical Report TR 09-01. January 2009. Dept. of Computing Science. University of Alberta. Canada. All rights reserved. Submitted for publication.

We note that requirement (c) is important because there might be more than k nodes reporting the k highest values due to possible ties in their observed values. A query providing all sensors whose temperature measurements are in the k highest values could be important for, e.g., setting priorities to manage fire-fighting resources. Ties are to be expected for a number of reasons. For example, the measured values may be coarsely quantized in order to save in terms of bits required for their representation, and hence for lowering the volume, and the energy cost, of transmitted data. Moreover, sensors located in close proximity to each other could frequently be measuring the exact same value. Finally, we expect that the larger the network, the more likely that two or more sensors report the same value.

Sensor	Sensor Value
s_1	10
s_2	15
s_3	20
s_4	23
s_5	20
s_6	15
s_7	18
s_8	16

Table 1: An example of 8 sensor values.

Consider a WSN consisting of a set of sensors, $S = \{s_i : i = 1, 2, ...N\}$. Time is discrete and counted in rounds. Each sensor produces only one value per round. Let $S_{p,j}$ be the set of sensors that produced the p^{th} highest value, $v(S_{p,j})$, during the j^{th} round. The *exact top-k-values monitoring problem* then is to find the set of k highest values, $D_j = \{v(S_{p,j}) : p = 1, 2, ..., k\}$, for each round, j. later, We also obtain the set of sensors, $\cup_{p=1}^{k} S_{p,j}$, that observed the k highest values. An example of 8 sensors and their corresponding values during a given round consider the data presented in Table 1. A top-2 query would return $D_j = \{23, 20\}$. Note that k *unique* values are produced, and because of ties, the number of nodes reporting the top-k values may be larger than k, e.g., $\{s_4, s_3, s_5\}$ in this case. Solutions not concerned with dealing with ties might return the *top-k sensors*, i.e., either set $\{s_4, s_3\}$ or set $\{s_4, s_5\}$, but not $\{s_4, s_3, s_5\}$.

A particular node in the network acts as the, so-called, *root* or *sink* node, which is responsible for disseminating the queries issued by a user into the network, and to return the results of such queries back to the user. A trivial solution to top-k queries is a centralized approach whereby, in every round, all sensors send their measurements to the sink which then locally calculates the top-k values. This solution is of little practical interest because it introduces large communication overhead, and hence energy consumption. An alternative approach is to construct a spanning tree rooted at the sink, and impose some form of aggregation and coordination to save on communication overhead. An example of such a logical tree is the Shortest Path Tree (SPT) which is used by several existing solutions, e.g., [9, 17].

Earlier solutions, e.g., TAG [9] require every node to send an update (containing its own value or aggregated values) during every round irrespective of the fact that only k such values will eventually become part of the actual answer. In an ideal solution only the sensors that have values in the top-k should send their values to the sink. Unfortunately, these nodes do not know of their own "special" status *a-priori*. Filtering based solutions e.g., FILA [17], have been proposed recently for suppressing updates from nodes that are *unlikely* to become part of the solution to the top-k query. The intuition behind using filters is that nodes that reported the top-k values during a round are more likely to produce the top-k values again in the next round. It also means that the updates from the sensors, which had not produced the top-k values are potentially not required to compute the result in the next round.

In this paper we present two contributions. We first explore the fact that the underlying logical tree structure plays an important role in the query processing cost. Next, we design an efficient filtering-based solution that exploits the underlying logical tree topology effectively to answer the top-k queries, while carefully adhering to the query semantics as defined above.

When designing a logical structure to process the top-k queries our intuition is that by exploiting the spatial proximity of sensors one can create a logical tree in which messages between nodes of a tree can be exchanged efficiently, and local decisions can be made to suppress sensors updates more effectively. We use clusters for efficient dissemination of messages and local decision making before routing intermediate results further up the tree. In particular we propose the use of a Dominating Set [16] to exploit the physical topology of a given network and assemble a logical tree that allows more efficient processing of the top-k queries. Through the use of Dominating Set nodes we essentially form clusters in which dominating nodes play the role of cluster-heads. Local processing can then be performed in these clusters avoiding unnecessary data communication further up in the tree, hence decreasing the query processing cost and increasing the network lifetime. Cluster-heads can communicate with the root via multi-hop shortest paths. We call this structure a Dominating Set Tree (DST). Towards the second part of our goal we present a new filtering based algorithm, which we name EXTOK—for EXact TOp-K (values). To the best of our knowledge, EXTOK is the first algorithm for top-k queries in WSN that satisfies all four requirements (a)-(d) presented above simultaneously. We implement EXTOK on SPT as well as on DST, and the results confirm that the underlying logical tree does play a non-negligible role on the query performance. Furthermore, we examine EXTOK's performance extensively, using both real and synthetic datasets, and we find that EXTOK is an overall very efficient solution.

The remainder of the paper is organized as follows. Next we review related work noting that several versions of the so-called top-k query has been researched. In Section 3 we elaborate on the construction of a DST and the reasoning for the same. Section 4 details EXTOK, the new filtering based solutions we propose, it also presents a proof of EXTOK's correctness. Next, in Section 5 we evaluate, and compare with previous proposal, the performance of EXTOK both on SPT and DST while using real and synthetic datasets. Finally, Section 6 concludes the paper.

2. RELATED WORK

Top-k queries in distributed systems is a widely studied problem. Olston *et. al.* addressed the problem of caching approximate values with an appropriate precision [11]. Their work lead to the idea of implanting filters in a distributed environment to suppress communication messages. Babcock and Olston [1] extended that and applied the idea of cached values for the top-k monitoring problem. The key idea is to use the cached values as range-based arithmetic filters. Filters are adjusted dynamically when they are violated. A coordinator node monitors the filter constraints of the rest of the nodes and also maintains the top-k result.

This fundamental idea of installing filters for suppressing unnecessary updates has turned out to be especially useful with WSNs. A number of algorithms proposed in the literature [13, 15, 17] rely on this idea for continuous monitoring of sensor values, a problem that is closely related to the problem that we investigate in this paper. The difference among the previously proposed algorithms lies in the various strategies being used for maintaining the filters at successive periods. Our work is different from [13, 15, 17] in the sense that we adhere to the precise semantics set forth in Section 1. Recall that we seek the *exact* top-k unique values and the full set of sensors that observed them in the WSN. FILA [17] tracks the top-k *sensors*, i.e., a subset of k sensors that has observed the highest values within the WSN, and as we illustrated in Section 1 (using data from Table 1) FILA does not guarantee to find the complete and correct set of k values within the WSN. Furthermore, while it makes the typical assumption that a multi-hop path has be to be traversed by messages towards the sink, it assumes that messages from the sink can reach all nodes in a single-hop. In [13], Silberstein *et. al.* propose solutions that combine the idea of *temporal* and *spatial* suppression for continuously collecting "all" sensor values from the WSN. Clearly this problem is different than the problem that we consider. In yet another study, Silberstein *et. al.* carried out a detailed investigation of MAX queries in [15]. An important conclusion of their study is that threshold based filtering solutions are more effective than the range based filtering solutions proposed in [1]. We note that while in our work we also use the notion of threshold values for suppressing updates, unlike Silberstein *et. al.* we do consider the case of tied sensor values, which impacts their filtering based solution. In fact, as we shall see shortly, handling ties in a provably correct manner, which EXTOK does, is important and non-trivial. In our previous work [10] we already made use of the DST for processing MAX (top-1) queries, where only the value itself was of importance and the corresponding node ids were irrelevant. That solution cannot be simply generalized for the exact top-k values query that we now consider. Finally, none of the solutions proposed in [13, 15, 17] pay due attention to the underlying logical topology, which is part of our solution.

Not surprisingly, several other versions of the top-k problem exist. For instance, a variation of the top-k query is to rank the objects based on the aggregated scores on a set of attributes stored at distributed locations. The Threshold Algorithm [4] is the best known solution for this problem. The main constraint of this algorithm is that it assumes single-hop communication. Zeinalipour-Yazti *et. al.* propose a solution to a similarly defined problem but which is developed in the context of a multi-hop WSN [18]. This exemplifies the observation that solutions proposed for a distributed, albeit wired, system which is also rich in resources are not directly applicable to WSNs.

