Trading Computation & Precision for Energy in Wireless Sensor Networks with Applications

Konstantinos Kalpakis Shilang Tang
Computer Science & Electrical Engineering Department
University of Maryland Baltimore County
Email: {kalpakis, stang2}@csee.umbc.edu

Abstract

Wireless ad hoc networks of battery–powered microsensors are proliferating rapidly and transforming how information is gathered and processed, and how we affect our environment. The limited energy of those sensors poses the challenge of using such systems in an energy efficient manner to perform various activities. In many of these activities, a basic step is that of a sensor sending a data set to a base station. Since communication drains sensor energy fast, reducing the amount of bits transmitted can lead into substantial energy savings.

We consider the problem of communicating a given set S of elements from a universe in a way that transmits substantially fewer bits, with a very small computation overhead at the sender. We present an approach that utilizes Bloom filters to construct a small footprint signature for S, from which candidates for S are computed. Our approach trades some practical computation cost and a small precision loss at the receiver (base station), which is anyway resource rich, for reducing the number of bits transmitted. The precision loss stems from the fact that the recovered set at the receiver may include false positive(s).

We experimentally show that our approach substantially reduces the number of bits transmitted, while still having practical receiver running times and few false positive errors. Moreover, we show that our approach can utilize application–dependent characteristics to further reduce false positives, or even eliminate them as in our experiments with topology gathering and energy map gathering, without noticeable increase in running times.

1 Introduction

Recent advances in sensing technologies and low–power electronics have led to the development of distributed wireless ad hoc networks of battery powered microsensors [14]. Wireless sensor networks (WSN) are to consist of thousands of such inexpensive sensors, that can be readily deployed in physical environments to collect useful information and affect their environment. Sensors typically communicate with one another over short distances and establish multi–hop communication paths to base stations. Managing WSNs is challenging due to limited energy, large number of sensor nodes, inhospitable working environments, and unpredictable deployment. Foremost among these challenges is how to best utilize the limited energy of the sensors.

Two basic approaches are used to conserve energy: limiting radio operations [4, 16], and reducing the amount of data transmitted [6, 8, 10, 11]. Reducing the amount of data transmitted is attractive, since radio communication has been regarded as a major energy consumer [12, 14]. Madden et al [12] find that Mica motes running TinyDB use 41% of their energy on communications. Pottie and Kaiser [14] report that future platforms could execute $3x10^6$ instructions to transmit 1K bits over 100 meters. Note that, currently, for the Berkeley sensor motes, it takes in the order of 1μJ to transmit a single bit, 0.5μJ to receive a bit, while executing roughly 100 instructions consumes up to 0.8μJ [9].

In this paper, we consider the problem of communicating any given set of elements from a known universe in a way that transmits substantially fewer bits with a very small computation overhead at the sender. Sending a set of elements from a sensor to a receiver is a basic operation in many activities in wireless sensor networks, therefore, achieving this goal leads to substantial energy savings for the sensors. We explore trading some practical computation cost and a small precision loss at the receiver (base station), which is anyway resource rich, for reduced amount of bits transmitted. The precision loss is stemming from the fact that the set recovered at the receiver always includes all the elements in the original set, but it may include some extra elements.

The basic idea of our approach is making use of Bloom filters [2]. Consider a sensor (sender) that wants to com-

municate a set S of elements from a universe U to the base station (receiver). [1] The sensor constructs an easily computable signature for S which it sends to the base station. The base station performs some computation with the received signature and finds a set of candidate sets for S together with an approximate set that is always a superset of S. The set signature is chosen so that it requires fewer bits to send than sending S directly. To compute the signature of a set S, the sensor sorts S and partitions it into small–size blocks. We describe two methods for such partitioning: fixed and varying block–sizes (adaptive). In adaptive partitioning, block sizes vary so that the desired block–spans are achieved. [2] Then, the sender constructs a signature that contains the block boundaries, the block–sums, a Bloom filter for the set of block–products, and the block–sizes. [3] Upon receiving its signature, the base station recovers set S by constructing a set of candidates as follows: find candidate blocks with the prescribed block–sums and block–products contained in the Bloom filter, and then using these block candidates, find set candidates that have the same signature as the one received. By taking the union of the set candidates, the base station finds an approximate set that contains S, and by intersecting all the set candidates finds those elements that must be in S. The receiver could make false positive mistakes but would never make a false negative mistake, due to the property of Bloom filters. When the base station receives signatures of multiple sets, and depending on the application, it may utilize additional information to eliminate some set candidates without introducing any false negative errors. This is the case for the two such applications we consider: topology gathering and energy map gathering. Though the computation overhead at the sender is small, the receiver may incur considerable computational costs. The receiver's computation cost is proportional to the sum of block–span$^{\text{block–size}}$ of all the blocks.

