Secure and Highly-Available Aggregation Queries in Large-Scale Sensor Networks via Set Sampling

Haifeng Yu
National University of Singapore
haifeng@comp.nus.edu.sg

ABSTRACT

Wireless sensor networks are often queried for aggregates such as predicate count, sum, and average. In untrusted environments, sensors may potentially be compromised. Existing approaches for securely answering aggregation queries in untrusted sensor networks can *detect* whether the aggregation result is corrupted by an attacker. However, the attacker (controlling the compromised sensors) can keep corrupting the result, rendering the system *unavailable*.

This paper aims to enable aggregation queries to *tolerate* instead of just detecting the adversary. To this end, we propose a novel *tree sampling* algorithm that directly uses sampling to answer aggregation queries. It leverages a novel *set sampling* technique to overcome a key and well-known obstacle in sampling — traditional sampling technique is only effective when the predicate count or sum is large. Set sampling can efficiently sample a set of sensors together, and determine whether any sensor in the set satisfies the predicate (but not how many). With set sampling as a building block, tree sampling can provably generate a correct answer despite adversarial interference, while without the drawbacks of traditional sampling techniques.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewalls)*; G.3 [**Probability and Statistics**]: *Probabilistic algorithms (including Monte Carlo)*

General Terms

Algorithms, Security, Design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IPSN'09, April 13–16, 2009, San Francisco, California, USA.
Copyright 2009 ACM 978-1-60558-371-6/09/04 ...$5.00.

Keywords

Aggregation queries, secure aggregation queries, sampling algorithms, set sampling, tree sampling

1. INTRODUCTION

Background. Wireless sensor networks are often queried for aggregates such as predicate count (e.g., number of sensors sensing fire), sum, and average. To answer these aggregation queries, traditional (non-secure) approaches typically use *in-network aggregation* along an *aggregation tree* [17]. In the aggregation tree, the root is the base station and all other tree nodes are sensors. Each sensor combines the results from its children, incorporates its own value, and then forwards a single value to its parent. In untrusted environments, sensors may potentially be compromised. These compromised sensors may launch two kinds of attacks: i) report arbitrary readings themselves, and ii) manipulate the partial aggregation results that pass through them. It is generally impossible to prevent the first attack without domain specific knowledge. Fortunately, it is also well-know that for robust aggregates [24] such as predicate count and sum, the first attack's influence is limited, as long as the fraction of malicious sensors is not overwhelming. The second attack can be much more serious since even a single malicious sensor can completely corrupt the final result. Thus the need to make aggregation queries secure against the second attack has been widely acknowledged [1, 2, 6, 10, 14, 15, 18, 26, 27].

Previous results and motivation. Most pioneering efforts [1, 6, 14, 15, 18] on secure aggregation queries make strong restrictive assumptions (e.g., assuming a single malicious sensor [14, 15] or assuming single-level aggregation [1, 6, 18]). Hierarchical secure aggregation [2, 10] is one of the first protocols that make general aggregation queries secure with provable guarantees. Unfortunately, the protocol can only *detect* but not *tolerate* malicious sensors. Namely, they enable the user to verify whether the result is corrupted. But even a *single* malicious sensor can keep preventing the verification from succeeding, in which case the user can never get a correct result. Thus despite that compromising a single sensor is a *local* attack on one sensor, the effect of such

attack is "amplified" by the protocol into a *global* DoS attack that renders the entire system unavailable. In such cases, [2] suggests having all sensors send individual readings back to the base station, which can be prohibitively expensive in large-scale systems. Another recent approach [23] has similar limitations – the attacker can launch multi-hop flooding attacks [5, 22, 25] from a few malicious sensors to stall the aggregation process.

We argue that for many applications, it is critical for the system to tolerate instead of just detecting the adversary. In other words, the aggregation queries must be both secure and highly-available. When under attack, detecting the adversary only makes the system *harmless* and not cause any damage. It is tolerating the adversary that makes the system *useful*. Without availability guarantees, the adversary (with human intelligence and judgment) would make the system unavailable precisely when the service is needed the most. For example, in battlefield monitoring, the enemy could cause the system to be unavailable exactly when the battle starts.

Our approach and results. This paper aims to design protocols with provable guarantees that can always correctly answer aggregation queries despite adversarial interference. To the best of our knowledge, this is the first effort toward such a goal in the general setting. A natural and seemingly obvious direction is to extend previous detection-only protocols to automatically pinpoint and revoke malicious sensors whenever the final result is corrupted. Unfortunately, because of the nature of these protocols and also because the malicious sensors may interfere with pinpointing as well, pinpointing malicious sensors is far from trivial and is a active research topic by itself. Secondly, malicious sensors may interfere sequentially one by one. This means that the service disruption time and the revocation overhead are at least linear with the number of malicious sensors.

