Singlehop Collaborative Feedback Primitives for Threshold Querying in Wireless Sensor Networks

Murat Demirbas Serafettin Tasci Hanifi Gunes Atri Rudra
Department of Computer Science and Engineering
University at Buffalo, SUNY, Buffalo, NY, 14260
{ demirbas | serafett | hanifigu | atri }@buffalo.edu

***Abstract*—In wireless sensor network (WSN) deployments, Receiver-side Collision Detection (RCD) has been proposed for speeding up collaborative feedback collection from a singlehop neighborhood. Using RCD, an initiator node can query the existence of a predicate P in its neighborhood in constant time by making all P-*positive* nodes answer simultaneously. Despite the collisions, the initiator is still able to infer useful information from a broadcast using RCD: an activity in the network means the predicate P holds for at least one node while silence indicates that P does not hold at any queried node in the network.**

In this study we investigate the threshold querying problem, where the initiator has to learn whether P holds in the network for at least threshold t number of nodes in singlehop of the initiator. To answer the threshold queries in an efficient fashion, we present a number of adaptive RCD-based querying mechanisms that dynamically re-groups the queried nodes in the network. We evaluate our algorithms on a real sensor network implementation and also carry out several simulations to contrast our approach with the traditional techniques. The experiments reveal that our algorithms achieve significant time improvements in threshold queries over traditional techniques.

I. Introduction

Wireless sensor networks (WSNs) comprise of a large number of inexpensive low-power sensor nodes that are spread accross an area for the purpose of monitoring the pyhsical environment [2], [3], [22], [24]. Each sensor node is battery powered and has certain amount of processing power, short-range wireless communication capabilities and integrated sensors. The large scale deployment of sensor networks and the resource scarcity of sensor nodes motivate the need to design efficient data aggregation and in-network information processing techniques. In order to cope with the bandwidth, energy and latency concerns related to centralized approaches, in-network processing has been advocated and widely adopted [1], [9], [15], [19]. In-network information processing exploits the computation capacity of sensor nodes to process data locally close to where it originates from. This can be achieved either by (1) summarizing the data relayed towards basestation or (2) performing decisions locally to avoid contacting the basestation for each decision made.

An example of both scenarios appears in intrusion detection applications, where a node that locally detects an event, initiates a protocol to collect the feedback of its neighboring nodes to confirm the event. Each positive answer indicates the detection of a threat at the corresponding participant node. The initiator then decides on the action to perform based on the number of positive answers from the queried participants. For example, if at least a threshold t number of positive nodes report to the initiator, the initiator notifies the basestation of the threat, and otherwise it logs the event as a false-positive. In such a scenario, the initiator does not need to know the identities of the responding nodes or even the exact number of detected threats. It is only concerned about whether the number of detected threats is above the threshold value. We call this as the *threshold querying problem*.

Due to the absence of a priori knowledge of which nodes will report positive, the initiator cannot propose a reporting schedule that includes only the positive nodes [1]. In order to avoid collisions, all nodes in the network should be assigned a different time slot to reply. In the worst case, such an approach has a time complexity of $O(n)$ where n denotes the number of nodes in the network. This does not scale well as the number of nodes increase and hence it is impractical for dense networks.

An alternative solution to the threshold querying problem is to set no ordering on the reply messages and use CSMA to provide successful delivery of the messages. This solution has a time complexity of $O(x \log x)$ where x is the number of positive answers to the query. Although CSMA achieves a decent time when x is small, it has an unacceptably high time cost when as x gets larger. Moreover, CSMA is prone to message collisions and loss due to hidden terminal problem, and the situation worsens for large x. Thus, it is impossible to tell whether $x > t$ or $x < t$ holds with certainty using CSMA.

In this study, we investigate Receiver-side Collision Detection (RCD) approaches for collaborative feedback collection from singlehop. Recently Demirbas et al. proposed *pollcast* [10], an RCD based primitive for collaborative feedback collection in WSNs. Using pollcast, an initiator node can query the existence of a predicate P in its neighborhood in constant time by making all the P-*positive* nodes answer simultaneously. Despite the collisions, the initiator is still able to infer useful information from a broadcast using RCD: an activity in the network means the predicate P holds for at least one node, while silence indicates that P does not hold at any queried node. RCD based approaches, such as pollcast, are particularly well suited for applications where

[1]Nor can the positive nodes coordinate to agree on such a schedule easily.

traffic explodes in a bursty manner and a fast response is required. By performing tests on cleverly chosen groups of nodes, RCD based approaches are able to scale well with increasing number of positive nodes in the network.

More specifically, in this paper, we tackle the problem of answering threshold queries in WSNs efficiently by a variation of group testing. We propose a family of algorithms to implement a threshold querying primitive, *tcast*. We discuss the basic algorithm in Section IV. In the basic algorithm, which was derived from [4], the initiator first partitions the nodes into $2t$ equal-sized groups, and then using pollcast the initiator queries each group (bin) one after the other. Each positive node in a group sends its reply message simultaneously. In this process, in-group collision does not pose a problem since this method does not require the acquisition of messages correctly. After one such round, the initiator marks to exclude the nodes in the silent groups from the next rounds, and re-calculates the range of values x can have, and if threshold question cannot be precisely answered with the available data, it moves on to the next round to repeat the same process.

Using the above process the initiator gets a precise answer for the threshold value in $\log \frac{N}{2t}$ rounds where t is the threshold and N is the total number of participant nodes. The intuition behind this result is as follows. At a given round there are two cases to consider 1) t groups replied positively, hence threshold is reached and algorithm stops, or 2) more than t groups returned silence meaning that the initiator can exclude all the nodes in those groups in the next round (i.e., the number of nodes to be queried has at least halved in this round). Note that in the worst case the algorithm terminates when the number of nodes to be queried reduces below $2t$, and, hence, the upperbound on the rounds for $\log \frac{N}{2t}$. In Section V, we present an extended version of this algorithm, where the group number selection is performed adaptively with respect to the estimate that the initiator has about x, the number of positive nodes.

In Section VI, we adopt a probabilistic approach for threshold querying in applications where the expected number of positive answers follows a bimodal distribution. For example, in an intrusion detection WSN deployment with n nodes, we may know from the system model in advance that at any time either there is a false detection with a few positive answers ($x < t_1$) or a true detection with a significant number of positive answers ($x > t_2$), where $t_1 \ll t_2$. In this case, the probabilistic sampling approach we adopt from the data stream algorithms community [18] enables us to find the result with *high probability* using a **constant** number of queries, independent of n, x, and t.

We conducted several simulations and experiments to evaluate the effectiveness of our algorithms. Our results show that tcast primitive is especially low-cost for $x \ll t$ and $x \gg t$, and, naturally, requires more queries for $x \approx t$, where x denotes the number of positive nodes and t denotes the threshold. Since $x \approx t$ is the rare case and $x \ll t$ and $x \gg t$ are the common cases in most realistic detection applications, tcast provides an effective and efficient solution. Our results also show that while CSMA performs well for small x, tcast performs much better than CSMA as the number of positive replies increase, due to the inherent scaling problem of CSMA. We discuss these experiments in the corresponding sections in the paper.