We call our approach TOE, and TOE–a when varying block–sizes are used. We experimentally evaluate the proposed algorithms through a simulation study. The experimental results demonstrate that both TOE and TOE–a achieve substantial savings in the number of bits transmitted (up to 53% in our simutations for a network with 1000 sensors), compared to sending S directly, at practical running times at the base station with few false positive errors. Moreover, for the topology and energy map gathering, we show, experimentally, that by utilizing easily derived inter-set dependencies, the number of false positives becomes zero, without noticeable increase in the running times.

Let us note here that our approach works for any (discrete) universe, and that in contrast to other Bloom filter applications, our approach also constructs candidate sets and identifies elements that must be in the set, in addition to answering membership queries.

The rest of the paper is organized as follows. Section 2 gives a brief overview of Bloom filters. We describe our approach in section 3, and present the results of our experimental evaluation in section 4. Related work is discussed in section 5, and conclusions are given in section 6.

[1] We assume, w.l.o.g., that U is a contiguous sequence of numbers.

[2] The span of a block is the number of elements from U between the block's boundaries.

[3] A block–sum (block–product) is the sum(product) of all the elements of a block.

2 Overview of Bloom Filters

Bloom [2] introduced a space–efficient probabilistic data structure, called Bloom filter, for representing a set that supports membership queries. A Bloom filter F of size m and order k is an m–bit vector together with k independent hash functions $H = \{h_1, \cdots, h_k\}$ that hash each element of a universe U to k bit positions of the bit vector (filter). Initially all its bits are 0. Filter F represents a set $S = \{x_1, x_2, \cdots, x_n\} \subseteq U$ of n elements as follows: for each element $x_i \in S$, set the bits $h_j(x_i)$ to 1, for $j = 1, 2, \ldots, k$. We call the ratio $m_o = m/n$ the filter's density. We answer a membership query $y \in S$? with yes if all the k bits determined by the k hash functions evaluated at y are set to 1 in the Bloom filter; otherwise, we answer no. We say that an element y is consistent with F, if the answer to the membership query for y is "yes". Clearly, it is possible to have false positive errors but not any false negative errors. The probability f of a false positive error is $f = (1 - (1 - 1/m)^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k$. Bloom filters have some important properties:

- there are no false negative errors.
- the false positive error rate f dependents on the filter's order k and density m/n, and it is independent of the size $|U|$ of the universe.
- the expected number of false positive errors is $f \cdot |U|$.
- for any two sets $S_1, S_2 \subseteq U$, a Bloom filter for $S_1 \cup S_2$ ($S_1 \cap S_2$) can be obtained by taking the logical OR (AND) of the Bloom filters for S_1 and S_2, provided these two filters are of the same order and size, and use the same hash functions.
- a Counting Bloom filter [7] support insertion and deletion of elements; a counting Bloom filter is obtained from a standard Bloom filter by replacing each bit with a counter (and incrementing/decrementing those counters instead of setting/resetting the corresponding bits).

3 Trading cOmputation for Energy

We describe two algorithms, TOE and adaptive TOE (TOE–a), for trading computation for energy when communicating sets between sensors and base stations. The primary difference between the two algorithms is on how a set is partitioned into blocks.

3.1 TOE

The TOE algorithm uses the following system–wide parameters: a universe U of elements, a block size p, and a Bloom filter density m_o (e.g. number of bits per block).