As a result, we choose to substantially depart from most existing approaches [1, 2, 6, 10, 14, 15, 18, 27], which typically try to fix the security holes in in-network aggregation. We propose a novel *tree sampling* protocol that directly uses randomized sampling to answer aggregation queries. Despite adversarial interference, tree sampling can always produce a standard (ϵ, δ)-approximation of the correct result (i.e., the approximation is within $(1 \pm \epsilon)$ multiplicative factor of the correct result with probability at least $1 - \delta$).

Sampling has the nice security property that each sample involves the reading of a single sensor, and thus its integrity can be easily verified. This conveniently avoids the key challenge in in-network aggregation, where intermediate sensors need to aggregate multiple values into a single one and malicious sensors may not aggregate faithfully. However, sampling has its own well-known challenge — it is only efficient when the predicate count or sum is large. More precisely, the well-known lower bound [4] shows that with n sensors and the predicate count being b, $\Omega(\frac{n}{b}\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ samples are needed to obtain an (ϵ, δ)-approximation for b . When b is small, the term $\frac{n}{b}$ can approach $\Omega(n)$. Since all samples are forwarded to the base station, the sensors near the base station will thus send/receive $\Omega(\frac{n}{b}\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ bits. Such overhead quickly makes naive sampling impractical.

To address this challenge, we propose a simple but powerful *set sampling* technique to efficiently sample a set of sensors together. Sampling a set will tell whether any sensor in the set satisfies the predicate (but not how many sensors), with only $O(1)$ bits communication overhead on any sensor. Leveraging set sampling, our tree sampling protocol uses a binary tree to construct randomized sets and then samples those sets adaptively. This binary tree is a local data structure maintained by the base station, and *has no relation to network topology*. To compute the predicate count using tree sampling, each sensor only needs to send/receive $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log n)$ bits*, where $\delta' = \delta/(\log\max(4\epsilon^2 n, 2))$. Under practical parameters, $\log\frac{1}{\delta'}$ is almost never larger than $3\log\frac{1}{\delta}$.† In such cases, we have a cleaner form of $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log n)$ as the total number of bits.

Obviously, tree sampling breaks the previous lower bound of $\Omega(\frac{n}{b}\frac{1}{\epsilon^2}\log\frac{1}{\delta})$, and reduces the linear communication overhead to logarithmic overhead (with respect to n). Simulation shows that under the same parameters, naive sampling can achieve a similar estimation error as tree sampling only when roughly $b > 0.2n$. Next, leveraging the nice security property of sampling, we show that some minor (but subtle) modifications to tree sampling are sufficient to make it robust against malicious behavior (without affecting the protocol's overhead).

In summary, our novel tree sampling protocol provides qualitatively improved *functionality* (i.e., guaranteed availability) compared to existing secure aggregation protocols. It solves a key challenge in naive sampling, reducing the linear overhead to logarithmic overhead. The protocol then leverages the nice/clean security property of sampling to achieve our end goal. It is worth noting that we do not aim for better *performance* than previous detection-only protocols, though we will see later that some of them have comparable performance overheads as ours.

2. RELATED WORK

Hierarchical secure aggregation [2, 10] assumes that the exact set of live/reachable sensors is known. As a result, any *single* dead sensor, destroyed sensor, radio-jammed sensor, or compromised sensor can keep causing the final result to fail verification. It is not obvious how to relax this assumption, especially because their protocol needs the base station

*With some additional algorithmic tricks, we can actually do better such that each sensor sends/receives only $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\min(\frac{1}{\epsilon}, \epsilon^2 n) + \frac{1}{\epsilon}\log\frac{1}{\delta'}\log\max(\epsilon^2 n, 2))$ bits. For clarity, we leave these additional tricks to the appendix.

†For example, $\log\frac{1}{\delta'} \leq 3\log\frac{1}{\delta}$ holds for any $\epsilon \leq 0.5$, $\delta \leq 0.1$, and $n \leq 10^{10}$.

to verify the XOR of the MACs (i.e., Message Authentication Codes) generated by all sensors. Pinpointing and revoking the problematic sensors in untrusted sensor networks can be far from trivial, and is an active research topic by itself. In particularly, when the verification fails in [2, 10], the base station does not actually know which sensors' MACs are missing. Similarly, the recent SECOA secure aggregation protocol [21] also involves verification of folded values. It can thus become unavailable whenever a single malicious aggregator keeps corrupting the aggregation result.