While we present the tcast operation for the WSN domain, the tcast operation may also be useful and adopted for RFID inventory management systems due to the scalability requirements of those systems [25].

Outline of the paper. We discuss the related work in Section II. In Section III, we present the design issues behind our methods and also provide a description of the system model. Then in Section IV, we present the basic tcast algorithm along with its implementation and simulation results. In Section V, we present our adaptive bin number selection algorithm, and in Section VI we describe our probabilistic solution to threshold querying. We conclude the paper in Section VII.

II. Related Work

A. Singlehop collaborative feedback primitives

For exploiting the time advantage of simultaneous transmission while alleviating the effects of collisions Demirbas et al. proposed a two phase polling primitive *pollcast* [10] which exploits RCD. Pollcast implements RCD by using the Clear Channel Assessment (CCA) signal from the radio chip. In a later study, Dutta *et al.* [14] presented *backcast*, a three phase primitive where a poller initially broadcasts a predicate message that contains an ephemeral identifier. All nodes for which P hold, start listening on that ephemeral identifier. Then the poller multicasts a poll message to the ephemeral address specified by the predicate message, and all nodes that match the destination respond with identical hardware acknowledgments (HACKs). The poller radio receives these HACK packets to detect the existence of one or more positive answers within the queried nodes. In backcast, HACKs are identical and thus interfere non-destructively, so the radio can latch onto and decode the superposition of multiple simultaneous HACKs. The advantage of backcast is that it provides a very robust implementation of pollcast primitive in the presence of interferences. Since the initiator radio is tuned to the HACK, it is unaffected by other interference. Traffic from neighboring regions cannot trigger a false-positive on HACK, and hence backcast is suitable to do singlehop feedback collection even in noisy multihop network environments.

Both pollcast and backcast are designed for "at least one positive answer" semantics, and they do not solve the threshold querying problem directly. However, as we show in our algorithms we can build a threshold querying solution using pollcast and backcast as building blocks.

In [13], a method which is also capable of detecting the exact neighbors that participate in voting has been proposed. Their method is based on the well-known *Ortogonal Frequency Division Multiplexing* (OFDM). However, this method is designed for 802.11 type wireless networks and the complexity of the method makes it infeasible for WSN radios.

Aspnes at al. [4] study the more general problem of computing an arbitrary aggregate function f over the bits of

information in the sensor nodes and proves asymptotic bounds on the number of pollcast/backcast queries needed to compute f. Their main result is that the number of queries is very closely related to the "fatness" of the best "decision tree" for f. In this paper we only consider the case when f is the threshold function. For the threshold function, Aspnes at al. show that $O(t\log(n/t))$ queries suffice and $\Omega(t\log(n/t)/\log t)$ queries are necessary. In fact, the $O(t\log(n/t))$ bound is proved by the "2tbin algorithm" in Section IV.

Unlike the theoretical results in [4], which are asymptotic in nature and are geared towards the worst-case input, in this paper we try to achieve good bounds for every possible input. Further, we also care about the constants in the upper bounds, which [4] did not. Finally, this paper investigates practical issues in the implementation of the threshold primitive and compares and contrasts this primitive with existing alternatives.

B. WSN programming abstractions

Several programming abstractions have been proposed for WSNs [17], [26], [27]. Our tcast primitive can be employed by these programming abstractions for implementing quick and ad hoc feedback collection from singlehop. Singlehop wireless broadcast has been identified as a narrow-waist suitable for standardization efforts in the WSNs [7]. Our tcast primitive supports in-network processing, and may help boost these standardization efforts.

C. Applications

Some applications of tcast operation are false-positive suppression, clustering, and the querying of the neighborhood for debugging purposes. These uses-cases are especially prevalent in intrusion survillance applications [2], [3] for querying of the neighborhood for classification of an intruder (say as a soldier, car, or tank) by counting the detections in the neighborhood. Other applications that involve these uses-cases include pursuer-evader tracking [8] and using robots as mobile basestations in WSNs [11], [21]. Finally, tcast operation may also find use-cases in RFID inventory management systems due to the scalability requirements in those systems [25].

III. Model

In our model, one of the nodes is designated as an initiator for a threshold querying session. The initiator is not necessarily more powerful than other nodes in terms of energy, memory or computational resources. Any node can become the initiator and start a threshold querying session.

We model the threshold querying problem in a group testing framework. We have N participant nodes (excluding the initiator), where x (unknown to the initiator) of the nodes are *positive* and the remaining $N-x$ nodes are *negative* with respect to a query predicate P. The goal of the initiator is to find out in fewest number of queries whether the amount of positive nodes exceed a threshold t or not. For solving it efficiently, the initiator can divide the nodes into b bins (groups) and then query each bin with one query cost. We say that a bin is *empty* if there is no positive node in the bin, and a bin is *nonempty* if there is at least one positive node in the bin. When the initiator queries a bin, it can only learn whether the bin is empty or nonempty.

Although our querying model is the same as that of group testing [12], our objectives are different. Group testing aims to determine the positiveness or negativeness of each node, whereas in threshold querying our objective is to answer whether threshold number of positive nodes exists or not.

A. 1^+ versus 2^+ Collision Model

We consider two models in this study based on the capabilities of the radios. In 1^+ model, there is either silence or channel activity in the vote phase. If the leader node does not hear any reply to its query, it understands that no nodes hold the query. If there is an activity, it cannot identify the message and therefore cannot be sure whether there is only a single message or multiple messages. This scenario is simple and does not require the acquisition of any message and can be accomplished by just monitoring the channel activity using a method such as RSSI, CCA, or by HACKs.

In 2^+ scenario, the radio has the capability of locking to a message and receiving it correctly while omitting all other messages. In case of an activity in the channel, the initiator can get the message if there is only one reply. Moreover, due to the *capture effect* phenomena [28], there is a chance to get one of the messages when there are multiple replies with decreasing probability as the number of messages increase. This situation also has an important side effect: If there was no capture effect, when a message is received correctly in 2^+ scenario, it would be certain that there is only one replier. Thus, all nodes in that group except the replier could be excluded from the next round since it is clear that none of them has a positive answer to the query. But in the presence of capture effect, when a message is received correctly, it is not certain that it is the only positive reply. Therefore no other nodes can be excluded from the next round. The advantages of 2^+ scenario over 1^+ can be listed as follows:

- When an activity is detected but no message is received, we can conclude that at least two nodes replied.
- When a message is received correctly, since we can get its node ID, we can exclude it from the next round.

B. Network Model

We assume all nodes are within singlehop, however, the tcast operation can also be used to query a singlehop neighborhood in a multihop WSN. As we discussed in the related works section, the backcast operation is tolerant to interference from neighboring regions in that it will not have false-positives due to interference. (Since the initiator concludes a "non-empty" feedback only if it receives a HACK, the interference cannot yield a false-positive "non-empty" decision.) As a result, using backcast as a building block, tcast is also tolerant to false-postives due to interference. However, backcast (and as a result) can still be prone to false-negative decisions in a multihop environment, because due to interference from

neighboring regions the initiator may not lock onto a HACK successfully. While superposition of HACKs strengthen the signal and improve the probability of successful reception even in the presence of other interference, there are no guarantees for the absence of a false-negative. Employing another service for reducing interference from neighboring regions can help improve the situation further. Testing and evaluation of tcast in a multihop network environment with interfering traffic is part of our future work.