Whenever a sensor wants to send set $S \subseteq U$ of n elements, it sorts S in increasing order, and partitions it into blocks of equal size p. The sensor then computes a signature for S that consists of the block boundaries $\underline{d}$ [4], the block sums $\underline{\sigma}$, the block sizes $\underline{\omega}$, and a Bloom filter for the set of block products. It then sends the signature of S, while omitting the sizes of all the blocks but of the last one, to the base station. The complete sender–side algorithm is given in Figure 1.

```
SendSet(S, p, m_o)
1   let the set S in increasing order be S = {s_1, s_2, ..., s_n}
2   let (ω, σ, d, F) ⟵ ComputeSignature(S, p, m_o)
3   send message msg = (σ, d, ω_b, F) to base station

ComputeSignature (S, p, m_o)
1   let (d, b) ⟵ GetBlockBoundary(S, p)
2   for i ⟵ 1 to b do
3       B_i ⟵ {s_j ∈ S | d_i < s_j < d_{i+1} }
4       ω_i ⟵ |B_i|
5       σ_i ⟵ Σ_{x∈B_i} x
6       π_i ⟵ Π_{x∈B_i} x
7   let ω ⟵ {ω_i}, σ ⟵ {σ_i}, and π ⟵ {π_i}
8   let F ⟵ an initially empty Bloom filter with length m = b · m_o bits
9   Hash π to F
10  return (ω, σ, d, F)

GetBlockBoundary (S, p)
1   let n ⟵ |S|, b ⟵ 0, and i ⟵ 1
2   let d be {s_1}
3   while i < n do
4       if (i + p + 1) ≤ n then
5           b ⟵ b + 1
6           i ⟵ i + p + 1
7           append s_i to the end of d
8       else if (n − i) ≤ 2 then
9           append {s_j ∈ S | i < j ≤ n} to the end of d
10          break
11      else
12          b ⟵ b + 1
13          append s_n to the end of d
14          break
15  return (d, b)
```

Figure 1. Sender–side TOE algorithm.

Whenever the base station receives a message with the signature of a set S, it computes a superset $\hat{S}$ of S as well as a set of candidate sets for S from that signature. First, it finds candidates for each block, of size l, by identifying all l–subsets $X \subseteq U$ that fall within the block's boundaries, have sum $\sum_{x \in X} x$ equal to that block's sum in the signature, and their product $\Pi_{x \in X} x$ is consistent with the Bloom filter in the signature. Second, using the block candidates, it identifies candidates for S as follows. Take the Cartesian product C_o of all the block candidates, remove from C_o all those tuples whose Bloom filter is not equal to the Bloom filter in the signature, and then create a set of candidate sets C_S by taking, for each tuple in C_o, the union of all the elements in that tuple with the block boundaries. It may happen that some additional information is available from other sources, such that the candidates for S must include (exclude) all the elements in some set S_{in} (S_{out}). Such side information, whenever available, is utilized to further eliminate candidates from C_S. Observe that the set of elements that belong in every candidate set must be in the original S; those elements are guaranteed to be in S, and are thus appended to S_{in}. The union of all the elements in the candidate sets is an approximation $\hat{S}$ to S. The algorithm returns $\hat{S}$, the set of guaranteed elements S_{in}, and the set of all candidate sets C_S. The complete receiver–side algorithm is given in Figure 2.

The running time of the sender–side TOE is $O(n \log n)$. The running–time of the receiver-side TOE depends on the block spans. The running–time of the receiver–side TOE is

$$\Theta\left(\sum_{i=1}^{b} g_i^p + C_o\right) = O\left(n g_{max}^p / p + C_o\right), \qquad (1)$$

where g_i is the span of block i and $g_{max} = \max\{g_i\}$.

We compute the savings in bits that are communicated in a msg from a sender to a receiver, by sending the signature of S instead of S itself. Suppose that each element of U, as well as the block sums use no more than λ bits. The number of bits for the size of the last block, which is included in the msg, is $\lceil \log_2 (p-1) \rceil$. Consequently, the savings in bits communicated is

$$\lambda(n - |\underline{d}| - |\underline{\sigma}|) - |\underline{\sigma}| \cdot m_o - \lceil \log_2 (p-1) \rceil, \qquad (2)$$

when using a Bloom filter with density m_o.

3.2 Adaptive TOE (TOE–a)

To further decrease the amount of bits communicated, one may attempt to increase the size of the blocks a set S is partitioned to. Unfortunately, increasing the size of a block may lead to an increase of the block span as well as an increase to the number of block candidates, which in turn may adversely affect the running time of the receiver–side of TOE (see Eq. (1)). Blindly increasing block sizes

[4] The symbol $\underline{x}$ denotes a sequence $x_1, x_2, \ldots$.