Roy et al. [23] use verifiable Flajolet-Martin synopses [9] to make the aggregation secure. The idea is for sensors to generate MACs to "vouch" for the "1" bits in the Flajolet-Martin synopsis. However, because the MACs can only be verified by the base station and not by intermediate sensors forwarding the message, such a design makes multi-hop flooding attacks [5, 22, 25] possible. Namely, the adversary can inject many messages with fake MACs, which will all be forwarded to the base station and stall the propagation of legitimate messages. Compared to both approaches [2, 10, 23], our protocol can always answer the aggregation query despite all such attacks.

Yang et al. [27] propose a heuristic approach where the sensors are partitioned into groups and each group produces a single aggregation result. Groups with "outlier" results will be further probed. The error bound in their approach depends on deployment-specific factors such as sensor reading distribution and positions of the malicious sensors. In comparison, we aim to achieve provable approximation error. For secure aggregation in the Internet, Garofalakis et al. [12] use a similar approach as [23] except that the vouches are public key digital signatures. Generating digital signatures can be (prohibitively) expensive for resource-constrained sensors.

To the best of our knowledge, this paper is the first to propose the concept of tree sampling and set sampling for estimating aggregates such as predicate count and sum. Sampling of individual sensors is used in [1, 27] for detecting corrupted aggregation results, instead of computing the result. Sampling of individual sensors is also used in trusted environment to catch big events or "elephants"[11] (similar to b being large). Set sampling needs to leverage some properties of sensor networks (i.e., the ability of preloading keys onto the sensors). This could be part of the reason why despite sampling being well-studied in many other contexts (e.g., databases), set sampling has not been used for estimating aggregates. The well-known notion of *Ranked Set Sampling* (RSS) [3] in statistics, though with a similar name, is fundamentally different from our set sampling. RSS is designed for cases where each item may be either examined with low fidelity (e.g., visual inspection) or with high fidelity (e.g., precise measurement). With RSS, a set of sampled items is first examined with low fidelity and one of them is further examined with high fidelity.

3. MODEL AND PROBLEM STATEMENT

We consider a multi-hop sensor network with n deployed sensors and a trusted base station. The number of live sensors is unknown to our protocol. Each sensor shares a unique symmetric key with the base station. We assume that the base station knows an upper bound on the (multi-hop) round-trip time of the network.

Attack model. The adversary may compromise an arbitrary number of sensors, and potentially launch attacks from more powerful devices such as laptops. The adversary has a network-wide presence and may eavesdrop or inject messages at any point in the network. To make the system unavailable, the adversary may further launch a wide range of DoS-related attacks: i) physically destroying the sensors, ii) radio-jamming the sensors, and iii) launching DoS-related attacks from compromised sensors. One example of the last kind of attack is multi-hop flooding [5, 22, 25][‡] where the compromised sensors generate many fake responses. These responses are then all forwarded by honest sensors to the base station, which can stall the propagation of legitimate replies. Because the forwarding capacity of sensors is usually quite limited, it is rather easy for the fake messages to saturate such forwarding capacity. Multi-hop flooding is a serious attack in the sense that the honest sensors (by forwarding the fake messages) are unknowingly helping to "amplify" the scope of the attack.

We aim to provide aggregation results of the readings from those sensors that are not destroyed or radio-jammed. Destroyed sensors, radio-jammed sensors, and compromised sensors may potentially partition the sensor network. In such cases, our protocol will compute the aggregation results of the readings from those sensors that have paths to and from the base station. We do not assume symmetric links. If needed, by computing a secure count in parallel, the base station can estimate the number of reachable sensors which have contributed to the final result.

In the remainder of the paper, to unify terminology and simplify discussion, we pessimistically consider all physically destroyed, radio-jammed, compromised, and partitioned sensors (that do not have paths to/from the base station) as *malicious* sensors. Other sensors are *honest*. A malicious sensor is byzantine and controlled by the adversary.

Approximation error and performance metrics. Our goal is to compute (ϵ, δ)-*approximation* answers for aggregation queries such as predicate count, sum, and average. An (ϵ, δ)-*approximation* is guaranteed to be within $(1 \pm \epsilon)$ multiplicative factor of the *correct* answer with probability at least $1 - \delta$.[§] As in [1, 2, 10, 23], here a *correct* result allows malicious sensors to report arbitrary readings for themselves, but they are not allowed to add additional fabricated readings or

[‡][5, 22, 25] consider the multi-hop flooding attack (sometimes with a different name) in contexts other than aggregation.