IV. Basic Tcast Algorithm

In this section, we give a basic tcast algorithm in Section IV-A, and a small variation of this algorithm in Section IV-B. We present simulation results for these algorithms in Section IV-C. Finally, in Section IV-D, we present experiment results from our implementation of the basic algorithm on the WSN motes, which corroborates the findings in our simulations.

A. 2tBins Algorithm

We present the 2tBins algorithm in Algorithm 1. This algorithm is the same as the algorithm we discussed in [4] with the exception that the distribution of nodes to the bins is performed randomly here whereas it was performed deterministically in [4].

In the beginning of each round, the initiator divides the sensor nodes into $2t$ equal-sized groups (bins) randomly, hence the name 2tBins. The initiator also resets $silentBins$ to 0 at the start of a round. The initiator then starts querying each group one after the other. If a group responds to the query with silence as in Line 7, the initiator knows that all nodes in this group are negative and excludes those nodes from n, the set of nodes that may potentially be positive. In this case, the initiator also increments the $silentBins$ count by 1. The initiator then checks whether it can terminate the algorithm yet. On Line 11, the initiator checks whether it has seen at least t non-empty bins in this round. Since a non-empty bin implies at least one positive node, the condition on Line 11 guarantees that the initiator can declare that the threshold is satisfied by the nodes. On Line 14, the initiator checks whether it can conclude that the threshold is impossible to satisfy and terminate the algorithm. This is clearly the case if $|n| < t$.

When the initiator completes querying all the groups in a round, it moves to the next round and repeats the same process with the remaining nodes in n. Notice that, at the end of each round, the initiator manages to halve n from the previous round: Since the algorithm did not terminate in the previous round, that implies $silentBins \geq t$, in another words at least half of the $2t$ bins queried have been detected as empty and the nodes in those bins have been disposed. In the worst case scenario, by assuming that at each round $t+1$ bins are discarded, we get an upper bound of $2t \cdot (\log \frac{N}{2t})$ queries for the query cost of the 2tBins algorithm. In the average case, the algorithm achieves much better results as we show in Section IV-C. Since in the companion theoretical work [4], we had proved that *any* algorithm needs to make $\Omega(t \cdot (\log(N/t) \cdot \log t))$ queries for solving threshold querying problem, our 2tBins algorithm is optimal up to a $\log(t)$ factor.

[1]Note that in the worst case the algorithm terminates when n reduces below $2t$, hence the upperbound on the rounds for $\log \frac{N}{2t}$.

Algorithm 1 2tBins algorithm

```
1:  Given P poll predicate, t threshold value, n set of voters
2:  ForEach round Do
3:    silentBins ← 0
4:    Group nodes in n into 2t equal-sized bins randomly
5:    ForEach group g Do
6:      Multicast the poll predicate P to group g
7:      If group g is silent Then
8:        remove nodes in g from n
9:        silentBins++
10:     EndIf
11:     If g.index − silentBins ≥ t Then
12:       Return true //threshold achieved
13:     EndIf
14:     If |n| < t Then
15:       Return false //threshold is impossible to achieve
16:     EndIf
17:   EndFor
18: EndFor
```

B. Exponential Increase Algorithm

It is easy to see that 2tBins algorithm can become wasteful for $x \ll t$. For instance, for $x = 1$ and $t = 2$ the 2tBins algorithm pays at least $2t$ querying cost in the first round, and upto a total of $2t \cdot \log N$ queries in total. To address this inefficiency with respect to small x values, we present a small variation of the 2tBins algorithm, called *exponential increase algorithm*.

We present the exponential increase algorithm in Algorithm 2. This algorithm differs from the 2tBins algorithm only in Lines 1, 4, and 18. In the first round, the initiator divides the nodes into $binNum = 2$ groups and starts by testing these two groups. This is for eliminating a large number of negative nodes quickly for improving the performance for the case $x \ll t$. At the end of each round, as shown in Line 18, the initiator doubles the $binNum$ to get the number of bins to be used in the next round. This doubling is used for increasing the number of bins quickly so that the algorithm also handles the case where $x \gg t$.

We tried some variations on the exponential increase algorithm as well. One variation was a *pause-and-continue* scheme which does not double the number of groups if a significant number of nodes are eliminated in a round, and continues with doubling if that is not the case. Another variation was to increase the number of groups in the next round to four-folds rather than two-folds of the number in the current round when all groups tested non-empty. We experimented with both of these variations in simulations extensively but neither of them gave a consistent improvement. Therefore, we excluded these

results from the paper, so in Section IV-C we discuss results for the basic exponential increase algorithm.

Algorithm 2 Exponential Increase algorithm

```
1: binNum ← 2
2: ForEach round Do
3:    silentBins ← 0
4:    Group n into binNum equal-sized bins randomly
5:    ForEach group g Do
6:       Multicast the poll predicate P to group g
7:       If group g is silent Then
8:          remove nodes in g from n
9:          silentBins++
10:      EndIf
11:      If g.index − silentBins ≥ t Then
12:         Return true //threshold achieved
13:      EndIf
14:      If |n| < t Then
15:         Return false //threshold is impossible to achieve
16:      EndIf
17:   EndFor
18:   binNum ← binNum ∗ 2
19: EndFor
```

C. Tcast Simulations

Simulation setup. We assume every node is in the single-hop neighborhood of each other. We repeated each configuration with 1000 runs and calculated the average number of queries. Note that when a bin is empty, we do not count it as a query and skip it. In the experiments it is handled by arranging the order of bins so that empty bins are at the end. Since early termination is used in all experiments these empty bins never occupy a time slot in the simulations.

We compared our tcast algorithms with two simple solutions: *CSMA* and *sequential ordering*.

In *CSMA*, we put no restriction on the reply times of the nodes. The nodes use carrier sensing and send when they sense the medium as idle. In case of a collision they use exponential backoff to calculate the next time slot to send their messages. The initiator does not need to wait to receive all the replies, the initiator declares the querying finished when it receives sufficient number of answers from the nodes to conclude either $x \geq t$ or that it is impossible achieve the threshold.

In *sequential ordering*, the initiator assigns a sequence number for all participating nodes to enable the nodes to send the reply messages without collisions. To implement sequential ordering, the initiator puts the nodes in order, calculates the reply sending times for each node and then broadcasts the schedule. The problem with this solution is the time syncronization requirement to prevent inconsistencies between the local clocks of the nodes. An alternative implementation is to make the initiator contact the next node after a successful message reception from the previous node. This alternative takes longer but avoids the time synchronization requirement. In our simulations we used the first solution which favors the sequential ordering results more.