ReceiveSet(msg, p, m_o, S_{in}, S_{out}, U)
1 let msg be $(\underline{\sigma}, \underline{d}, \omega_b, F)$
2 let $b \longleftarrow |\underline{\sigma}|$
3 **for** i from 1 to b **do**
4 let $C_i \longleftarrow \emptyset$
5 **if** $i = b$ **then**
6 $C_i \longleftarrow GetBlockCandidates(\sigma_i, d_i, d_{i+1}, \omega_b, F, U)$
7 **else**
8 $C_i \longleftarrow GetBlockCandidates(\sigma_i, d_i, d_{i+1}, p, F, U)$
9 let $\underline{C} \longleftarrow \{C_i \mid i = 1, \ldots, b\}$
10 let $(C_S, S_{in}) \longleftarrow GetSetCandidates(\underline{C}, F, \underline{d}, S_{in}, S_{out})$
11 $\hat{S} \longleftarrow \{\cup c_j \mid c_j \in C_S\}$
12 **return** $(S_{in}, \hat{S}, C_S)$

GetBlockCandidates($\sigma, low, high, k, F, U$)
1 let $\widehat{U} \longleftarrow \{u_j \in U \mid low < u_j < high\}$
2 let $C \longleftarrow \emptyset$
3 **foreach** k–subset X of $\widehat{U}$
4 let $\pi \longleftarrow \Pi_{i=1}^{k} x_i$
5 **if** $\sum_{i=1}^{k} x_i = \sigma$ and π is consistent with F **then**
6 add X to C
7 **return** C

GetSetCandidates($\underline{C}$, F, $\underline{d}$, S_{in}, S_{out})
1 let m be the number of bits of the Bloom filter F and b be $|\underline{C}|$
2 let $C_o \longleftarrow C_1 \times C_2 \times \cdots \times C_b$
3 **foreach** tuple $\tau = (c_j \mid j = 1 \ldots b) \in C_o$ of b block candidates **do**
4 let $\underline{\pi} \longleftarrow \left\{ \Pi_{x \in c_j} x \mid j = 1, \ldots, b \right\}$
5 let F' be a Bloom filter of size m for $\underline{\pi}$
6 **if** $F' \neq F$ **then**
7 remove τ from C_o
8 let $C_S \longleftarrow \emptyset$
9 **foreach** tuple $\tau = (c_j \mid j = 1 \ldots b) \in C_o$ **do**
10 let c be the set of the elements in $\underline{d}$ and in τ
11 add c to C_S
12 **while** C_S changes **do**
13 $C_S \longleftarrow EliminateSetCandidates(C_S, S_{in}, S_{out})$
14 $S_{in} \longleftarrow S_{in} \cup \{\cap c_j \mid c_j \in C_S\}$
15 **return** (C_S, S_{in})

EliminateSetCandidates(C_S, S_{in}, S_{out})
1 **foreach** $c \in C_S$ **do**
2 **if** $\neg (S_{in} \subseteq c \wedge S_{out} \cap c = \emptyset)$ **then**
3 remove c from C_S
4 **return** C_S

Figure 2. Receiver–side TOE algorithm.

SendSetAdaptive(S, p, Ψ, m_o, U)
1 let the set S in increasing order be $S = \{s_1, s_2, \ldots, s_n\}$
2 let $(\underline{\omega}, \underline{\sigma}, \underline{d}, F) \longleftarrow$
$ComputeSignatureAdaptive(S, p, \Psi, m_o, U)$
3 send message $msg = (\underline{\sigma}, \underline{d}, \underline{\omega}, F)$ to base station

ComputeSignatureAdaptive (S, p, Ψ, m_o, U)
1 let $(\underline{d}, b) \longleftarrow GetBlockBoundaryAdaptive(S, p, \Psi, U)$
rest of code is identical to lines 2–10 of ComputeSignature procedure of Figure 1

GetBlockBoundaryAdaptive (S, p, Ψ, U)
1 let $n \longleftarrow |S|$, $b \longleftarrow 0$, and $i \longleftarrow 1$
2 let $\underline{d}$ be $\{s_1\}$
3 **while** $i < n$ **do**
4 **if** $(i + 3) \leq n$ **then**
5 pick biggest j that $j \leq p \ \wedge \ (i + j + 1) \leq n \wedge$
$\wedge \ GetSpan(s_i, s_{i+j+1}, U) \leq \psi_j$
6 $b \longleftarrow b + 1$
7 $i \longleftarrow i + j + 1$
8 append s_i to the end of $\underline{d}$
9 **else**
10 append $\{s_j \in S \mid i < j \leq n\}$ to the end of $\underline{d}$
11 **break**
12 **return** $(\underline{d}, b)$

GetSpan(low, $high$, U)
1 let $\widehat{U} \longleftarrow \{u_j \in U \mid low < u_j < high\}$
2 **return** $|\widehat{U}|$

ReceiveSetAdaptive(msg, m_o, S_{in}, S_{out}, U)
1 let msg be $(\underline{\sigma}, \underline{d}, \underline{\omega}, F)$
2 let $b \longleftarrow |\underline{\sigma}|$
3 **for** i from 1 to b **do**
4 let $C_i \longleftarrow GetBlockCandidates(\sigma_i, d_i, d_{i+1}, \omega_i, F, U)$
rest of code is identical to lines 9–12 in the ReceiveSet procedure of Figure 2

Figure 3. Adaptive TOE (TOE-a).

to achieve better bit savings will soon lead to unacceptable running times.