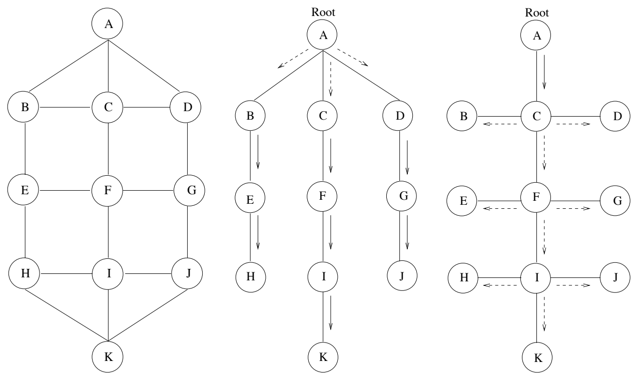
Similar, yet different problems have been investigated elsewhere. In [14] the authors use a model-based optimization technique for answering the approximate top-k queries; the goal is to minimize the number of true answers missed in the approximate answer. More recently the authors of [3] propose exploiting the spatially correlated sensor data to build partial order trees (POT) to answer the top-k queries. Their main idea is to select "hot spots" (sensors with highest readings) in a network to build a logical topology to reduce unnecessary sensor updates. While it works well for highly non-uniform data distribution its performance degrades rapidly when there are very few or no hot spots. In this paper we make no assumptions with respect to the distribution of data values in the WSN.

There are also solutions that only apply aggregation and do not use any filtering mechanism. TAG [8] is a classical example of such non-filtering based approach that can be used for the top-k query problem. In TAG an underlying logical tree topology is used in which, before forwarding the data to its parent, a node aggregates the data received from all of its children.

3. DOMINATING SET TREE

A common characteristic among most of the previous work, e.g., [1, 3, 13, 15, 17, 18] is that while they exploit the underlying logical tree structure for performance gains, they do not seem to devote the necessary attention on how to assemble such a tree. Consider a WSN with the physical topology depicted in Figure 1(a), where edges represent the ability of a pair of nodes to communicate directly, i.e., they are within each other's communication range. Assume that node A is the sink node. To start the top-k query processing, the first step is to disseminate the query in the network to all sensors. An obvious solution is for A to broadcast the query in its neighborhood, and request all of its neighbors to do the same. This is repeated until all nodes in the network have received the query, i.e., query dissemination by means of *flooding*.

In this example we note that several nodes can be reached by one single transmission, due to the broadcast nature of wireless transmissions. Therefore, a sensible alternative to flooding is to use a logical topology, e.g., a SPT built on top of the physical topology to disseminate the query. For



(a) Network (b) SPT (c) DST

Figure 1: A network of eight nodes. Solid lines represent edges, and arrowed lines represent messages with the distinction that all arrowed dashed lines coming out from a node represent a single broadcast message from that node.

our specific network shown in Figure1(a), a SPT is shown in Figure1(b). Using a logical tree topology a query can be received by all nodes if every node except the leaves of the tree broadcasts the query. The specific SPT requires a total 8 messages (transmitted by nodes A, B, C, D, E, F, G, I) to disseminate the query.

A better alternative to SPT is to ensure that only the smallest possible subset of nodes relay the query, as long as the transmissions from this particular subset of nodes ensures that all nodes receive the query. The equivalent graph problem is that of determining a Minimum Connected Dominating Set (MCDS) [2]. Given a graph $G(V, E)$, a Dominating Set (DS) is a subset of node V' from G such that all nodes from the set $\{V\text{-}V'\}$ are connected to (are neighbors of) at-least one of the nodes in V'. If the nodes in V' are connected, then they form a Connected DS (CDS). The CDS serves as the basis for ad-hoc network routing, e.g., to form the Multi-Point Relays (MPR) [19]. Frequently, the CDS is seen as a structure of interconnected clusters [7] where the dominating nodes are called *cluster-heads* in recognition of the fact that they "represent" collectively, neighboring (dominated) nodes. Naturally, we are interested in the Minimum CDS (MCDS), as it represents the minimum number of transmissions to reach all nodes in a network. For the network shown in Figure 1(a), nodes C, F, and I constitute an MCDS. Unfortunately, finding an MCDS is known to be an NP-hard problem, and the quest for a tight approximation remains an area of active research [2]. Nevertheless there exist distributed algorithms that approximate the MCDS to within a constant factor, at the cost of $O(|V|)$ time complexity and $O(|V| \log |V|)$ message complexity. A distributed approximation algorithm presented in [16] is based on the concept of Maximal Independent Set (MIS), which is closely related to DS. The MIS is a subset of vertices characterized by the property that no two vertices are adjacent, and the set of vertices is maximal (no further vertex can be added without breaking the property). It is trivial to show that an MIS is also a DS (but the converse is not necessarily true). We use a similar approach, whereby through an initial MIS construction, we eventually build a tree, i.e., a connected subgraph, rooted at the sink node. We will call it a

Dominating Set Tree (DST). An obvious problem then is to construct a minimum cost DST (MDST), which is equivalent to finding a *constrained* MCDS such that the sink node must be part of the solution, i.e., the sink node is included in the MCDS. The proof that finding the solution for this constrained-MCDS problem is also NP-hard in included in the Appendix.

For the network shown in Figure 1(a) the corresponding DST rooted at A is shown in Figure 1(c). The cluster-head nodes C, F, and I that constitute the MCDS become the non-leaf nodes of the DST. Furthermore, sets of nodes {B,D}, {E,G}, and {H,K,J} that are cluster members of nodes C, F, and I, respectively, become the leaf nodes of the DST. In this example a query can be disseminated using 4 messages (transmitted by nodes A, C, F, I) using the DST compared to 8 messages for the SPT.

3.1 DST Construction

We assume that the DST construction is a one-time cost, performed at the beginning of a network's lifetime. For this reason, we assume it is acceptable to construct a DST off-line. We consider fairly simple distributed construction schemes that assume that each node is aware of its neighbors and it can support reliable message transfers. In the beginning of the network deployment first we construct an MIS, and based on the MIS we construct a DST.

DST construction is a two phase process that is triggered by the sink node by sending a construction request message. In the first phase, each node that receives the construction request, forwards the request while incrementing the distance metric (included in the message) that keeps track of the number of hops from the sink (hop-count). In the same message each node announces its weight to its neighbors. As weight of a node we use the ratio of its degree over its hop-count. Each node also selects a prospective parent (for the DST to be constructed) that has a minimum hop count to the sink, breaking ties using the node ID.

In the second phase, a node becomes a cluster-head if it has the largest weight among its neighbors (node ID is used to break ties). If it does not have the largest weight among its neighbors, it waits until its neighbors with higher weights have determined whether they will become cluster-heads or not. If none of the higher weight neighbors has become a cluster-head, the node assumes the role of the cluster-head, and announces this fact to its neighbors. The cluster-heads form an MIS, and the DST is formed by the union of the shortest paths from all MIS (cluster-head) nodes to the root. Cluster members that find themselves on the shortest paths of the MIS nodes to the root become *connectors*. The rest of the cluster members become leaf nodes of the DST by choosing their corresponding cluster-head as their new parent. Recall that all nodes had selected a parent having a minimum hop count to the sink during the first phase itself. In the second phase, only cluster members that are not connectors update that their cluster-head is their new parent. In the rest of the paper we often use the term dominating nodes to represent the non-leaf nodes (that are either cluster-heads or connectors), and non-dominating nodes to represent the leaf nodes (or cluster members) of a DST.

The MIS construction outlined here is essentially the same as in [16]. Like above, [16] also builds a tree from the constructed MIS. Internal nodes of the constructed tree are eventually chosen to represent the CDS. The basic difference from [16] is that in our DST construction, the root of the tree should be the one given, i.e., in DST, the root of the tree is part of the input. Recall that our DST construction is triggered by a particular sink node that will eventually become the root of the DST.

Throughout this paper we assume that an ideal transmission schedule is also constructed that dictates when nodes should listen, transmit, or sleep. TDMA schemes for unicast and broadcast communication proposed in the literature can be used for this purpose [12]. A naïve solution to this problem is to allocate time slots for every task that nodes need to perform in order to process the query, e.g., directing when a child (within a cluster) should transmit to update its parent, and when a parent should be awake to receive values from its children, and so on. This naïve solution is likely not efficient for filtering based algorithms, because the nodes that were scheduled to transmit in their particular time slots may never use their time slots because of their filtering constraints. A detailed investigation of the scheduling problem is out of the scope of this study. For the purpose of this paper it is sufficient to assume that a schedule is in place for nodes to perform every task that is required to process the top-k queries.