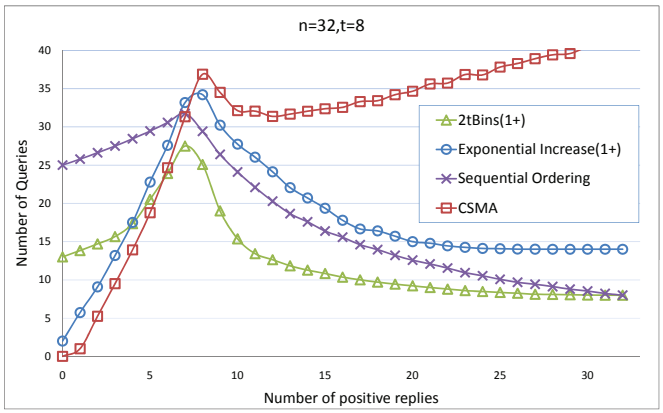

Fig. 1. Performance of tcast in 1+ scenario

1) **1+ Simulations:** Figure 1 shows the simulation results for 1^+ scenario. The figure shows that while the tcast primitive is low-cost for $x \ll t$ and $x \gg t$, for $x \approx t$ tcast requires naturally more queries. Under the observation that $x \approx t$ is the rare case and $x \ll t$ and $x \gg t$ are the common cases in most realistic intrusion detection applications, tcast provides an effective and efficient solution.

Figure 1 also shows the comparison of CSMA with the methods proposed in this study. CSMA cost increases proportional to x, and while CSMA is acceptable for $x << t$, it becomes inacceptable for $x > t$. For large n the overhead of CSMA becomes more significant.

In the comparison of the *exponential increase* scheme with the $2tBins$ scheme, we see that when $x \ll t$, *exponential increase* algorithm is more successful since it can aggressively eliminate the negative nodes. On the other hand, when $x \gg t$, exponential increase performs consistently worse than 2tBins due to the initial redundant rounds of *exponential increase* scheme in which no node is eliminated.

In all curves of Figure 1, we see that the worst case occurs when $x \approx t$. This is explained by the fact that when there are t positive nodes, it becomes harder to determine in a few queries whether it is less than or more than t. When the number of positive replies is sufficiently large, the result is found only in t queries by seeing more than t non-empty groups without the completion of a round. Thus, the required query number decreases as the number of responders approaches to n. In the other direction; when $x \ll t$, again the result is found in a few queries by observing that many bins are empty so that even if the remaining bins are full of positive nodes, $x \geq t$ is impossible. In the extreme case, when there are no positive replies, the query number becomes $(n-t)/(\frac{n}{2t})$ where $\frac{n}{2t}$ is the number of nodes in a group and $n-t$ is the required number of nodes that should be eliminated so that at most t nodes remain.

Figure 1 shows that the sequential ordering scheme starts with a large cost overhead (approximately $n-x$) for $x \ll t$.

The query cost of sequential ordering starts to become more acceptable only for $x \gg t$.

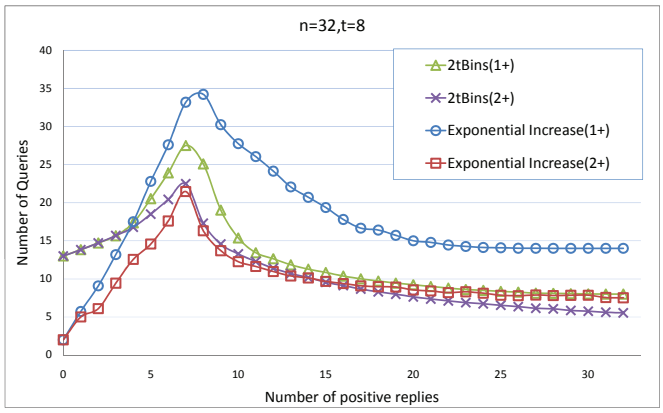

Fig. 2. Performance of tcast in 2+ scenario

2) **1+ model versus 2+ model:** As expected, Figure 2 shows that using 2^+ collision model we can achieve lower number of queries compared to the 1^+ model. This superiority results from the two advantages of 2^+ scenario over 1^+: First, in case of a collision 2^+ can deduce that at least two nodes replied in that group while 1^+ can detect the existence of one positive reply. Second, since 2^+ is capable of getting the identity of a node correctly, it is possible to exclude this node from the next round.

The superiority of 2^+ is especially evident around $x = t-1$ in the 2tBins method. This advantage comes from the fact that most bins contain exactly one positive node around this p value and therefore no in-group collision occurs in the time slots of these bins. So when we use the 2^+ scheme, we can identify and eliminate many positive nodes in the first round and then start with a very low number of bins in the second round.

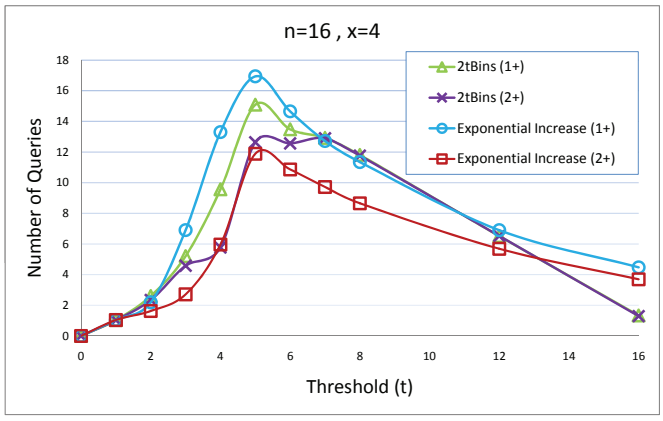

Fig. 3. Performance of tcast as threshold changes

Figure 3 shows the behaviour of tcast as the threshold t changes while the number of positive nodes x is kept constant at 4. The maximum number of queries peaks around $x = t$ and declines as t approaches to 0 or n. Moreover, the relationship between 1^+ and 2^+ is preserved for all t values supporting the fact that the latter always performs better than the former.

D. TCast Experiments on the Motes

1) Methodology: In TCast the initiator broadcasts a predicate P along with a group identifier that maps each participant node to a group, and then query each group separately. For our implementation, we use backcast [14] as it provides a robust implementation of pollcast [10]. Backcast employs an ephemeral identifier to be used for radio hardware address enabling group querying. Any broadcast to a radio address that matches the ephemeral identifier triggers a hardware acknowledgement(HACK) from the node. CC2420 radio supports two hardware addresses 16-bit and 64-bit long addresses enabling two concurrent backcasts at most. In our experiments, we employ the radio's short address to match the ephemeral identifier.

The 802.15.4 MAC exposes an acknowledgement request flag. The receiver acknowledges any incoming frame with acknowledgement request flag set if it is configured for automatic acknowledgements only when the incoming frame passes both CRC and radio's hardware address recognition checks. Importantly, the transmitted ACKs are identical for a given sequence number, which enables nondestructive superposition of simultaneous HACKs to be successfully received at the querying node [6].

2) Setup: We implement our TCast primitive in the TinyOS 2.0 embedded operating system [16]. The experiments are based on the widely-used Telos motes [20] with the CC2420 radio and IEEE 802.15.4 stack.