To mitigate this issue, we describe the adaptive TOE (TOE–a). The parameter p for TOE–a now becomes the maximum block size permitted. TOE–a uses an additional system parameter $\Psi = \{\psi_2, \psi_3, \cdots, \psi_p\}$, which specifies an upper bound ψ_i on the span of blocks of size i, for $i = 2, 3, \ldots, p$. These upper bounds on the block spans are used when a set is greedily partitioned into blocks of size j as big as possible, provided $j \leq p$ and the block–span is $\leq \psi_j$. In addition, the sender in TOE–a needs to include the sizes of all the blocks, not just the size of the last block. These are the essential differences between TOE and TOE–a. The details of TOE–a are given in Figure 3.

The bit–savings of TOE–a over the naive method of sending S iteself is

$$\lambda(n - |\underline{d}| - |\underline{\sigma}|) - |\underline{\sigma}| \cdot (m_o + \lceil \log_2 (p - 1) \rceil). \qquad (3)$$

Even though TOE–a may give less bit–savings with respect to TOE, our experimental evidence suggests that for highly dynamic sets TOE–a offers a better balance among bit–savings, efficiency, and effectiveness (e.g. false positive errors).

4 Experimental Evaluation

We describe two kinds of experiments: experiments with individual sets to assess performance on recovering individual sets, and application experiments, where multiple sets are sent/recovered in the context of two applications: gathering the topology and the energy map of an ad hoc sensor

network of realistic sizes. [5] The experiments with individual sets allow us to analyze and compare performance of the TOE and TOE–a. The application experiments demonstrate that the proposed algorithms, used together with some application characteristics (e.g. topology is symmetric), provide substantial energy–savings without any false positive errors.

We assume networks that are ad hoc, symmetric, and have 1000 sensor nodes. Each sensor has an unique identifier (ID). [6] We use the following performance metrics: bit–savings, match effectiveness, and match efficiency. The bit–savings is the difference between the number of bits sent by TOE and TOE–a vs. sending the set directly, not counting packet overhead. Including packet overhead, due to packet headers, increases the benefit of our algorithms even more. [7] Match efficiency is measured by the running–time to complete execution of the $ReceiveSet$ procedure at the receiver (base station). Match effectiveness is evaluated with two measures: the percentage of the sets for which the $ReceiveSet$ procedure returns only one candidate set (i.e. there are no false positives), and the percentage of elements identified as guaranteed in the recovered set when false positives are present. All the simulations were done using Java and run on a Gateway GT5012 PC. For symmetry with TOE–a, hereafter, we refer to the TOE algorithm as TOE–f ("f" for fixed block sizes).

4.1 Set Experiments

We describe two experiments: one for comparing the performance of TOE-f and TOE–a, and one for analyzing the adaptivity of TOE–a.

For each experiment, we generate random sets of nodes over the whole network with size varying from 10 to 40, and transmit their IDs to base station. For each set size, we generate 100 sets and compute the average of the performance metrics over these sets. The density of the Bloom filters is $m_o = 12$ and the block span–size thresholds for TOE–a are $\Psi = (\psi_2, \psi_3, \psi_4, \psi_5) = (1000, 500, 100, 50)$.

In the experiment for comparing TOE–f and TOE–a, we fix the span of all the generated sets at 200. The span of a set is defined to be the number of elements of the universe between the smallest and largest elements of the set. The performance metrics (bit–savings, effectiveness, and efficiency) of TOE–f and TOE–a are shown in Figure 4. Compared to TOE-f with block size $p = 5$, TOE-a provides less bit–savings but much better match effectiveness and match efficiency. Compared to TOE-f of smaller block size, TOE-a achieves better bit-savings yet comparable match effectiveness and match efficiency. Overall, TOE-a provides a very good balance among the three performance metrics.