3.2 Failures and Recovery

The remaining of this paper assumes reliable communication. We are aware though that unreliable communication is a fact of life, in particular in wireless communications. In fact, in Section 5.3, we investigate the impact of unreliable communication (specifically, link failures) on the correctness of the produced top-k result. Due to space limitations, we do not elaborate on protocols that could be used to deal with failures. Nevertheless, for the sake of completeness, we sketch in the following paragraphs the basic ideas behind them. For a variety of reasons, energy depletion being the main one, node failures are possible. Certain node failures could result in network partition, in which case nothing (short of replacing them with new nodes) can be sensibly done to reconnect the network. However, we expect that a large number of node failures still leave the network connected. Nevertheless, such failures can still disrupt the logical topologies we consider here, i.e., the trees used for query dissemination and execution/collection. More insidiously, there could be temporary *link* failures, i.e., failures for nodes to communicate due to the condition of the wireless channel because of noise, interference, the movement of obstructions, etc. Such failures are quite common but should not be treated the same way as node failures since they represent temporary disruptions. In this latter case, a node that was supposed to report its value might not be able to get its transmission across, leading even to incorrect query result. Next, we discuss briefly some approaches to dealing with failures. A naïve solution is to reconstruct the logical tree from scratch. Any perceived failures are reported to the root, and the root re-initiates the distributed, albeit costly, re-construction of the tree. Nevertheless, for particular kinds of failures, less costly alternatives exist that attempt to fix the logical topology locally. For example, children of a failed (unresponsive) parent node can initiate a local reconstruction by choosing a new parent among their neighbors. We note that any optimality (or bounds to optimality) guaranteed by the complete re-construction of the tree are typically absent from techniques that rely on local repairs. Therefore, local repairs should be seen in the light of providing a compromise that avoids costly (but with possible optimality properties) global reconstruction, by providing instead "inexpensive" (few messages) and fast response to failure events. By the same token, the exclusive reliance on local repairs may, after several failures have been repaired, render the logical tree significantly distant from the optimal one. Hence, the ability to, on occasion, reconstruct the entire logical tree might still be useful even when local repairs are employed.

Fortunately, when it comes to locally repairing a logical tree, several relevant techniques that were originally proposed for the maintenance of multicast routing trees exist [5]. Some of them can be used (with various modification) to maintaining the trees, i.e., SPT and DST. As an example of simple local repair strategies, consider first the case of SPT. Children of a failed parent can broadcast a control message to find their new parent. Nodes that receive this message may reply back with their "tree level" (hop count to the root). The disconnected children can choose a parent from among the replying nodes with the smallest tree level. Leaf node failures do not elicit any repairs. Similarly, in a DST, when a leaf node fails no action is taken. However, in a DST, when a dominating node fails, its children can broadcast a message to find another dominating node among their neighbors (i.e., either a cluster-head or a connector). Suitable nodes reply back and the requesting node selects as parent the one with the lowest hop count to the root. If no reply is received, then the requesting node was dominated by only one dominating node (the one that failed) and subsequently the requesting node will connect to a non-dominating node (from within its neighborhood) that has the lowest hop count to the root. The chosen non-dominating node will then become a dominating node. A more detailed study on failures and recovery techniques in DST is outside the scope of this paper and is subject of our current research. Nonetheless, the impact of transient link failures in the query results is investigated in Section 5.3.

4. PROCESSING EXACT TOP-K QUERIES

In the following we look into two ways in which the filtering based methods for the top-k queries *in particular* can benefit from the DST structure. We note that if the underlying logical tree structure is an SPT, then the paths followed from nodes to the root/sink are optimal in terms of the number of hops, but the same is not true for a DST (see, e.g., nodes B, D, E and G in Figure 1). This apparent sub-optimality of a DST for routing data will be countered by benefits from reducing the amount of traffic mainly by installing *filters* and through efficient dissemination of messages in the tree.

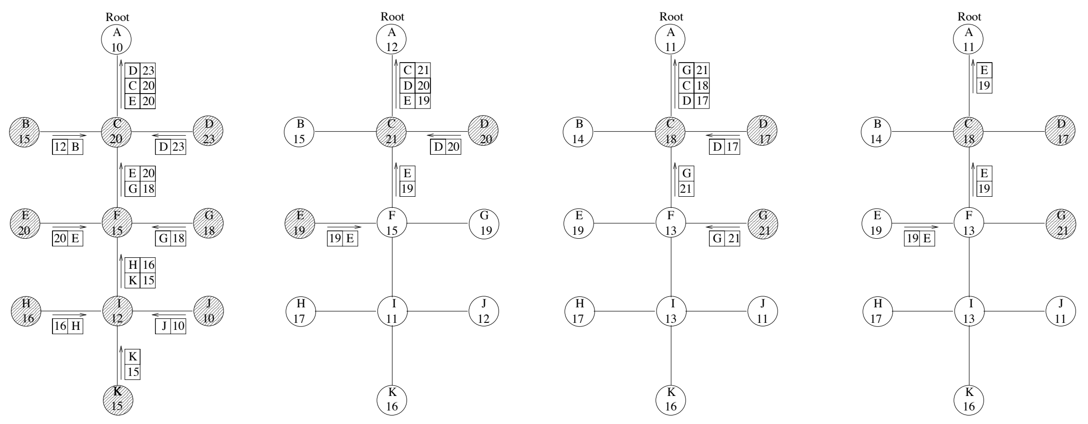
Filtering based solutions require a "feedback" mechanism between the root and the nodes, mainly to update the filters and to validate the top-k result [17]. In essence, this filtering based solution employs two types of communication messages that are exchanged between the nodes of a tree: (i) nodes-to-root messages, i.e., updates triggered from the nodes, and (ii) root-to-nodes messages, i.e., filter updates and validation messages triggered from the root. A DST structure is well suited for the second case because of its clustering characteristics; messages from the root can be broadcasted efficiently. DST also allows for efficient and early aggregation. In the following we present EXTOK assuming a DST logical topology.

Before we detail how the algorithm works we present an overview of the whole process. EXTOK's execution can be thought of as starting from the leaf (non-dominating) nodes and progresses towards the root. Every node runs the algorithm during every round according to its given schedule and in accordance with the precedence constraints imposed by the logical relationship. In the first round EXTOK works similarly to TAG, in which all nodes send their updates, and the root after collecting values finds the top-k values. The root also calculates a threshold value, henceforth referred to as τ which is the minimum value of the current top-k values, and that will be sent to the nodes and installed as their filter. At this point nodes enter in one of two operation modes. A node is said to be in a "temporal monitoring" mode (TM-node for short) if it produced one of the current top-k values. TM-nodes are required to report *any* changes to their current value at *every* round as it may yield a change in the current top-k answer set. Otherwise, a node is said to be in a "filtering" mode (or a F-node) if its value did not contribute to the current answer set. F-nodes are required to report their new values only when they observe a value that could belong to the answer set, i.e., a value that is greater than or equal to τ.

After the first round each subsequent round in EXTOK consists of three stages. In the first stage nodes trigger their update according to their operating mode. After receiving values during the first stage, the root proceeds to validate the current top-k results, and, if necessary, it initiates a validation procedure at the end of the first stage. During the second stage of the algorithm, *some* nodes may reply back in response to the validation procedure invoked by the root. At the end of the second stage the root determines the correct answer to the query. During the third stage the root adjusts the value of τ based on the newly computed result, and informs all other nodes about it.

We illustrate the execution of EXTOK during the three initial rounds for a top-2 query in Figure 2. Values observed by the nodes during a particular round are depicted within the respective nodes. In the first round all nodes are TM-nodes (shaded in Figure 2) (except the root that is not required to trigger its update during any of the rounds), and $\tau = -\infty$[1]. In the first round (Figure 2(a)) every node sends its update to the root. As the nodes push their values up in the tree, aggregation is performed by parent nodes. For instance, node I, after receiving values from its children H, J and K, discards the values {10, 12} to forward its local top-2 values only, i.e., {16, 15}, to its parent F. Note that since EXTOK considers the unique top-k values, the local top-2 result forwarded by node C consists of the three pairs {⟨C, 20⟩, ⟨D, 23⟩, ⟨E, 20⟩} (2 unique values being observed at 3 different nodes). Finally, the root finds the top-2 result and also determines τ, which is the lower bound on the current top-k values (20 in this particular example). The root broadcasts τ after which only nodes C, D and E become TM-nodes, i.e., if their values change, they must propagated their updated value. The rest of the nodes will trigger an update only if their value is greater than $\tau = 20$ (i.e., they become F-nodes). During the second round nodes C, D and E change their values and trigger updates as shown in Figure 2(b). In order to correctly compute the results the

[1]In general, this should be set to the application's minimum meaningful value.



(a) Round 1 (Answer Set: {D:23, C:20, E:20}) (b) Round 2 (Answer Set: {C:21, D:20}) (c) Round 3 (Answer Set needs to be validated (d) Round 3 (Answer set validated: {G:21, E:19})

Figure 2: Initial rounds of a top-2 query on DST.

root always requires the current values of the TM-nodes that changed during a given round, therefore, their values cannot be aggregated *en-route*; note that E's value is propagated all the way to the root, by-passing the aggregation. Since the top-2 values received by the root during the second round (i.e., 21 and 20) do not invalidate $\tau = 20$, and also the value of every filtering node is less than 20 (otherwise they would have triggered their own update) the root can correctly find the top-2 values in the second round without taking further action. Note that at the end of the current round E has not received an update for τ and since its own value is smaller than τ it becomes a F-node automatically.