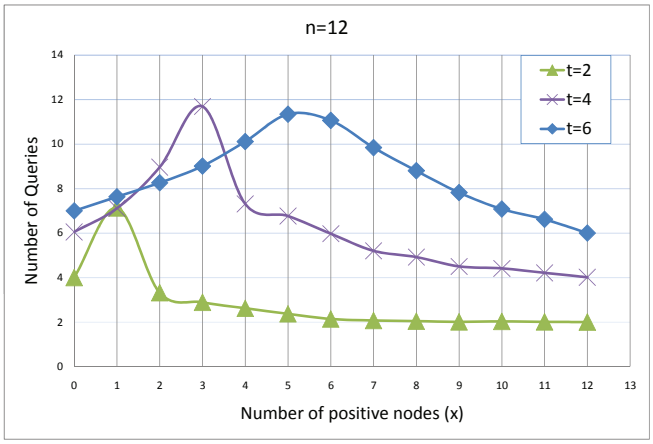

Fig. 4. Experimental results for TCast with 2tBins algorithm

Our experimental setup consists of an initiator and 12 participant TelosB motes. All motes are directly connected to a central controlling unit (in our case the laptop) via serial port interface. The initiator mote exposes *configure*, *query* and *reboot* functions via serial interface to the laptop, while the participant provides only *configure* and *reboot* procedures. The laptop first configures the motes with corresponding run

settings then stimulates the initiator to start querying the participant nodes using TCast over the radio and finally the laptop collects the query result from the initiator by invoking the corresponding procedures. We repeat each run 100 times. Each mote is rebooted between two consecutive runs in order to remove the effect of the previous run.

3) Results: The experiments capture the overall performance of TCast for the number of participant nodes is set to 12 and the threshold to 2,4,6 respectively. We implemented 2tBins algorithm as an example to demonstrate that the proposed algorithms are applicable and yield similar results to the simulations. In each scenario, the experimental results shown in figure 2 follow similar patterns to that of the outlined simulation results except the fact that we observe some false negatives due to radio irregularities. We report no false-positives runs but only 102 false-negative runs out of 7,200 separate TCast. This renders an error rate of %1.4, which is acceptable given the non-ideal radio assumptions. It is notable here that majority of the false-negatives occur when the queried group has only one positive node, a participant node with positive answer. As the number of superposing HACKs increase, the error rate slashes down.

V. Adaptive Bin Number Selection (ABNS)

The *2tBins Algorithm* and *Exponential Increase Algorithm* are not designed to adapt to x during the selection of bin numbers b. For instance, when $x = n$, the optimal b value should be t: Since all bins will be full, t queries will suffice to find the answer. On the other extreme, if $x = 0$, just one bin that spans all nodes will reveal that no nodes hold a positive answer to the query. In this section we present an adaptive bin number selection procedure based on a statistical estimate p on the expected value of x.

A. Theoretical Background

Since the initiator does not know x, the best we can do is to use p as a guess for x. Since the optimal b value is affected by p, as well as n and t, we know that b must be a function in the form:

$$b = f\left(n, t, p\right) \tag{1}$$

In order to derive f, we use the following approximate function $g(b)$ which maximizes the expected number of eliminated nodes in a query:

$$g(b) = \left(1 - \frac{1}{b}\right)^{p} \cdot \frac{n}{b} \tag{2}$$

Here, $1 - \frac{1}{b}$ is the probability of a positive node not entering a particular bin. Since there are p positive nodes, $\left(1 - \frac{1}{b}\right)^{p}$ gives the probability of a particular bin being empty. We multiply this probability with $\frac{n}{b}$, the expected size of a bin, to find a good estimate on the expected number of eliminated nodes in a query. Since we want to maximize $g(b)$, we take the derivative of (2) with respect to b and set the result to 0. The solution gives the optimal value of b.

$$\frac{\partial}{\partial b} g(b) = 0 \tag{3}$$

$$\Rightarrow \left(p \cdot \left(1 - \frac{1}{b}\right)^{p-1} \cdot \frac{n}{b^3}\right) - \left(\left(1 - \frac{1}{b}\right)^{p} \cdot \frac{n}{b^2}\right) = 0$$

$$\Rightarrow \left(1 - \frac{1}{b}\right)^{p-1} \cdot n \cdot b^{-3} \cdot (p - b + 1) = 0$$

$$\Rightarrow b = p + 1 \tag{4}$$

Equation 4 says that optimal bin number is independent from n and t and directly proportional to p. We also made simulations which yielded best results when $b = p + 1$ supporting theoretical findings. However, our simulation results showed that this equation is valid only when $p < t$. This makes sense because the optimization function which aims at maximizing the number of eliminated nodes per query is not meaningful when $p \geq t$. When $p \geq t$, the purpose becomes finding at least t non-empty bins so as to reveal that the answer to the threshold problem is positive with minimum number of queries.

Still, we have an important problem left: we do not know the real p value. p is supposed to be the estimate for x the unknown quantity of positive nodes. But we can start the simulation with an initial random or expected p value, p_0, and then update our estimate at each round. To find p_{i+1}, we can use the result of the current round. If we compare the number of empty bins, e_{real}, in the current round to the expected number of empty bins, $e_{expected}$, we can estimate p_{i+1}.

$$e_{expected} = \left(1 - \frac{1}{b}\right)^{p} \cdot b \tag{5}$$

If we set $e_{real} = e_{expected}$, we can calculate p as follows:

$$p = \frac{\log e_{real} - \log b}{\log\left(1 - \frac{1}{b}\right)} \tag{6}$$

B. ABNS Algorithm

In the previous subsection, we have derived equations to calculate b at round i using p_i (Equation 4), and p_{i+1} from e_{real} at round i (Equation 6). Using this information, we present an iterative threshold querying algorithm which tries to optimize the bin number b at each round.

Algorithm 3 ABNS Algorithm

1: e_{real} : # of empty bins in a round
2: $i \leftarrow 0$
3: $p_0 \leftarrow 2t$ //initial p estimate
4:
5: **While** threshold query is unresolved **Do**
6: $b_i = p_i + 1$
7: run tcast round with b_i bins and find new e_{real}
8: $p_{i+1} = (\log e_{real} - \log b_i)/(\log\left(1 - \frac{1}{b_i}\right))$
9: $i = i + 1$
10: **EndWhile**

In Algorithm 3, at each iteration of $while$ loop, the initiator first determines how many bins will be used in this round of T-Cast (Line 3). Then it runs that round and acquires the number of empty bins at that round. By using the result, it calculates the new p estimate in Line 3. Note that p_i is just a rough estimate on p which is utilized in solving the threshold querying faster.

C. ABNS Simulations

We performed simulations to investigate the performance of the ABNS algorithm with variations on the bin selection strategy. In the simulations, we used two different p_0 values; $p_0 = t$ and $p_0 = 2t$. We run each method 1000 times and took an average of the runs to increase precision.