[§]This holds for all system sizes, not just for "sufficiently large" n.

change the reported readings of honest sensors. For convenience later, we assume that $\epsilon \leq 0.5$ and $\delta \leq 0.5$. We further define $\delta' = \delta / \log \max(4\epsilon^2 n, 2)$. All log's in this paper are base-2.

We use *communication complexity* (also called *node congestion* in [2]) as the performance measure, which is the number of bits sent and received by each sensor (including those bits forwarded for other sensors during multi-hop forwarding). Tree sampling samples $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log n)$ sets in $O(\log n)$ *stages*, where all samples within a stage can be done in parallel. We thus expect the time complexity to be constrained by the communication complexity, and thus do not discuss time complexity explicitly.

Our protocol will only use symmetric key cryptography. Public key cryptography has become feasible [13] in some specific contexts [7] with specific sensor hardware. But it remains much more expensive than symmetric key operations. Using only symmetric key cryptography thus makes our protocols more general and more efficient.

4. OVERVIEW OF OUR APPROACH

The main challenge in secure in-network aggregation is to prevent malicious sensors from behaving dishonestly when aggregating multiple values into one. Somewhat ironically, leveraging sensors to aggregate data in-network is exactly the key idea behind in-network aggregation. In comparison, sampling has the nice security property that each sample only involves a single sensor's reading and thus it is easy to make it secure (e.g., by including a MAC on the sample). The challenge is how to avoid requiring an excessive number of samples when the predicate count or sum is small.

Our novel design of *set sampling* and *tree sampling* overcomes this challenge. To ensure that the sample request (reply) always reaches the sampled sensor (base station) despite adversarial interference, the request (reply) often needs to be flooded to all sensors already[¶]. Set sampling allows any sensor in some given set to respond to the request, since they all see the request already. But to avoid $\Omega(1)$ communication complexity near the base station, only a *single* response will be forwarded back to the base station. Sampling each set thus conceptually involves flooding the network twice, requiring each sensor to send/receive exactly two 8-byte payloads (see later for explanation on the payload size).

It is worth emphasizing that flooding is not inherently more expensive than local communication among neighboring sensors. For example, most previous secure aggregation protocols [2, 10, 12, 23] do not involve flooding, but require each sensor to communicate with its neighbors. However, flooding the network x times incurs exactly the same overhead as requiring each sensor to send/receive x messages to/from its neighbors. Instead of quantifying the overhead as numbers of flooding or numbers of local communication, we directly use communication complexity as a unified metric.

Tree sampling samples $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log n)$ sets where multiple samples can potentially be combined in one message. Our simulation later will show that even under $n = 10,000$, tree sampling can achieve an average ϵ of 0.08 while incurring only 250 to 300 samples. These samples are taken in around 5–15 sequential stages. For smaller n, the number of samples needed will be smaller as well.

Tree sampling has a rather similar communication complexity as Roy et al.'s detection-only aggregation protocol [23] and also Garofalakis et al.'s protocol [12]. All these protocols have a $\frac{1}{\epsilon^2}$ factor. This is quite fundamental in approximate counting and sampling: Even the well-know synopsis diffusion protocol [20], which is for aggregation in trusted environment, has such $\frac{1}{\epsilon^2}$ factor. Because of this $\frac{1}{\epsilon^2}$ factor, tree sampling works best if ϵ is not too small. As some concrete examples, Garofalakis et al. [12] mainly focus on ϵ between 0.1 and 0.25, while synopsis diffusion [20] considers ϵ around 0.15. Quite interestingly, the motivation behind synopsis diffusion is exactly to *reduce* the large ϵ error (well above 0.15) that results from message losses in traditional tree-based aggregation such as TAG [17].[‖] In other words, driving ϵ much lower than 0.1 can be non-trivial *even in trusted environments*.

5. SET SAMPLING

This section first describes a simple *keyed predicate test* protocol to make sampling robust against multi-hop flooding attacks, and then discuss set sampling. We discuss tree sampling afterward.

Keyed predicate test. Sampling is in general quite robust against adversarial interference. The sampled sensor can generate a MAC (i.e., Message Authentication Code) on the reading and then flood the reply back to the base station. But the malicious sensors can still launch multi-hop flooding attacks [5, 22, 25] and inject many fake replies to stall the propagation of the legitimate reply. Without public key cryptography, a forwarding sensor cannot authenticate the source of a reply and thus cannot tell whether the reply originates from the sampled sensor. As a result, the sensor will forward all the fake replies, potentially exhausting its forwarding capacity. This is similar to a DoS attack on router forwarding capacity in the Internet, except that multi-hop flooding attacks are much easier to launch due to sensors' limited forwarding capacity. Rate limiting will not help because a malicious sensor can claim that it is just forwarding replies originated from other sensors. Public key cryptography will not completely remove the problem either, since the sensors will

[¶]One could use secure routing to avoid flooding, but secure routing itself is an active research area and may not be "highly available".