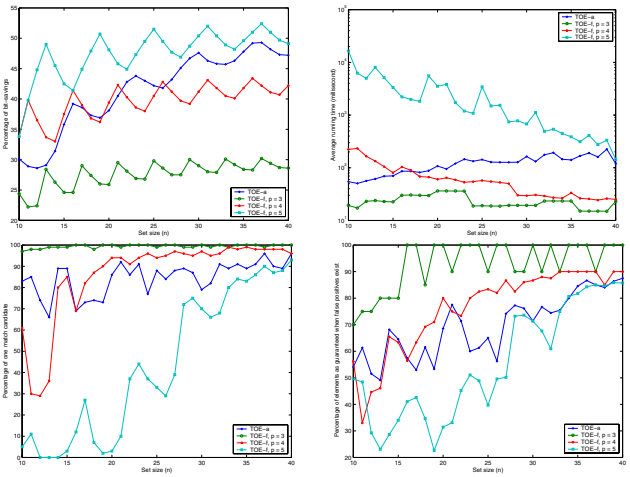

Figure 4. Performance metrics for TOE-f vs. TOE-a.

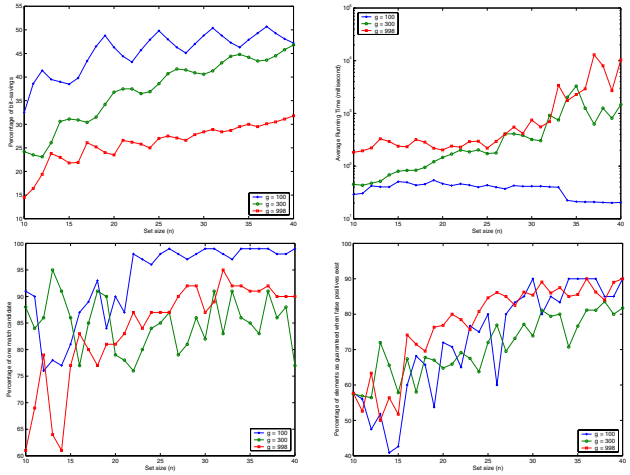

Figure 5. Adaptivity of TOE-a.

In the experiment for the adaptivity of TOE–a, for each set size, we generate 100 sets with span g of 100, 300, and 998. The results are shown in Figure 5. As expected, the span g of a set has significant impact on the performance of TOE–a, but TOE-a adapts very well on it – even for the extreme case of $g = 998$.

[5] Due to space limitations, experiments with sets with a single block level are omitted. These experiments illustrate how the various parameters impact the performance, and are used for choosing the parameter values for the experiments reported.

[6] We experimentally find that using consecutive prime numbers as IDs leads to better selectivity by $\underline{\sigma}$ in line 5 of the $GetBlockCandidate$ procedure.

[7] For example, TinyOS has packet header of 7 bytes for a data payload of up to 29 bytes per packet.

4.2 Application Experiments

In many applications, multiple sets are communicated to base station where candidates for those sets are being computed. Since those sets may be related and provide us additional information, it is interesting to investigate how such inter–set dependencies may be utilized, especially for eliminating false positives. To this end, we consider two applications: gathering the topology and gathering an energy map of a sensor network.

For the topology gathering application, each sensor sends to the base station the set of its immediate neighbors, and the base station attempts to construct the complete topology of the network. Topology gathering may be initiated on demand or periodically as needed, and the data communicated is appropriately timestamped. Symmetric topologies afford us simple inter–set dependencies for eliminating match candidates: at base station, when constructing the complete topology, if v is guaranteed to be u's neighbor then so is u for v. By incorporating this simple test in the candidate elimination procedure of the receiver, and applying it in an iterative manner until no candidates are eliminated anymore, we expect that the number of false positive errors will be decreasing. Note that applying this test does not introduce any false negative errors. Clearly, in this case, we need to carefully consider the number of iterations until no further candidate eliminations are obtained.

We consider neighborhoods with span bounded by 100, 200, 300, and 998. For each neighborhood span, we generate 5 networks of 1000 nodes with neighborhood sizes between 15 and 35. We then, compute average performance metrics. The results are shown in Table 1. The table shows the bit–savings, the one candidate set match effectiveness (percentage of neighborhoods with only one candidate) before applying any inter–set dependency on candidate elimination (1–match (before)), the number of iterations $Iter$ for the candidate elimination utilizing inter–set dependencies, the one candidate set match effectiveness after applying the candidate elimination (1–match (after)), and the total running time T in seconds. Note that it took one iteration over all the neighborhoods to eliminate all possible false positives. Thus, TOE–a is a promising method to gather network topology at substantially lower communication costs with very few if any false positive errors.