To assess the performance of ABNS algorithm, we defined an oracle bin number selection algorithm assuming we have always precise knowledge of $p = x$. Oracle algorithm gives the lower bound on the number of queries in tcast. We calculate the bin numbers to be used in oracle algorithm with respect to $p = x$ values as follows:

- Firstly, we have shown in (4) that for small x values $b = x + 1$ is the optimal selection. In our preliminary experiments, we saw that this situation holds when $x \in \{0, \frac{t}{2}\}$. Therefore, in this region the bin number is chosen as $x + 1$.
- Secondly, we stated in Section IV-C that $x \approx t$ is the hardest case for threshold querying problem and $2t$ is the optimal bin number for this case.
- Finally, when $x = n$, since all nodes will answer, it is clear that only t bins will be enough to disclose that $x \geq t$.

By interpolating these three cases, for the oracle algorithm we can calculate b values that gives the lower bound on query numbers.

$$b = \begin{cases} x + 1 & \text{if } x \leq t/2 \\ 3x - t & \text{if } t/2 < x \leq t \\ t \cdot (1 + \frac{n-x}{n-t+1}) & \text{if } x > t \end{cases}$$

When we analyze Figure 5, we see that 2tBins algorithm consistently performs almost as good as oracle when $x > t/2$. Thus, we can deduce it is not easy to get an improvement over 2tBins algorithm in this region. On the other hand, when $x \leq t/2$, the gap between 2tBins and Oracle increases as p decreases. Therefore, we need a method to anticipate if $x \leq t/2$ and if it is, we must use appropriate b values.

In a WSN deployment, given historical data about previous x values, we can make an inference about the real x value and use it in the selection of p_0 in the first tcast round. If we do not have any information about whether $x \leq t/2$, then we would need to start with a small p_0 and improve it gradually at each round. While this method improves the performance when $x \leq t/2$, it will worsen the performance when $x > t$. This situation is exemplified in Figure 5: If we select $p_0 = t$ instead of $2t$, it decreases the query count at the left edge of the figure, while adding some overhead for the cases where $x \gg t$. Overall the ABNS algorithm leads to improvement because it makes the buckling pattern stand out more especially for $x \ll t$.

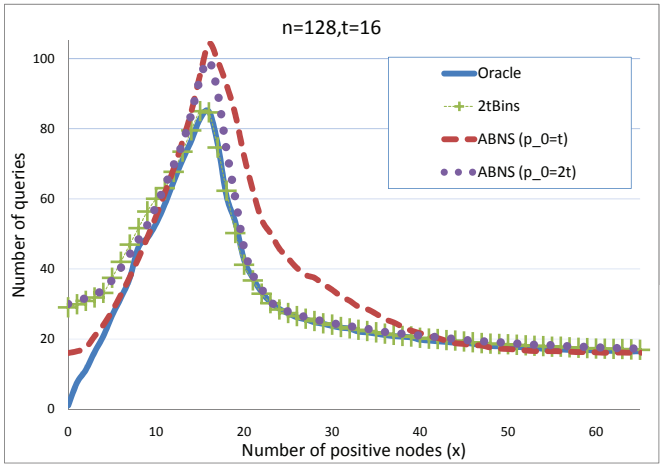

Fig. 5. Performance of Adaptive Bin Number Selection (ABNS) algorithm

D. Probabilistic ABNS

Since a better estimate on initial p value, p_0, is a key factor in improving the overall success of ABNS algorithm, we make use of the probabilistic approach here to get a better estimate on p_0. As we mentioned above the 2tBins algorithm performs almost as good as oracle when $x > t/2$, but when $x < t/2$ ABNS performs much better than 2tBins. Based on this observation, we adopt a probabilistic sampling approach from data streams algorithms [18] to estimate p_0 as follows. We use only one bin, hence one query. We put the nodes into this bin with $\frac{2}{t}$ probability. We then query this bin, which has an expected $\frac{n}{2t}$ nodes. If the bin is empty, we deduce that $p_0 < t/2$ and assign $p_0 = t/4$. Otherwise, if the bin is nonempty, we conclude that $p_0 > t/2$ and in this case we do not use the probabilistic approach: since 2tBins is consistently successful when $p_0 > t/2$, we simply switch to 2tBins method.

*Remark:*We discuss this probabilistic sampling method [18] further in the next section in the context of probabilistic querying for the case where the distribution of positive replies for a query exhibits a bimodal distribution. However, we note that in the case of probabilistic ABSN, this method still works for providing us a hint, without the assumption or requirement for bimodal distribution. The simulation results in Figure 6 show that this hint is often correct and helps to improve the performance of ABSN. *End of Remark.*

Figure 6 shows the simulation results of probabilistic ABNS algorithm. We see that the probabilistic ABNS algorithm eliminates both the cost of ABNS($p_0 = t$) when $t < x < 2t$ and the cost of ABNS($p_0 = 2t$) when $x < t/2$. Moreover, the probabilistic ABSN performs almost as good as oracle function, which can be regarded a lower bound on the required number of queries in the threshold querying problem.

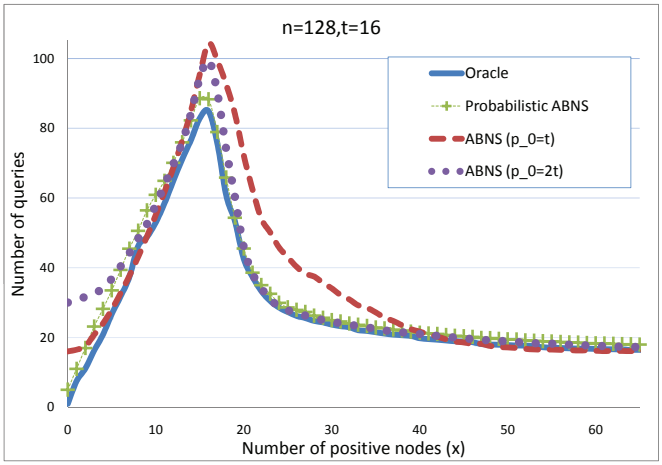

Fig. 6. Performance of the probabilistic ABNS algorithm

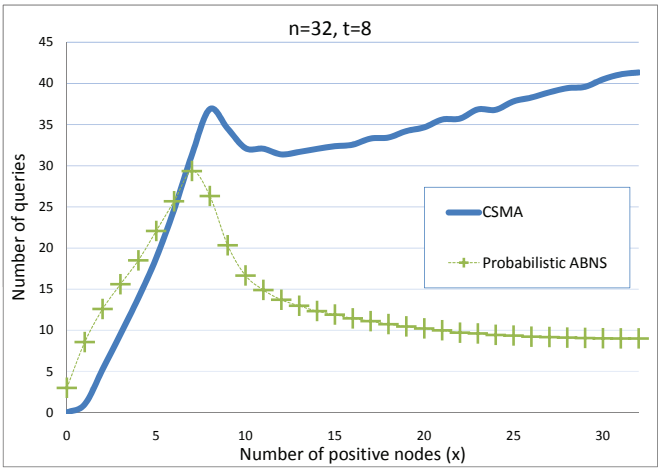

Fig. 7. Probabilistic ABNS vs. CSMA

Figure 7 shows the comparison of the probabilistic ABNS algorithm with CSMA for $N = 32$ and $t = 8$. We see that the probabilistic ABNS algorithm performs close to CSMA for $x < t$, and outperforms CSMA significantly for $x > t$.