[‖]One may wonder how message losses will affect the ϵ in tree sampling. Because tree sampling uses flooding, its robustness against message loss is at least as good as synopsis diffusion. One can view tree sampling as simultaneously addressing the security issue and the message loss issue.

need to both generate and verify signatures. This restricts the applicability of cryptographic techniques (e.g., Rabin signatures) with asymmetric cost.

Fortunately, there are existing solutions to tolerate multihop flooding attacks in the context of authenticated broadcast (from the base station). For example, Ning et al. [22] mitigate such attacks using message puzzles created by the base station (which has ample computational resources). Our keyed predicate test protocol will leverage such multihop-flooding-resilient "base station → sensor" communication to enable multihop-flooding-resilient "sensor → base station" communication (for the propagation of sample replies). From now on, by "authenticated broadcast" or "broadcast", we mean authenticated broadcast approaches (e.g., [22]) that are resilient to multi-hop flooding.

The keyed predicate test protocol tests whether any sensor holding a particular symmetric key K satisfies a certain predicate. We do not make any assumption on the number of sensors holding K – this will be useful later. For now we assume that a sensor holding K also knows some "name" for uniquely referring to K, so that the base station can refer to the key by its name without revealing the key itself. We remove this (mild) assumption later. In the protocol, the base station first (authenticated) broadcasts to all sensors:

$$\langle \text{name of } K, \text{ the predicate, } N, \text{ H}(\text{MAC}_K(N))\rangle$$

Here N is a nonce and H() is some well-known one-way hash function. $\text{MAC}_K(N)$ is the MAC generated on N using key K. An honest sensor holding K and satisfying the predicate will generate and locally broadcast the reply, $\langle \text{MAC}_K(N)\rangle$, to its neighbors. Any malicious sensor holding key K may do so as well. Other sensors simply record/store the hash $\text{H}(\text{MAC}_K(N))$. This hash serves as a fingerprint of $\text{MAC}_K(N)$. It allows the sensors to verify the valid reply of $\langle \text{MAC}_K(N)\rangle$, without enabling them to generate the reply if they do not know K. A (malicious) sensor not knowing K can replay the valid reply after seeing it at least once, but we will show that this does not matter. If an honest sensor receives a message whose hash matches the hash stored, it forwards the message via a local broadcast and then discards the stored hash.

Such design effectively prevents multi-hop flooding attacks: The only message that will propagate in the network is $\langle \text{MAC}_K(N)\rangle$, and furthermore every sensor will forward it *at most once*. Malicious sensors can inject fake responses, but those will never be forwarded. Fundamentally, we can achieve this because the base station knows the potential reply (i.e., $\text{MAC}_K(N)$), and can pre-compute and broadcast its hash. This also means that keyed predicate test only allows the base station to pose "yes/no" questions.

If within some timeout, the base station receives $\text{MAC}_K(N)$, we say that the test *succeeds*. The following theorem summarizes the security property of the protocol. Intuitively, only a sensor holding K can potentially create the first reply, and once a valid reply appears, it will keep propagating:

THEOREM 1. *If some honest sensor holding K satisfies the predicate, then the keyed predicate test always succeeds. If no honest sensor holding K satisfies the predicate and no malicious sensor holds K, then the keyed predicate test can never succeed.*

Proof sketch: The second claim is trivial since $\text{MAC}_K(N)$ will never be generated. For the first claim, it is obvious that some honest sensor A will generate the reply $\text{MAC}_K(N)$. The question is whether this reply will be propagated to the base station, given that each sensor only forwards at most one message. But if a sensor drops the reply from A because it previously forwarded some other reply, that other reply must be $\text{MAC}_K(N)$ as well. More formally, there exists a path (which is implicit and unknown by the protocol) from A to the base station that consists of only honest sensors (see Section 3), and each sensor on that path must have forwarded a reply $\text{MAC}_K(N)$. Thus the base station is guaranteed to receive $\text{MAC}_K(N)$. □

Sample a set efficiently. Set sampling enables the base station to test whether any sensor in a given set satisfies the predicate (but not *how many* sensors). It is straightforward to implement set sampling using keyed predicate test. For example, consider a set of three sensors $\{A, B, C\}$. At deployment time, we can load a symmetric key K corresponding to the set onto the three sensors. To sample the set, the base station simply performs a keyed predicate test on K. The communication complexity incurred is always $O(1)$ and is independent of the set size. Notice that such set sampling is possible only because we can preload K onto the three sensors. In other application domains of sampling (e.g., databases), these properties usually do not hold.