Changes to the values in nodes C and D in the third round (Figure 2(c)) trigger a propagation of their update. Node G (which was a filtering node) also triggers its update because its new value is above τ. At the end of the stage one in the third round the root receives values {21, 18, 17} yielding {21, 18} as the (temporary) answer set. At this point the root finds that the current top-2's lower bound (18) lower than the current value of τ (20). This means that other unreported values from F-nodes may now be part of the answer set. To determine the new correct result, the root sends a validation query in the tree seeking values that are greater than or equal to 18. In response to the validation query, node E replies back with its value (Figure 2(d)). The root finally determines the (guaranteed) correct top-2 values, i.e., {21, 19}, and also updates $\tau = 19$, which is propagated down in the tree to update the nodes' filters. Thus after round 3, E and G will become TM-nodes, and C and D will become F-nodes. Pseudocode for EXTOK's algorithms for the root node and all other nodes is described in Algorithms 1 and 2 respectively in the Appendix.

4.1 EXTOK's Algorithm

4.1.1 Initialization

In the first round every node but the root sends its update. Aggregation is applied by the dominating nodes of the tree. If the total number of values received by a dominating node plus its own value is less than k then that node sends all values to its parent; otherwise it sends the top-k values only. After receiving values from its immediate children the root can determine the top-k values and the corresponding nodes that generated them.

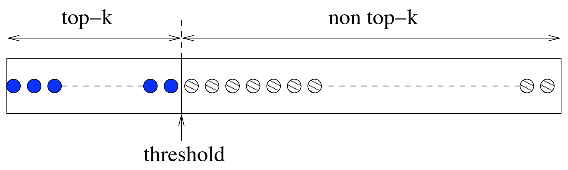
The root computes an arithmetic filter for the non top-k nodes to suppress unnecessary updates in the subsequent rounds. For that the root uses a threshold value, τ set to the minimum value of the current top-k values. (Line 35 in Algorithm 2.) The root broadcasts τ so that all nodes can update their filters. (Line 37 in Algorithm 2, and lines 28 and 35 in Algorithm 1.) (For the sake of efficiency we make the assumption that if a new τ is not broadcast within a round the nodes will behave as if the same current τ had been broadcasted.) Let $v(s_{i,j})$ be the value of node s_i during the j^{th} round. After τ is updated, a node s_i becomes a TM-node for round $(j+1)$ round if $v(s_{i,j}) \geq \tau$, otherwise it becomes a F-node. After that EXTOK works in three sequential stages detailed next.

4.1.2 Stage 1

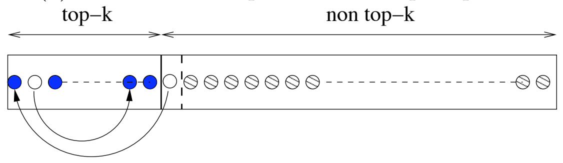
During this stage a TM-node, s_i, triggers its update only if $v(s_{i,j}) \neq v(s_{i,j-1})$, while a F-node, $s_{i'}$, triggers its update only if $v(s_{i',j}) \geq \tau$. (Refer to line 4 in Algorithm 1.) The root requires the new values of all triggering TM-nodes in a given round, therefore their values cannot be aggregated *en-route*. However, the values of the filtering nodes can and indeed are aggregated *en-route*. After the root has received values from all of its children, it determines the correctness of the current top-k result. As represented in Figure 3 there are a few cases that the root needs to consider and which we discuss next. For the sake of explanation we use Figure 3(a) to illustrate the set of top-k and the non-top-k values separated by the current threshold (τ) value, further we refer to the former as "top-k-space".

S1a: In this case we consider updates triggered by changes in TM-nodes only, in particular changes that happen *within* the top-k-space (Figure 3(b)). The only interesting event is when there is a new and higher value for the lower bound of the current top-k values. In this case a new τ is computed and broadcast to all nodes. Consequently some TM-nodes may now become F-nodes. This scenario has no effect on current F-nodes.

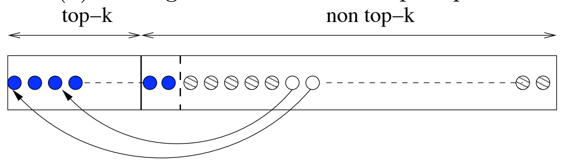
S1b: Next we consider changes triggered by updates from F-nodes only, i.e., changes that happen outside the



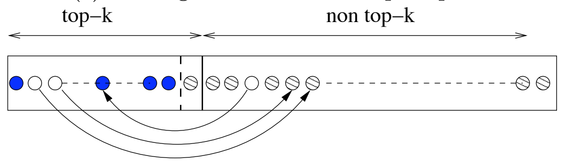
(a) Values in the top-k and non top-k spaces.



(b) Moving values within the top-k space.



(c) Moving values into the top-k space.



(d) Moving values in and out of the top-k space.

Figure 3: Various scenarios of changing values that can impact the top-k result and the threshold setting. The dark-color filled circles represent the top-k values and the circles with a filled pattern represent the non top-k values. A non-filled circle represents a "space" created by a moving top-k or non top-k value. Solid (dashed) vertical lines represent the new (previous) threshold values.

top-k-space (Figure 3(c)). By definition this means that the new values of these nodes are greater than or equal to τ, and also that those nodes may become TM-nodes. It is also possible that a new τ exists and needs to be broadcast, and again depending on that value, TM-nodes may need to switch to F-nodes and vice-versa. (Refer to lines 15∼17 in Algorithm 2.)

S1c: The last case of interest is more general; values are coming in or leaving the top-k-space (Figure 3(d)). The first situation can be triggered by certain updates from either TM- or F-nodes, while the second happens when a TM-node's value that is unique falls below the current τ. When the root receives all updates it will have k' values that are equal or above the current τ value and m values that are below τ. If $k' \geq k$, the root computes the current lower bound τ and, if there is a change, broadcasts it. As before, nodes may then need to switch their operating modes accordingly. Even if τ does not change, and therefore is not broadcasted, nodes may still switch their mode since they know their own values and can presume that τ did not change. A more interesting scenario however, is when the root ends up with $k' < k$ values greater than τ. (Refer to lines 18∼25 in Algorithm 2.) Then there are two cases to consider depending on the relation between $k - k'$ and m, in both cases the root determines a suitable *probe value* to be sent in a *validation query* to all F-nodes; their responses are considered in Stage 2 (which we discuss shortly.). The purpose of the probe is to restrict the subset of the nodes that need to respond to the validation query.

– If $k - k' \leq m$ the root has enough values to complete the answer set, but it is possible that F-nodes that did not trigger an update, and therefore whose values are not known to the root, should be part of the answer. (Refer to lines 20∼22 in Algorithm 2.) To solve that potential problem the probe value is set equal to the $(k-k')^{th}$ highest value from the set of m values of sensors that have dropped below τ.

– On the other hand if $k - k' > m$ the root does not have enough values to complete the answer set and is unable to set a sensible bound for the probe value therefore it operates similar to the first "TAG-like" round over the F-nodes (TM-nodes need not be queried as they would have already sent useful updates by themselves), and the probe value is set to $-\infty$. (Refer to lines 23∼25 in Algorithm 2.)

To illustrate the situations above consider an example where $\tau = 30$, $k' = 8$ and the set of m values below τ at the root is $\{25, 22, 19, 18, 15\}$. If $k = 10$, then the root needs $k - k' = 2$ values to complete its answer set. Only values greater than or equal to 22 need be considered since the root already has values above it to complete the answer set, thus the probe value in the validation message is set to 22, i.e., the 2^{nd} ($(k-k')^{th}$) highest value among the m values available. If $k = 15$ then the root has no means to set a bound on the probe value as it has less values than it needs, thus it sets the probe to $-\infty$ in order to guarantee a correct answer.

4.1.3 Stage 2

In the second stage all F-nodes, which had not triggered their update in the current round, reply back in response to the validation query only if their value is greater than or equal to the probed value. (Refer to lines 14∼25 in Algorithm 1 and line 28 in Algorithm 2.) Aggregation is applied *en-route* and can actually be "tightened". Note that the node needs no more than $k - k'$ values, thus if a node receives more than $k - k'$ values, then it forwards only top $k - k'$ values to its parent. The root may or may not receive any values in response to the validation query. In any case the root can correctly determine the top-k result from the values it has received from the TM-nodes during the first stage, plus the F-nodes that have replied back in response to the validation query, if any, during the second stage, in addition to the k' values above τ it already has. At this point the root can recompute a new value for τ from the newly computed top-k values, and if it changes, broadcast it.