VI. Probabilistic Model

In some WSN applications, such as intrusion detection [2], [3], [8], the distribution of positive replies for a query exhibits a bimodal distribution: If there is no activity in the network, there are only a few replies which are possibly false positives. If there is an activity, we expect a significant number of nodes to detect it. The first situation can be simulated as a normal distribution with mean μ_1 and variance σ_1^2 where $\mu_1 \approx 0$. The latter can be represented by another normal distribution around mean μ_2 such that $k \leq \mu_2 \leq n$ and variance σ_2^2. The value of k depends on the application. But most of the time $k \gg 0$ and variance of the two constituent distributions are small. Therefore, when there are x positive nodes in a query, it is possible to guess whether there is a real activity in the network or it is just a false alarm.

In addition, since the distribution of x follows a bimodal distribution with distant peaks, we can solve the threshold querying problem in this model by adopting a probabilistic sampling approach [18] as follows.

1) Put the nodes to a bin one by one with probability $\frac{1}{b}$ for each. Query this bin:
   a) If there is no answer, conclude that there is no activity in the network with a *high* probability.
   b) If there is an answer, it means there is an activity (x belongs to the $\mathcal{N}(\mu_2, \sigma_2^2)$) with a *high* probability.
2) Repeat Steps 1 until you get the desired reliability of estimate.

The main advantage of this method is that it has a time complexity of $O(1)$. In other words, the number of queries is independent from all n, t and x values. The only factor that affects the runtime is the degree of seperation of two sub-distributions which form the bimodal distribution.

On the other hand, unlike the methods that were described in Sections IV and V, this algorithm cannot guarantee the correctness of the result. But in most cases it is easy to get an error rate less than 5% with a few queries.

A. System Model

Assume we have history information which shows that most of the time either $x \leq t_l$ or $x \geq t_r$ where t_l and t_2 are the two boundary values selected as $t_l = \mu_1 + 2\sigma_1$ and $t_r = \mu_2 - 2\sigma_2$ in our simulations. We know from Section V-A that the probability of i^{th} bin b_i being non-empty is $1 - \left(1 - \frac{1}{b}\right)^x$ where recall that b is the number of bins. So we can derive the following inequalities:

$$Pr\{b_i \; is \; 1^+\} \leq 1 - \left(1 - \frac{1}{b}\right)^{t_l} \qquad \text{if } x \leq t_l \qquad (7a)$$

$$Pr\{b_i \; is \; 1^+\} \geq 1 - \left(1 - \frac{1}{b}\right)^{t_r} \qquad \text{if } x \geq t_r \qquad (7b)$$

These inequalities give the probabilities for a single bin. If we repeat the procedure r times, we find the expected number of 1^+ bins in r queries as:

$$E(1^+ \; bins) \leq r.\left(1 - \left(1 - \frac{1}{b}\right)^{t_l}\right) = m_1 \qquad \text{if } p \leq t_l \; (8a)$$

$$E(1^+ \; bins) \geq r.\left(1 - \left(1 - \frac{1}{b}\right)^{t_r}\right) = m_2 \qquad \text{if } p \leq t_r \; (8b)$$

There is a gap between the ranges of Equation 8a and 8b. If we call the size of this gap $\Delta = |m_1 - m_2|$, we can tolerate at most an error of $\varepsilon < \frac{\Delta}{2}$ (see Figure 8).

After performing r queries, if $x \leq t_l$ we expect the number of 1^+ bins to satisfy Equation 8a. However, despite the tolerance ε, the occurrence of an incorrect decision (i.e.

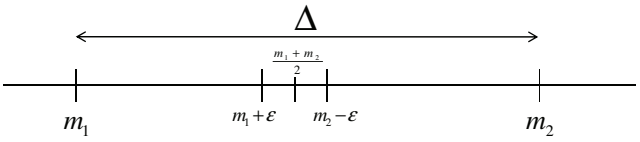

Fig. 8. Δ increases as the two sub-distributions of the bimodal x distribution move away from each other (*i.e.* m_1 moves leftwards as μ_1 decreases and m_2 moves rightwards as μ_2 increases)

system finds $x < t$ while in reality $x \geq t$) is still possible. The probability of this possibility has an upper bound given by the additive *Chernoff* inequality:

$$Pr\left[\{\#\ of\ 1^+\ bins\} > r.\left(1-\left(1-\frac{1}{b}\right)^{t_l}+\varepsilon\right)\right] \leq e^{\frac{-\varepsilon r}{2}} \tag{9}$$

Same situation holds for the case $x \geq t_r$. In Equation 9, if we want the failure probability to be less than $\delta/2$, then the right-hand side in (9) must be less than $\delta/2$ which gives the required number of repeats as:

$$r \geq f(\delta) = \frac{2\log(1/\delta)}{\varepsilon \log 2e} \tag{10}$$

Here, if we relax the upper bound for the overall failure probability, δ, the number of required repeats decreases. For example in (10) if we take $n = 128$, $\mu_1 = 16$ and $\mu_2 = 96$ when $\delta = 1\%$ we need 19 repeats, while for $\delta = 5\%$ the required number of repeats decreases to 12.

B. Probabilistic Model Simulations

We carried out a number of simulations with different r values (number of repeats) and measured the accuracy of the probabilistic model. Accuracy is the percentage of correct decisions made by the system. For example, assume $x = 3$ and $n = 128$ which means the 3 positive nodes are probably false positives in the network. If the system decides that there is a real activity in the network, then this is an incorrect decision.

In each simulation, we did 1000 runs and estimated the percentage of correct decisions made by the probabilistic model. If the number of repeats, r, is larger than one, the final decision is made by checking whether the number of answers is more or less than $(m_1 + m_2)/2$.

Figure 9 depicts the change in accuracy as the distance between peaks ($d = \frac{\mu_1 - \mu_2}{2}$) increase where μ_1, μ_2 are selected as $\mu_1 = \frac{n}{2} - d$ and $\mu_2 = \frac{n}{2} + d$. We observe that the increase in r increases accuracy for all distances. In addition, even nine repeats is sufficient to get more than 90% accuracy when the two sub-distributions are distant enough ($d > 32$). This is a significant reduction in query cost compared to the exact threshold querying algorithms. But it can be used only when the application is tolerant to error.