Set sampling has several important restrictions. First, the sets must be constructed before deployment. Second, since sensors may be deployed incrementally, when adding new sensors, we cannot change the set membership of old sensors. Finally, due to limited capacity of sensors and because each set requires loading a distinct symmetric key onto the sensor, a sensor can only belong to a small number of sample sets. Thus, only an extremely small fraction of all the 2^n power sets of the sensors can be used.

6. TREE SAMPLING – ASSUMING ALL SENSORS ARE HONEST

Section 6 through 8 explain the tree sampling protocol, in a progressive fashion. Section 6 first describes tree sampling for predicate count while assuming all sensors are honest. Section 7 explains why the protocol can easily account for malicious sensors, and proves formal guarantees. Section 8 generalizes to sum and average.

6.1 Overview for Computing Predicate Count

	meaning	example values	asymptotically
ϵ	multiplicative approximation factor	0.2	
δ	probability of the error exceeding ϵ	0.05	
δ'	defined as $\delta/\log\max(4\epsilon^2 n, 2)$	0.0045	
n	maximum number of sensors deployed	$10,000$	
b	number of black sensors (i.e., sensors satisfying the predicate)	0 to n	
h	height of the sampling tree	$\log(4n)$	
n_i	number of keys at level i	2^i	
b_i	number of black keys at level i	0 to n_i	
r_i	b_i/n_i	0 to 1	
α	level α is the level found by the binary search		
c_1	# samples taken at each level examined by the binary search	40	$\Theta(\log\frac{\log\log n}{\delta})$
c_2	# samples taken at level α	200	$\Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$
c_3	minimum # black keys needed to invoke the occupancy bound	30	$\Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$

Table 1: Notations used in the tree sampling protocol. The example values are also what we use later in simulation.

We say that a sensor is *black* if it satisfies the predicate in the predicate count query. Let b be the number of black sensors (Table 1). The naive way of sampling would be to draw certain number of uniformly random samples from the n sensors, calculate the fraction of black sensors observed, and finally multiply that fraction with n to obtain an estimation for b. To obtain an (ϵ, δ)-approximation for b in the above approach, we will need to sample $\Omega(\frac{n}{b}\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ sensors [4]. Intuitively, when b is small, we need many samples in order to encounter any black sensor. Before encountering a reasonable number of black sensors, one cannot properly estimate b.

Set sampling offers the potential of overcoming such lower bound. Namely, we can construct the sets such that regardless of how small b is, some sets will contain black sensors. For example, if we define a set to include all sensors, then as long as $b > 0$, the keyed predicate test on that set will succeed. This by itself of course, does not allow us to properly estimate b. In tree sampling, the basic idea is to leverage different (related) sets of different sizes. The key research question is how to define these sets and how to sample them (not all sets will be sampled), with the goal of minimizing the number of sets sampled.

Using the sampling tree to construct random sets. We use a *sampling tree* to construct the sets for sampling. This tree is an internal data structure stored by the base station, where every tree node is a distinct symmetric key. *The sampling tree has no relation to the topology of the sensor network. In particular, nodes/edges in the tree are not sensors/communication links*. Specifically, the sampling tree is a complete binary tree with $4N$ leaves and $\log(4N)$ levels. We will explain the magic number "4" later. N is a reasonable upper bound on the intended sensor network deployment size. If N is overly large (e.g., exponential of n), tree sampling will unnecessarily incur $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log N)$ instead of $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log n)$ communication complexity. The practical difference is likely to be small though since it is logarithmic. To simplify notation, we assume $N = n$ in the following.

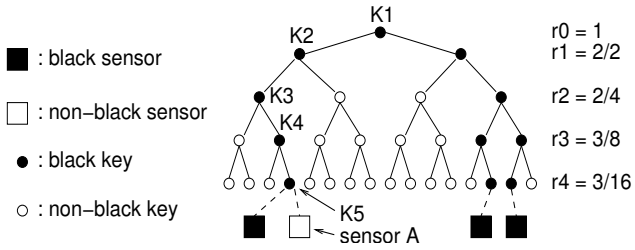

Figure 1: Example sampling tree with $n = 4$ and $b = 3$. Square boxes are sensors and circles are tree nodes (i.e., keys). Solid lines are tree edges and dashed lines are "association" relations between sensors and tree leaves.