4.1.4 Stage 3

In this "concluding" stage the root updates the nodes about the new threshold τ, and the nodes can set their mode accordingly. In the absence of a threshold update message during a given round nodes can safely assume that the threshold value has not changed and they set their (possibly) new mode for the next round based on their current value alone. (Refer to lines 28∼35 in Algorithm 1 and line 29∼38 in Algorithm 2.)

4.2 EXTOK's Correctness

The correctness of EXTOK's algorithm in the first round is straightforward to establish, as its behavior is very much similar to TAG's. The following theorem asserts and proves its correctness for subsequent rounds. We assume the availability of two functions: $V(S)$ that returns the number of unique values within a given set of sensors S, and $v(s_{i,j})$ that returns the value of a given sensor node s_i at round j.

THEOREM 1. *In any given round $j \geq 2$, EXTOK produces a correct top-k result.*

PROOF. Given a set S of nodes $s_{i,j}, i = 1, 2, \ldots N$, during a round j, let $S_{t,j}$ and $S_{f,j}$ be the sets of TM-nodes and F-nodes from which the root received values in the j^{th} round and let $S_{c,j} = S_{t,j} \cup S_{f,j}$. Further, let $S_{\tau+,j} = \{s_{i,j} \in S_{c,j},\ s.t.\ v(s_{i,j}) \geq \tau\}$ and similarly $S_{\tau-,j} = \{s_{i,j} \in S_{c,j},\ s.t.\ v(s_{i,j}) < \tau\}$. τ is set to the minimum value in the current answer set, i.e., it is the smallest of the current top-k values. Finally, let $k' = |S_{\tau+,j}|$, and $m = |S_{\tau-,j}|$. We distinguish two cases: $k' \geq k$ and $k' < k$.

C1: If $k' \geq k$ the root has at least k values that are greater than or equal to τ. By construction, the values that root does not know must be less than τ, therefore the root must have the exact top-k values.

C2: If $k' < k$, then the root requires additional $k - k'$ values to find the top-k result. Further $S_{\tau-,j} \neq \emptyset$ (otherwise we would necessarily have $k' \geq k$) and it contains only those TM-nodes whose values have fallen below τ; F-nodes will only send updates if their values become greater than or equal to τ. Recall that in Stage S1c of the algorithm the root sends a validation query containing a proper probe value $v(s_{q,j})$ leading to the following two sub-cases.

C2a: If $k - k' \leq m$, then the probe value, $v(s_{q,j})$, is the $(k-k')^{th}$ highest value from the set, $V(S_{\tau-,j})$. Assume the root receives k'' values in response to the validation query. By construction all k'' values that the root receives are greater than or equal to $v(s_{q,j})$ but smaller than τ otherwise they would have triggered an update and be known to the root already. The root now has k' values that are greater than or equal to τ, and $k - k'$ values that are less than τ but greater than or equal to $v(s_{q,j})$, and additional k'' values (received in response to the validation query) that are also less than τ but greater than or equal to $v(s_{q,j})$. The root now is guaranteed to have at least k values that are greater than or equal to $v(s_{q,j})$ and all other values in the tree are, again, by construction smaller than $v(s_{q,j})$. Thus the root must be able to find the correct top-k values.

C2b: If $k - k' > m$, then $v(s_{q,j}) = -\infty$, which means that the root will receive answers, possibly aggregated *en-route* from every F-node that had not triggered its update during the first stage, and clearly there will be enough values (current plus newly received ones) at the root for the correct top-k values to be found.

□

5. PERFORMANCE EVALUATION

In order to evaluate our proposal we used both synthetic and real datasets. The synthetic dataset was generated by simulating a network of sensors deployed in a 200m×200m area. Using this dataset we performed experiments by varying five parameters: number of sensor nodes (N), number of top values sought (k), wireless/radio range (ω), probability that a sensor's value changes between two consecutive rounds (γ) and percentage of change in sensor's value (δ). Table 5 shows the values used for each such parameter. For the placement of the sensors, the monitored area was divided into smaller cells, in particular 4 cells of equal size. Sensors were then placed randomly in each of these cells. Dividing the area into smaller cells helped ensuring that randomly placed sensors were able to form a connected network. To investigate the impact of randomly changing values (sensors measurements) on the performance of the algorithms we generated "temperature" values for sensors. The initial value of sensors was randomly set between 10 and 50 and could vary between rounds according to parameter δ (equally likely to be a negative or positive change).

Parameter	Values
N (# of sensors)	100, 200, **300**, 400, 500
k (# of top values)	1, 5, **10**, 15, 20
ω (radio range [m])	25, 30, **35**, 40, 45 (synthetic data)
	and 8, 10, **12**, 14, 16 (Intel data)
γ (prob. of change)	0.1, 0.2, **0.3**, 0.4, 0.5
δ (change [%])	2, 4, **6**, 8, 10

Table 2: Parameter values used in the experiments (default values are shown in bold face).

Results using the synthetic dataset are based on an average of 10 simulation runs in which each run consists of 200 rounds. In each of the simulation runs the root node was chosen randomly. Also, in each of these runs the placement of sensors was also random within the layout detailed above.

The experiments with real data was performed using the Intel Berkeley dataset[2], which consists of approximately 3.5 million readings from 54 sensors deployed in the Intel Berkeley Research lab. A few sensors for which the location was not know were removed from the dataset, and missing values in the data were replaced by using linear interpolation. Sensor readings were originally maintained by epochs, a monotonically increasing number for each of the sensors. We organized these sensor readings in such a way that the dataset has 60,000 rounds, each one containing one value for each of the 54 sensors. Note that in the case of this dataset only parameters k and ω are investigated using the original placement of the sensors, and having one such node randomly chosen as the root node for each run. As before the reported results are an average of 10 runs.

[2]http://db.csail.mit.edu/labdata/labdata.html

As discussed in Section 2, there are previously proposed approaches that deal with similar but yet *different* versions of the top-k monitoring problem, hence a direct comparison to those is not applicable. Nonetheless, TAG can be used to solve the same problem we solve, therefore, in order to have a baseline to evaluate our proposal, we implemented TAG as well. Our experiments have clearly shown that DST is consistently a better alternative than SPT for both EXTOK and SPT. Nonetheless, for the sake of completeness we show results obtained by using both TAG and EXTOK implemented on top of both SPT and DST topologies. Henceforth the notation X-Y means algorithm X implemented over logical topology Y. Unless otherwise noted, in the performance figures that follow we compute the transmission cost of TAG-DST, EXTOK-SPT and EXTOK-DST and present their performance results as a percentage (%) of improvement (positive gain) or deterioration (negative gain) over the transmission cost of TAG-SPT.

5.1 Transmission Cost

Results from our experiments using the synthetic dataset are summarized in Figures 4–8. In the first experiment we evaluate the impact of varying k on the EXTOK's performance (Figure 4). The foremost trend that we can see is that EXTOK-DST outperformed TAG-SPT by at least 70%. DST provides an efficient backbone for root-to-node communication which occurs often in EXTOK. Updates from nodes-to-root (which are more costly in DST) are significantly suppressed due to installed filters thus yielding an overall advantage to the use of DST for EXTOK.

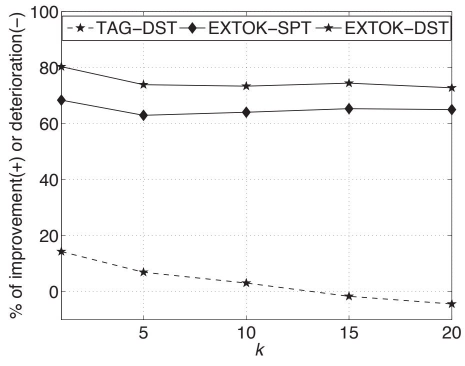


Figure 4: Relative transmission cost vs. k. Synthetic dataset.

In the second experiment we evaluate the impact of varying the nodes' radio range (Figure 5). Note that changing ω is bound to change the node degree (number of neighbors) of the sensors as well thus we also show the average node degree of the sensors. Clearly EXTOK-DST is the best option. The reason for the improved performance with the increased ω is that as ω increases the underlying tree becomes shorter, decreasing the number of hops to the root. Specific to the DST the size of the clusters increases, which results in a more "leafy" structure. It also means that there are increased number of non-dominating nodes (leaves) at the cost of decreasing the number of dominating nodes (non-leaves) in the tree, which are mainly responsible for relaying messages in the tree. That results in efficient root-to-nodes communication (for threshold updates and validation query

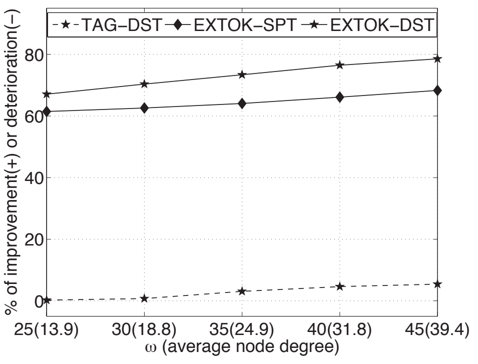


Figure 5: Relative transmission cost vs. ω. Synthetic dataset.