The second application we consider is that of gathering an energy map of the sensor network. In this application, each sensor is labeled by an e-label chosen among a small set of energy labels $E = \{e_1, e_2, \ldots\}$. Energy labels are used to indicate the amount of energy available (or consumed) by a sensor, and may be chosen in various ways. Sensors are grouped into disjoint clusters of nodes, and each cluster has a head that may change over time. Each cluster head constructs, for each energy label $e \in E$, the set of nodes S_e in its cluster that have label e and communicates S_e to the base station if $S_e \neq \emptyset$. The base station gathers all these energy sets, and then creates a map that provides the energy label of each sensor in the network. An energy map may be subsequently used by numerous other applications, such as scheduling, network management, data migration, etc.

Table 1. Topology and energy map gathering.

Topology gathering using TOE–a					
Span	Bit-savings	1–match (before)	$Iter$	1–match (after)	T (secs)
≤ 100	45.0%	88.9%	1	100%	78.6
≤ 200	38.5%	82.1%	1	100%	189.5
≤ 300	33.8%	84.4%	1	100%	435.7
≤ 998	25.1%	83.7%	1	100%	329.6

Energy Map gathering using TOE–a						
#Clusters	#e-Labels	Bit-savings	1–match (before)	$Iter$	1–match (after)	T (secs)
5	5	48.6%	88.8%	1.0	100%	3.6
5	10	41.0%	86.0%	1.0	100%	6.1
10	5	45.5%	91.2%	1.0	100%	3.4
10	10	35.8%	63.2%	1.8	100%	4.6

Suppose that TOE–a is used to communicate the signatures of energy sets of the clusters and at the base station, the energy sets are recovered. Observe that each node can have only one energy label. Thus, the base station, when computing candidates for the energy sets, can make use of the following simple test to eliminate candidate(s): if a sensor is found to be a guaranteed member of one energy set candidate, then it can *not* belong to any other energy set. [8] This test can be used iteratively to further eliminate set candidates for all the energy sets. In doing so, no false negative errors are introduced.

For the experiments with energy map gathering, we cluster the nodes into either 5 or 10 clusters, and randomly label the sensors with either 5 or 10 energy labels. The nodes of each cluster are then assigned consecutive IDs. We generate 5 networks of 1000 nodes for each combination of (#clusters, #e-labels) and then compute average performance metrics. The results are shown in Table 1. The experiments suggest that TOE–a coupled with the simple candidate elimination test is an effective approach to significantly reduce false positives, or even eliminate them as in our experiments, while still enjoying the substantial bit–savings offered by TOE–a.

For the experiments reported here, the running times at the receiver side of TOE–a are quite practical. For larger networks, since the computation of set candidates can be easily parallelized/distributed, the running times can still be kept practical.

[8] Note also that if a sensor is found to belong in only one set candidate with label e, among all candidates for the cluster across all labels, then it is guaranteed to be in S_e.

5 Related work

Bloom filters have been widely utilized in a variety of applications, since they offer a succinct and space–efficient way to represent a set of items and answer membership queries fast with some false positive errors. Broder and Mitzenmacher [3], have an excellent survey of applications of Bloom filters. The interested reader is referred to [3] and references therein for more details on various Bloom filter applications. Below, we focus on recent applications of Bloom filters in sensor networks.

Abadi et al [1] present algorithms for evaluating join queries over static data tables in sensor networks in an energy efficient manner. They use Bloom filters to disseminate the static data tables to every node, and to allow nodes to avoid transmitting sensor data tuples that would not join with any entries in the join table. Ye et al [15] use Bloom filters in their Statistical En-route Filtering (SEF) system that filters out false reports injected by attackers as they were forwarded toward the data collection point. To minimize the communication overhead and the corresponding energy consumption, they use Bloom filters to represent the message authentication codes (MACs) while retaining en-route verification of the MACs. Both works are concerned with energy conservation, and use Bloom filters for representing data sets and answering membership queries. Our approach, though similar to these in origin, is novel and differs from them in that we construct list of candidates for the set represented by Bloom filters.

Deb et al [5] describe a topology gathering approach, STEM, which is a distributed parameterized algorithm for extracting topology at multiple resolutions, which makes trade-off between topology details and resource expended to gather it. Our approach can be used in STEM to reduce the communication cost/energy in gathering the network topology at each resolution.