The root of the sampling tree is at level 0 while the leaves are at level $h = \log(4n)$ (Figure 1). Each tree node is a distinct symmetric key. At deployment time, the base station generates random keys as tree nodes. For each sensor deployed (which can be deployed incrementally), the base station picks a uniformly random leaf, and then loads onto the sensor all $\log(4n)$ keys on the tree path from the root to that leaf. For example, in Figure 1, the five keys $K1$ through $K5$ are loaded onto sensor A. We say that the sensor is now *associated* with that leaf. After deploying n sensors, it is possible for some leaves never to be chosen or to be chosen multiple times. The sampling tree is only stored by the base station.

Protocol overview. From now on, when we say "sample a key K", we mean invoking the keyed predicate test on key

Step		# samples
1	Sample the root and return $\hat{b} = 0$ if the root is not black;	$O(1)$
2	Use binary search (Figure 3) to find level α with r_α not too close to 0 or 1;	$o(\log n \log \frac{1}{\delta})$
3	Sample level α using c_2 samples to get $\hat{r}_\alpha$;	$O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$
4	If $\hat{r}_\alpha < \frac{3}{20}$, set $\hat{r}_\alpha = \frac{3}{20}$; If $\hat{r}_\alpha > \frac{5}{6}$, set $\hat{r}_\alpha = \frac{5}{6}$;	—
5	If $\hat{r}_\alpha n_\alpha \geq c_3$, return $\hat{b} = \log(1 - \hat{r}_\alpha)/\log(1 - 1/n_\alpha)$;	—
6	Exhaustively sample all keys at level α (where the number of keys is guaranteed to be small);	$O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'})$
7	Track down the tree level by level starting from level α and let i be the current level:	$O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log n)$
8	If $r_i > \frac{5}{6}$, set $r_i = \frac{5}{6}$;	—
9	If $r_i n_i \geq c_3$, return $\hat{b} = \log(1 - r_i)/\log(1 - 1/n_i)$;	—
10	If $i = h$, sample sensors associated with black leaves and return the # of black sensors;	—

Figure 2: Pseudo-code for the tree sampling protocol (see text for where the constants (e.g., $3/20$) are from).

K. The tree sampling protocol will sample some carefully (and adaptively) chosen keys on the sampling tree. A key K at level i corresponds to a random set containing roughly $\frac{1}{2^i}$ fraction of all sensors.

A key K in the sampling tree is *black* if there is at least one black sensor holding K. Obviously, if a leaf key is black, then all keys on the path from that leaf to the root must be black as well. We use n_i and b_i to denote the number of keys and black keys at level i, respectively. We define $r_i = b_i/n_i$ (Figure 1).

If we knew the color of all tree nodes (i.e., keys), then one could estimate b relatively easily. However, the color of the keys are unknown beforehand, since it depends on which sensors satisfy the predicate in the current query. We can sample individual keys to reveal their colors, and the goal of tree sampling is to estimate b by revealing the colors of only a small fraction of the keys. To do so, the protocol will leverage a number of interesting properties of the sampling tree.

Figure 2 presents the high-level pseudo-code of the protocol, where Step 2 in the protocol is detailed in Figure 3. The protocol starts by sampling the root key, and returns the precise count of 0 if the root key is not black. Otherwise it uses the binary search protocol from Figure 3 to find a certain level α on the tree where r_α is bounded away from both 0 and 1. Later at Step 5, if $\hat{r}_\alpha \cdot n_\alpha$ is larger than some number c_3, the protocol can translate $\hat{r}_\alpha$ to the final estimation for b. (Here the hat " ˆ " means estimated value, same below.) If not, the protocol continues onto lower levels until certain conditions are met, and then return an estimation for b. The next explains the protocol in much greater detail, while ignoring Step 4 and 8. Section 7 will explain those two steps, which serve to deal with malicious sensors.

6.2 The "Appropriate" Level α and Finding It via Binary Search

As explained earlier, the challenge in naive sampling arises when b/n is small. From the structure of the sampling tree, one can trivially prove that $b_i/n_i = r_i$ monotonically increases as we move up the tree (Figure 1). At the root level, we must have $b_0/n_0 = r_0 = 1$ as long as $b > 0$. This suggests that we can at least effectively use naive sampling to estimate b_i for all those levels near the top of the tree. However, knowing those b_i's may or may not help us to estimate b. For example, knowing that $b_1 = 2$ gives us little information about b: We can only infer that $b \geq 2$, without having any likely upper bound for b. But if b_i/n_i is not close to 1, then those $(n_i - b_i)$ non-black keys are evidence that b is not likely to be larger than some value. Thus we need to find an "appropriate" level i where b_i/n_i is neither too close to 0 (so we can estimate b_i effectively via naive sampling on that level) nor too close to 1 (so we can translate b_i to b). Or rigorously speaking, r_i needs to be bounded away from 0 and 1 by constants.