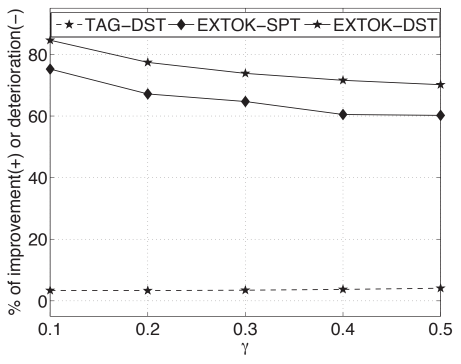


Figure 6: Relative transmission cost vs. γ. Synthetic dataset.

dissemination).

Varying γ and δ create a scenario that allows observing how the dynamics of the observed values affect the algorithms' performance. Naturally, the more dynamic the observed values, the more updates will be required. In essence this situation creates more communication traffic in the tree for EXTOK. Our experiments in this regard are summarized in Figures 6 and 7. Again EXTOK using both DST or SPT clearly outperforms TAG by a substantial margin. It is clear that the increase in communication traffic impacts EXTOK performance, while TAG is virtually oblivious to the same. The reason for EXTOK's behavior is that when values are changed more dynamically filters are violated more frequently, and more communication takes place between the root and nodes. That results in overall increase in the transmission cost of EXTOK. Nonetheless both implementations of EXTOK saved at least 60% over TAG.

In the next experiment we vary the number of nodes deployed (Figure 8). As the number of nodes are increased the relative performance of EXTOK, particularly on DST, improved. Increasing the number of nodes allows for a more effective DST structure as it implicitly increases the density of the nodes, thus increasing the relative performance of EXTOK-DST even further.

Results from our experiments using the Intel dataset are discussed next. The impact of varying k is summarized in Figure 9. As before EXTOK outperforms TAG regardless of the topology used. However, as the value of k increases

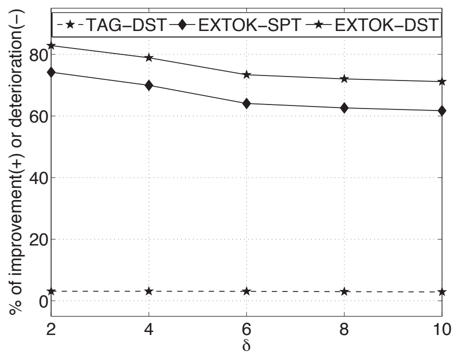


Figure 7: Relative transmission cost vs. δ. Synthetic dataset.

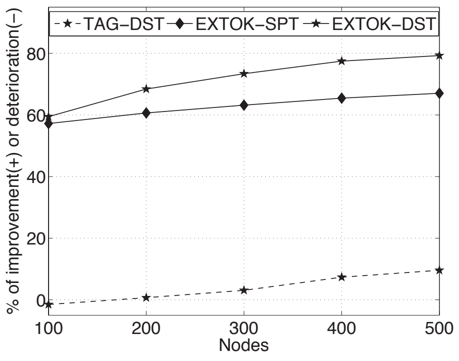


Figure 8: Relative transmission cost vs. N. Synthetic dataset.

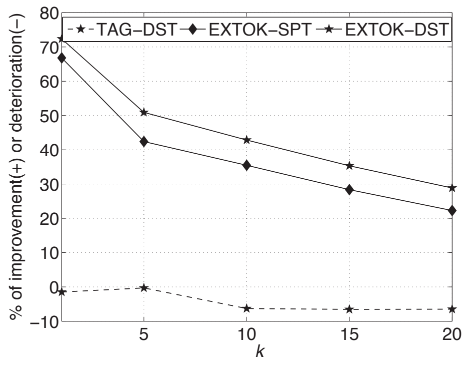


Figure 9: Relative transmission cost vs. k. Intel dataset.

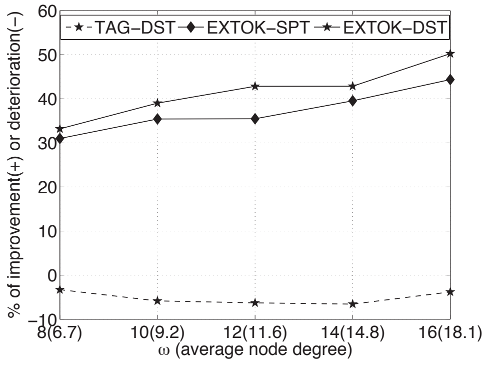


Figure 10: Relative transmission cost vs. ω. Intel dataset.

the performance gain with EXTOK decreases rather quickly. The reason is that with an increased k, the transmission cost in EXTOK increases more rapidly, because a larger k represents a larger fraction of the total number of nodes (54) are forced to perform temporal monitoring. It also means that threshold updates and query messages are used more frequently resulting in the increase of the transmission cost. In the second experiment with the Intel dataset we evaluate the impact of varying ω (Figure 10). The qualitative behavior is not very different from the case where synthetic data is used (Figure 5). Quantitatively though, there is noticeable difference, which can be explained by following two observations and their compounded effect. First, the average node degree is now smaller, which makes it harder to the DST to exhibit its advantages. Seconds, the Intel data set is more dynamic, naturally triggering more updates and consequently more nodes-to-root transmissions which are typically more expensive on the DST topology.

5.2 Energy Cost

Each bit received by a wireless transceiver incurs an energy cost, E_{rx}. It is also typical of transceivers used in wireless sensor platforms that receiving one bit requires less than the energy for transmitting one bit (E_{tx}). We will assume an ideal transmission schedule is constructed that dictates when nodes should listen, transmit, or sleep. One can readily see that a DST requires that many nodes need to receive the same transmission (a form of local multicast). Thus, while a DST may offer savings in terms of number of messages being sent, each message usually results in a higher reception cost than in the case of a SPT, because it is received by more than one node. To appreciate DST's energy cost versus that of SPT, we have performed extra experiments accounting for transmission and reception energy costs (we also assume an ideal, i.e., collision-free, schedule for medium access for both DST and SPT).

Models capturing the energy consumption have been proposed and used in various previous studies [8, 15], In order to make the presented results as technology neutral as possible, we assume that the unit of energy cost is the energy required for the transmission of a single bit, E_{tx}, and we use a parameter, R_c, to link transmission and reception cost via $R_c = E_{rx}/E_{tx}$. In our experiments R_c assumes values from the set {0.10, 0.25, 0.50, 0.75, 0.95}, and all other parameters are kept at their default values. The results capture the cost of DST's strategy which requires that multiple nodes receive the transmissions of dominating nodes.

In the case of EXTOK as the cost of reception increases the overall energy cost increases for both topologies. The reason is that while transmission cost is reduced by suppressing updates due to installed filters, the reception cost is higher as the root sends threshold updates and validation queries that are received by a large number of nodes. This can be verified from the results shown in Figure 11 in which the relative performance of EXTOK is decreasing as R_c value increases. Qualitatively similar results were obtained

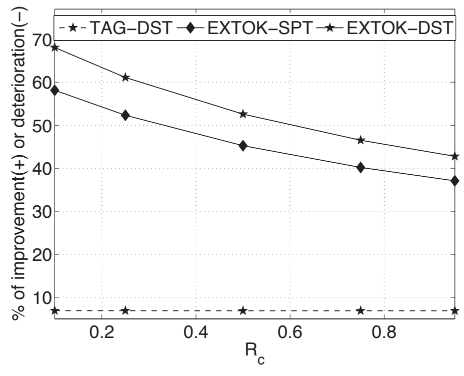


Figure 11: Relative energy cost. Synthetic dataset.

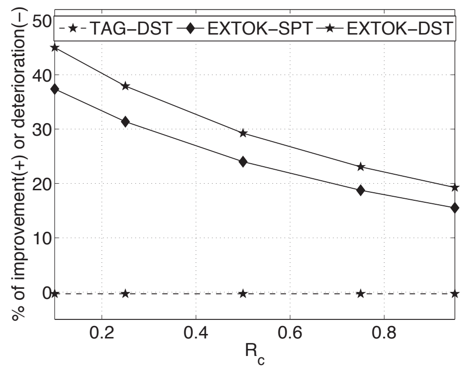


Figure 12: Relative energy cost. Intel dataset.