We also see that selecting r based on (10) ensures at least 90% accuracy if the two sub-distributions are totally seperated. As the bimodality increases, the required number of repeats

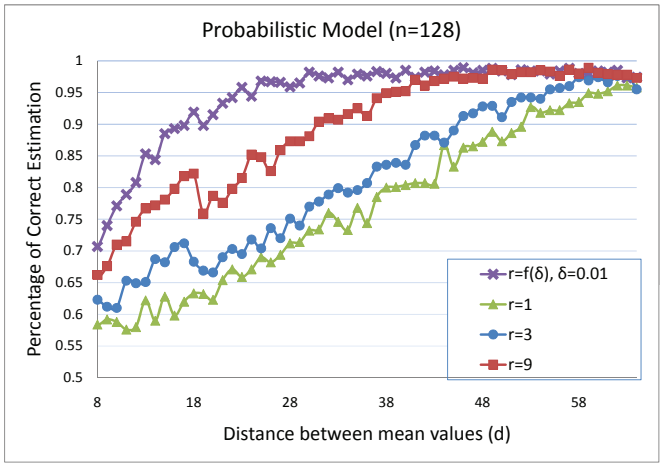

Fig. 9. Accuracy of probabilistic model as the number of repeats changes

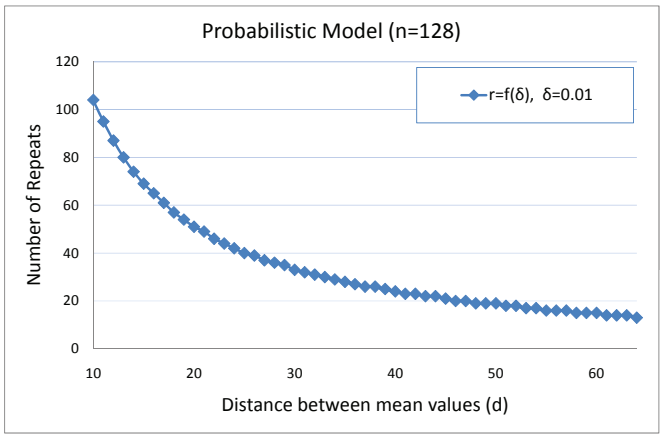

Fig. 10. Estimated number of repeats for 95% success rate

to maintain a high accuracy decreases (Figure 10). This total seperation occurs when $d > 16$. On the other hand, when $d \approx 8$, the probabilistic algorithm has a great difficulty in answering the threshold query having accuracies as low as 70%. The seperation of distributions when $d = 8$ and $d = 16$ are shown in Figure 11.

VII. Concluding Remarks

In this paper, we presented an efficient threshold querying primitive for singlehop collaborative feedback collection. Our results indicate that rather than using CSMA or sequential querying, by exploiting RCD and group testing nodes adaptively, we can answer the threshold queries more efficiently and scalably. Our extensive simulation results show that tcast primitive is especially low-cost for $x \ll t$ and $x \gg t$, and naturally requires more queries for $x \approx t$, where x denotes the number of positive nodes and t denotes the threshold. Since $x \approx t$ is the rare case and $x \ll t$ and $x \gg t$ are the common cases in most realistic detection applications, tcast provides an effective and efficient solution.

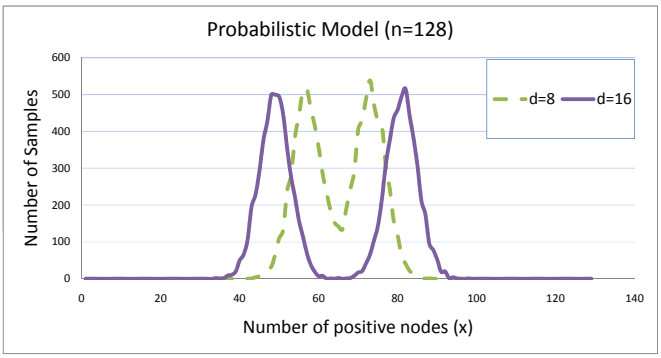

Fig. 11. The distribution of x is the combination of two normal distributions with seperation of $\mu_1 - \mu_2 = 2d$

We have also implemented our basic threshold querying primitive on WSN motes and deployed singlehop experiments with 14 motes. In our future work, we will deploy our implemention on the Kansei testbed [23], to get experimental results in a multihop network environment with interfering traffic. In future work we will also investigate implementations on RFID platforms [5], [25] as well.

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 2002.

[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y-R. Choi, T. Herman, S. S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks (Elsevier)*, 46(5):605–634, 2004.

[3] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Nesterenko, R. Shah, S. Kulkarni, M. Aramugam, L. Wang, M. Gouda, Y. Choi, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Exscal: Elements of an extreme scale wireless sensor network. *Int. Conf. on Embedded and Real-Time Computing Systems and Applications*, 2005.

[4] J. Aspnes, E. Blais, M. Demirbas, R. O'Donnell, A. Rudra, and S. Uurtamo. k+ decision trees. *International Workshop on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (Algosensors)*, 2010.

[5] M. Buettner, B. Greenstein, A. Sample, J. Smith, and D. Wetherall. Revisiting smart dust with rfid sensor networks. *Proc. 7th ACM Workshop on Hot Topics in Networks (Hotnets-VII)*, 2008.

[6] Chipcon. Cc2420 radio datasheet. http://focus.ti.com/docs/prod/folders/print/cc2420.html.

[7] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a sensor network architecture: Lowering the waistline. *The Tenth Workshop on Hot Topics in Operating Systems (HotOS X)*, 2005.

[8] M. Demirbas, A. Arora, and M. Gouda. A pursuer-evader game for sensor networks. *Proceedings of the Sixth Symposium on Self-Stabilizing Systems(SSS'03)*, pages 1–16, June 2003.

[9] M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani. A fault-local self -stabilizing clustering service for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems, Special Issue -Localized Communication*, 17(9):912–923, September 2006.

[10] M. Demirbas, O. Soysal, and M. Hussain. Singlehop collaborative feedback primitives for wireless sensor networks. *INFOCOM*, pages 2047–2055, 2008.

[11] M. Demirbas, O. Soysal, and A. S. Tosun. Data salmon: A greedy mobile basestation protocol for efficient data collection in wireless sensor networks. *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 267–280, 2007.

[12] Ding-Zhu Du and Frank K. Hwang. *Combinatorial Group Testing and its Applications*. World Scientific, 2000.

[13] A. Dutta, D. Saha, D. Grunwald, and D. Sicker. Smack: a smart acknowledgment scheme for broadcast messages in wireless networks. *SIGCOMM Comput. Commun. Rev.*, 39(4):15–26, 2009.

[14] P. Dutta, R. Musaloiu-e, I. Stoica, and A. Terzis. Wireless ack collisions not considered harmful. In *In HotNets-VII: The Seventh Workshop on Hot Topics in Networks*, 2008.

[15] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.

[16] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, 2003.

[17] O. Gnawali, K-Y Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler. The tenet architecture for tiered sensor networks. In *SenSys*, pages 153–166, 2006.

[18] Piotr Indyk. 6.895 Sketching, Streaming and Sub-linear Space Algorithms. *Lecture Notes*, 1, 2007.

[19] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.

[20] Moteiv. http://www.moteiv.com/.

[21] O. Soysal and M. Demirbas. Data spider: A resilient mobile basestation protocol for efficient data collection in wireless sensor networks. *The 6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2010.

[22] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. M. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Commun. ACM*, 47(6):34–40, 2004.

[23] OSU NEST ExScal Team. Kansei: Sensor testbed for at-scale experiments. http://www.cse.ohio-state.edu/kansei.

[24] G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson, and D. Culler. A macroscope in the redwoods. In *SenSys*, 2005.

[25] N. Vaidya and S. Das. Rfid-based networks: exploiting diversity and redundancy. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(1):2–14, 2008.

[26] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *NSDI*, pages 29–42, 2004.

[27] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSys*, pages 99–110, 2004.

[28] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. *The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.