Zhao et al [17] describe an in-network aggregation approach for collecting abstracted residual energy scan (eScan). An eScan depicts an aggregated picture of the remaining energy levels for different regions in a sensor field, and provides an abstracted view of energy resource distribution. A composite scan is built by combining local scans piecewise in-network, and at each step of aggregation, these partial scans are auto-scaled by varying their resolutions. In order to do in-network aggregation, each node is assumed to have location information, and the network is expected to have spatial locality on energy spending. Our approach for gathering energy information is different from eScan in that our approach provides detailed energy information of each node, doesn't depend on sensor node location information nor spatial locality on energy spending, and does the computation at the base station, which is resource-rich.

Mini et al [13] describe a prediction–based approach for constructing an energy map of a WSN. An energy dissipation model is proposed and is used to simulate the energy spending behavior of sensor nodes. Certain amount of information is still gathered from sensor nodes, and clearly the effectiveness of this method is dependent on the prediction accuracy. When the network is dynamic and has multiple on-going tasks, sensors may have to switch from different energy dissipation models and may spend different amount of energy during different periods within the same model. This leads to a complex prediction task with an adverse impact on accuracy. In contrast, our approach for gathering an energy map, provides almost real–time energy information of the network, and is less sensitive to dynamic changes in the network.

6 Conclusions

We consider the problem of energy efficiently transmitting a given set of elements from a known universe from a sensor to a base station in wireless sensor networks. Sending a data set is a basic step in many applications.

We present novel methods, TOE and adaptive TOE, for energy conserving data communication from sensors to base stations. Our approach reduces the number of bits transmitted by trading precision and computation at the resource-rich base stations. It can achieve significant bit-savings and lower bandwidth consumption, compared to direct transmission of a set, and can be easily used by various applications in large-scale networks such as gathering network topology and energy maps.

Our proposed methods use Bloom filters together with some easily derived additional information to construct small-footprint signatures for the set of elements. In achieving a small–footprint signature, some false positive errors may be introduced. Our methods compute a set of candidates for a given signature of a set S, identify elements that must be in S, and construct a superset of S. In addition, they still support efficient membership queries. Moreover, our methods can use additional application–dependent characteristics to further reduce, or even eliminate, the false positive errors without noticeable additional computation overhead.

We perform experiments to analyze the performance of the proposed methods and demonstrate their utility, effectiveness, and efficiency in two applications: topology and energy map gathering in wireless sensor networks.

References

[1] D. Abadi, S. Madden, and W. Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. In *Proceedings of the 31th VLDB Conferences*, Trondheim, Norway, 2005.

[2] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *CACM*, 13(7):422–426, 1970.

[3] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A surevy. In *Proceedings of 40th Annual Allerton Conference*, Oct. 2002.

[4] A. Cerpa and D. Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *Proceedings of the IEEE Infocom*, New York, June 2002.

[5] B. Deb, S. Bhatnagar, and B. Nath. Multi-resolution state retrieval in sensor networks. In *Proceedings of the IEEE ICC Workshop on Sensor Network Protocols and Applications*, Anchorage, AK, May 2003.

[6] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the 30th VLDB Conference*, Toronto, 2004.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.

[8] H. Gupta, V. Navda, S. R. Das, and V. Chowdhary. Efficient gathering of correlated data in sensor networks. In *MobiHoc'05*, Urbana-Champaign, Illinois, May 2005.

[9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System archtecture directions for networked sensors. *ACM SIGPLAN Notices*, 35:93–104, 2000.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Network (Mobicom)*, Boston, August 2000.

[11] K. Kalpakis, V. Puttagunta, and P. Namjoshi. Accuracy vs. lifetime: Linear sketches for approximate aggregate range queries in sensor networks. In *Proceedings of DIAL–POMC 2004*, 2004.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: The design of an acquisitional query processor for sensor networks. In *SIGMOD'03*, 2003.

[13] R. A. Mini, B. Nath, and A. A. Loureiro. Prediction-based approaches to construct the energy map for wireless sensor networks. In *21 Simpsio Brasileiro de Redes de Computadores*, Natal, RN, Brasil, May 2003.

[14] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[15] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical en-route filtering in largescale sensor networks. In *SenSys'03*, Los Angeles, CA, Nov. 2003.

[16] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocal for wireless sensor networks. In *Proceedings of the IEEE Infocom*, New York, June 2002.

[17] J. Zhao, R. Govindan, and D. Estrin. Residual energy scan for monitoring sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, March 2002.