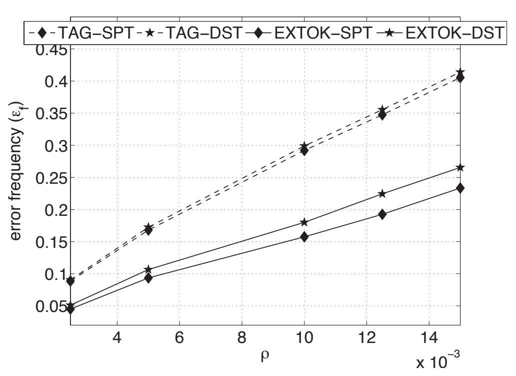


Figure 13: ϵ_f vs. ρ. Synthetic dataset.

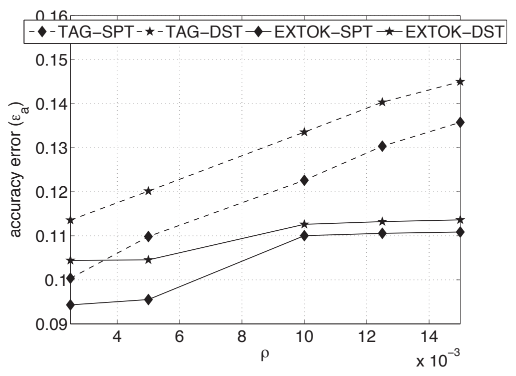


Figure 14: ϵ_a vs. ρ. Synthetic dataset.

when using the Intel dataset (Figure 12). With respect to TAG the interesting observation is that, besides being immune to changes in R_c, using DST instead of SPT again yields some gain in a random uniform topology; though admittedly small it comes at virtually no cost as the only difference is the underlying topology.

5.3 Robustness to Failures

In this section we evaluate the effectiveness of EXTOK in the presence of link failures which may cause the root to report an incorrect result. We assume the top-k result to be incorrect if one or more of the sensor *ids* is missing or its value is not correct in the reported top-k result set. Recall that $S_{p,j}$ is the set of sensors *actually* producing the p^{th} highest value, and let us denote $S'_{p,j}$ as the set of sensors that were reported as producing the p^{th} highest correct value; both with respect to the j^{th} round. Then we can define the error accuracy ratio, $\epsilon_a = |\cup_{p=1}^{k} S_{p,j} \setminus \cup_{p=1}^{k} S'_{p,j}|/|\cup_{p=1}^{k} S_{p,j}|$. To better understand the effect of failures we also compute the error frequency, ϵ_f, which is the fraction of rounds that produced an incorrect top-k outcome.

In the following experiments we set different values for the probability that a node, during a round, cannot communicate with a specific neighbor, i.e., a bidirectional *link* failure. We restrict our attention of course only to pairs of nodes (i.e., edges/links) that belong to the underlying logical tree structure. Each link fails independently of all the other links. Moreover, link failures are assumed to be independent from one round to the next. The link failure rate, ρ, impact on ϵ_a and ϵ_f is studied. (Note that, unlike before, all combinations algorithms-topologies are presented and also that lower values now denote better results.)

Results using synthetic data show that EXTOK is not only less prone to errors (Figure 13) but when they occur they do not affect the accuracy of the result as much as in TAG (Figure 14). This is explained by the smaller number of nodes that are actually sending data towards the root in EXTOK compared to TAG. Therefore a failure is less likely to affect a "sending node" and consequently less likely to affect EXTOK's results. Interestingly, for the first time in our experiments, SPT becomes a better alternative than DST for both algorithms. This is because when a failure affects a node which also happens to be a dominating node, more values are not transmitted, thus increasing the error accuracy with the increase of ρ. This is also true when messages are to be received by dominating nodes. In a sense, DST's advantage of reaching many nodes with a single transmission may become a liability for fairly large failure rates.

Results from our experiments on Intel dataset are summarized in Figures 15 and 16. Even though the TAG's error frequency decreased compared to the case of synthetic data, EXTOK's error frequency remains at least comparable with the important note that, energy-wise, it is always much less expensive as per the results discussed in previous sections. Again, and for similar reasons as before, SPT becomes a better alternative to DST. One interesting trend that can now be seen in Figure 16 is that as ρ increases ϵ_a increases a little faster than in the case of synthetic dataset. The reason for this behavior is that Intel data is relatively more dynamic. Frequent changes in sensors' value cause the sensors to vio-

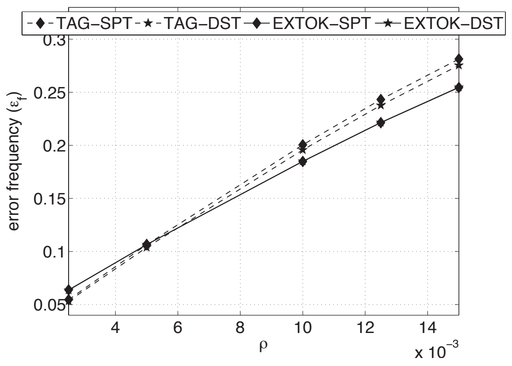


Figure 15: ϵ_f vs. ρ. Intel dataset.

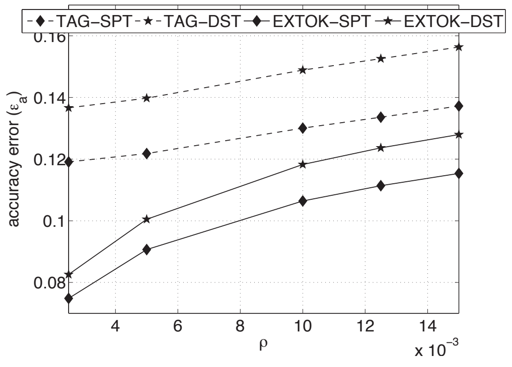


Figure 16: ϵ_a vs. ρ. Intel dataset.

late their filters more often and hence force the root to use EXTOK's Stage 2 more frequently. Recall the results from synthetic dataset where we observed the impact of the dynamism of the sensors' value on the communication traffic, which was evident from the increase in the transmission cost of EXTOK (decrease in its relative performance) as shown in Figures 6 and 7. The increased communication traffic also makes EXTOK more prone to failures not only for nodes-to-root messages, but also for root-to-node messages (which are absent in TAG). In particular when the root sends a validation query in Stage 2 F-nodes may not receive it due to failures, thus causing F-nodes that were supposed to reply back not to propagate their update, increasing EXTOK's error sensitivity more rapidly with increased ρ as shown in Figure 16.

6. CONCLUSIONS

The top-k query is important in sensor networks. Unlike previous work, we address the case where the *exact set of top-k values* are requested (along with respective node ids), regardless of how many nodes reported it. Existing solutions apply aggregation and filtering techniques for reducing communication costs. These solutions use underlying logical structures for performance gains without paying much attention to the quality of such structures. In this paper we discussed the importance of such underlying structure and proposed the use a new logical structure, DST, for processing the top-k queries in energy constrained sensor networks. Leveraging on the properties of a DST we proposed a new algorithm, EXTOK and proved its correctness. Our experiments, using real and synthetic datasets, revealed the effectiveness and superior efficiency of the combination EXTOK-DST for processing the top-k queries in sensor networks with respect to a variety of parameters.

We are currently working on exploring alternative paths on the physical topology when links failures occurs in the logical topology in order to further increase EXTOK's robustness. Another venue for further work is on whether DST can be used beneficially when processing other types of aggregate queries.

Acknowledgment

This research was partially supported by NSERC, Canada.

7. REFERENCES

[1] B. Babcock and C. Olston. Distributed top-k monitoring. *Proc. of SIGMOD*, pages 28–39, 2003.

[2] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and manets. In *Handbook of Combinatorial Optimization*, pages 329–369. 2005.

[3] Y.-H. Cho, J. Son, and Y.-D. Chung. POT: An efficient top-k monitoring method for spatially correlated sensor readings. *Proc. of the DMSN Workshop*, pages 8–13, 2008.

[4] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. of Computer and System Sciences*, 66(4):614–656, 2003.

[5] J. J. Garcia-Luna-Aceves and E. L. Madruga. The core assisted mesh protocol. *IEEE J. on Selected Areas in Communications*, 17(8):1380 – 1394, 1999.

[6] R. Laskar et al. On the algorithmic complexity of total domination. *SIAM Jour. on Algebraic and Discrete Methods*, 5(3):420–425, 1984.

[7] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE J. on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[8] S. Madden et al. TAG: a tiny aggregation service for ad-hoc sensor networks. *Proc. of OSDI*, 36:131–146, 2002.

[9] S. Madden et al. The design of an acquisitional query processor for sensor networks. *Proc. of SIGMOD*, pages 491–502, 2003.

[10] B. Malhotra, M. A. Nascimento, and I. Nikolaidis. Better tree - better fruits: Using dominating set trees for max queries. *Proc. of the DMSN Workshop*, pages 1–7, 2008.

[11] C. Olston, B. Loo, and J. Widom. Adaptive precision setting for cached approximate values. *Proc. of SIGMOD*, pages 355–366, 2001.

[12] S. Ramanathan and E. L. Lloyd. Scheduling algorithms for multihop radio networks. *IEEE/ACM Transactions on Networking*, 1(2):166–177, 1993.

[13] A. Silberstein, R. Braynard, and J. Yang. Constraint-chaining: On energy-efficient continuous monitoring in sensor networks. *Proc. of SIGMOD*, pages 157–168, 2006.

[14] A. Silberstein et al. A sampling-based approach to optimizing top-k queries in sensor networks. *Proc. of ICDE*, pages 68–78, 2006.

[15] A. Silberstein, K. Munagala, and J. Yang. Energy-efficient monitoring of extreme values in sensor networks. *Proc. of SIGMOD*, pages 169–180, 2006.

[16] P. J. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mob. Netw. Appl. J.*, 9(2):141–149, 2004.

[17] M. Wu, J. Xu, X. Tang, and W.-C. Lee. Top-k monitoring in wireless sensor networks. *TKDE*, 19(7):962–976, 2007.

[18] D. Zeinalipour-Yazti et al. The threshold join algorithm for top-k queries in distributed sensor networks. *Proc. of DMSN Workshop*, pages 61–66, 2005.

[19] Y. Zhao, L. Xu, and M. Shi. On-demand multicast routing protocol with multipoint relay (ODMRP-MPR) in mobile ad-hoc network. *Proc. of ICCT*, 2(2):1295–1300, 2003.

APPENDIX

NP-hardiness proof

Finding a Minimum Connected Dominating Set (MCDS) of a given graph is a known NP-hard problem [6]. Formally this problem is defined as following:

DEFINITION 1. *Given an undirected graph $G(V, E)$, its MCDS is a set of vertices $V' \subset V$ such that (i) V' is connected, (ii) $\forall v \in V - V'$: v' is connected to one of the vertices in V', and (iii) $|V'|$ is minimum.*

Next we examine the problem of finding a MCDS of a graph with the constraint that a given vertex must be is part of it. We call this the constrained-MCDS of a graph and define it as follows.

DEFINITION 2. *Given an undirected graph $G(V, E)$ and a particular vertex, $r \in V$, its constrained-MCDS is a set of vertices $V' \subset V$ such that (i) V' is connected, (ii) $r \in V'$, (iii) $\forall v \in V - V'$: v is connected to one of the vertices in V' and (iv) $|V'|$ is minimum.*

THEOREM 2. *The constrained-MCDS problem as defined above is NP-hard.*

Proof. Assume there exists an algorithm that takes two inputs $G(V, E)$ and $v_i \in V$ to solve the constrained-MCDS problem, and outputs the solution, V_i containing v_i. We can execute this algorithm, $|V|$ times, to find every solution with respect to every input vertex $v_i \in V$. Clearly we can check the cardinality of each output set, V_i (from the possible $|V|$ solutions) in polynomial time. If V_k is the set with the minimum cardinality, then V_k must also be the solution of the (unconstrained) MCDS problem since it does not restrict which set of vertices are selected. It means that a feasible solution of constrained-MCDS problem is also a feasible solution of the MCDS problem, which is NP-Hard. It also means that the constrained-MCDS problem is not polynomial time solvable unless P = NP, hence it is NP-Hard.

Pseudocode for EXTOK

Algorithm 1: EXTOK-Node(i : Node ID)

```
Input: $C(s_i)$, $v(s_{i,j})$, $\tau_j$, $k$
Repeat for every round begin
    /* Stage 1 : Trigger updates according to filter constraints */
    $Q = \emptyset$;
    $Flag = 0$;
    if ($State(s_i) = TN$ and $v(s_{i,j}) \neq v(s_{i,j-1})$) or ($State(s_i) = FN$ and $v(s_{i,j}) \geq \tau_j$) then
        $Flag = 1$;
        $Q = \{\langle v(s_{i,j}), s_i \rangle\}$;
    if $C(s_i) \neq \emptyset$ then
        for all $s_{i'} \in C(s_i)$ do
            if $Receive(Q', s_{i'})$ then
                $Q = Q \cup Q'$;
    if $Q \neq \emptyset$ then
        /* Apply aggregation appropriately before sending */
        $Send(Q, P(s_i))$;
    /* Stage 2 : If receive validation query, take appropriate action */
    begin
        if $Receive(ValidationQuery(v(s_{q,j}), ReqVal), P(s_i))$ then
            $Send(ValidationQuery(v(s_{q,j}), ReqVal), C(s_i))$;
        $Q = \emptyset$;
        if $State(s_i) = FN$ and $v(s_{i,j}) \geq v(s_{q,j})$ and $Flag = 0$ then
            $Q = \{\langle v(s_{i,j}), s_i \rangle\}$;
        if $C(s_i) \neq \emptyset$ then
            for all $s_{i'} \in C(s_i)$ do
                if $Receive(Q', s_{i'})$ then
                    $Q = Q \cup Q'$;
        if $Q \neq \emptyset$ then
            $Send(Q, P(s_i))$;
    end
    /* Stage 3 : Update filter setting */
    begin
        if not $Receive(\tau_{j+1}, P(s_i))$; then Set $\tau_{j+1} = \tau_j$;
        if $v(s_{i,j}) \geq \tau_{j+1}$ then
            $State(s_i) = TN$;
        else
            $State(s_i) = FN$;
        /* Update children about the new threshold */
        if ($Receive(\tau_{j+1}, P(s_i))$ and $C(s_i) \neq \emptyset$) then
            $Send(\tau_{j+1}, C(s_i))$;
    end
end
```

Algorithm 2: EXTOK-Root(i : Node ID)

```
Input: $C(s_i)$, $Q$, $S_{t,j}$, $\tau_j$, $k$
Repeat for every round begin
1       /* Stage 1 : Validate the received results */
2       $S_l = ExtractIDs(Q)$;
3       for all $s_{i'} \in S_{t,j} - S_l$; do
4           $Q = \{\langle v(s_{i',j-1}), s_{i'} \rangle\} \cup Q$;
5       $S_l = ExtractIDs(Q)$;
6       $S_{\tau+,j} = \emptyset$;
7       $S_{\tau-,j} = \emptyset$;
8       for all $s_{i'} \in S_l$ do
9           if $v(s_{i',j}) \geq \tau_j$ then
10              $S_{\tau+,j} = s_{i'} \cup S_{\tau+,j}$;
11          else
12              $S_{\tau-,j} = s_{i'} \cup S_{\tau-,j}$;
13      $k' = |V(S_{\tau+,j})|$;
14      $m = |V(S_{\tau-,j})|$;
15      /* Case 1 and Case 2 : Root has enough values that
        are above threshold */
16      if $k' \geq k$ then
17          $[V(S_{t,j}), S_{t,j}] = FindTopK(V(S_{\tau+,j}), S_{\tau+,j})$;
18      /* Case 3 : Select a probe value to inquire filtering
        nodes */
19      else
20          /* Sub-case 3.1 : Select an appropriate value
            from the given set of values */
21          if $k - k' \leq m$ then
22              $v(s_{q,j}) \longleftarrow (k-k')^{th}$ highest value of
                $V(S_{\tau-,j})$;
23          /* Sub-case 3.2 : Set the probe value to be
            minimum */
24          else
25              $v(s_{q,j}) = -\infty$;
26          /* Stage 2 : Send query and receive updates to
            validate the results */
27          $ReqVal = k - k'$;
28          $Send(ValidationQuery(v(s_{q,j}), ReqVal), C(s_i))$;
29          $Receive(Q')$; /* Received in response to the
            ValidationQuery */
30          $S_l = ExtractIDs(Q')$;
31          for all $s_{i'} \in S_{\tau+,j} \cup S_{\tau-,j}$ do
32              $S_l = s_{i'} \cup S_l$;
33          $[V(S_{t,j}), S_{t,j}] = FindTopK(V(S_l), S_l)$;
34      /* Stage 3 : Update threshold for new filter setting
        */
35      $\tau_{j+1} = FindMin\left(V(S_{t,j})\right)$;
36      if $\tau_{j+1} \neq \tau_j$ then
37          $Send(\tau_{j+1}, C(s_i))$;
38      $D_j = V(S_{t,j})$;
end
```