2.1 $x = 0, y = h$;
2.2 repeat forever
2.3 $\quad i = \lfloor (x + y)/2 \rfloor$;
2.4 $\quad$ take c_1 samples at level i to get $\hat{r}_i$;
2.5 $\quad$ if $(\hat{r}_i > \frac{5}{8})$ $x = i$;
2.6 $\quad$ else if $(\hat{r}_i < \frac{3}{16})$ $y = i$;
2.7 $\quad$ else return i;
2.8 $\quad$ if $(x + 1 = y)$ return x;

Figure 3: Binary search on the levels.

For any i, we must have $n_{i+1} = 2n_i$ and $b_i \leq b_{i+1} \leq 2b_i$. This in turn means that on the sampling tree, $0.5 \leq \frac{r_{i+1}}{r_i} \leq 1$ (Figure 1). In other words, as we move down the tree, r_i monotonically decreases and the maximum decrease is half. At the bottom of the tree, we must have $r_h \leq b/(4n) \leq n/4n = \frac{1}{4}$. Together with $r_0 = 1$, this guarantees the existence of an "appropriate" level:

LEMMA 2. *If $b_0 = 1$, then there must exist an i ($0 \leq i \leq$*

h) such that $r_i \in [\frac{1}{4}, \frac{1}{2}]$.

Proof sketch: Prove by contradiction. Since $r_h < \frac{1}{4}$ and $r_0 = 1$, we must be able to find two adjacent levels j and $j+1$, where $r_j > \frac{1}{2}$ and $r_{j+1} < \frac{1}{4}$. But this would imply $r_{j+1}/r_j < 0.5$, which is impossible. □

The interval of $[\frac{1}{4}, \frac{1}{2}]$ in the lemma is chosen for simplicity. Any interval $[x, 2x]$ where $0 < x < 0.5$ will work, as long as the sampling tree has n/x leaves. Since we use $x = \frac{1}{4}$, our sampling tree is designed to have $4 \cdot n$ leaves.

To find such an "appropriate" level, since r_i is monotonic with i, we use a binary search on the $\log(4n)$ levels (Figure 3). For each level i examined, the protocol samples c_1 uniformly random keys on that level to generate an estimate $\hat{r}_i$. Namely, $\hat{r}_i$ is simply calculated as the faction of black keys among the c_1 sampled keys. Here $c_1 = c_1' \log \frac{\log \log n}{\delta}$ for some universal constant c_1'. Classic sampling theory [4] shows that taking c_1 samples gives the following error guarantee:

LEMMA 3. *There exists a universal constant c_1', such that taking $c_1 = c_1' \log \frac{\log \log n}{\delta}$ samples on any level i will allow us to produce $\hat{r}_i$ satisfying the following property: With probability at least $1 - \frac{\delta}{\log \log(4n)}$:*

$$\begin{cases} |\hat{r}_i - r_i| \le \frac{1}{4} r_i & \text{for } r_i \ge \frac{3}{20} \\ \hat{r}_i < \left(1 + \frac{1}{4}\right) \times \frac{3}{20} & \text{for } r_i < \frac{3}{20} \end{cases} \quad (1)$$

We say that an $\hat{r}_i$ is *good* if it satisfies Equation 1. The first inequality above says that if r_i is not too small, then $\hat{r}_i$ is within $(1 \pm \frac{1}{4})$ factor of r_i. The second inequality shows that if r_i is small, then at least $\hat{r}_i$ is not likely to be too large. We pick the factor $\frac{1}{4}$ for simplicity – any constant smaller than $\frac{1}{4}$ will work as well. All other constants in tree sampling are derived from this $\frac{1}{4}$ factor and the interval $[\frac{1}{4}, \frac{1}{2}]$ in Lemma 2.

Lemma 2 tells us that some r_i must be within $[\frac{1}{4}, \frac{1}{2}]$. Taking estimation error into account, the binary search thus stops and returns if it finds a level α with $\hat{r}_\alpha$ between $\frac{1}{4} \times (1 - \frac{1}{4}) = \frac{3}{16}$ and $\frac{1}{2} \times (1 + \frac{1}{4}) = \frac{5}{8}$ (Figure 4). Notice that if $\hat{r}_\alpha \in [\frac{3}{16}, \frac{5}{8}]$, then r_α may not actually be in $[\frac{1}{4}, \frac{1}{2}]$, but it is guaranteed to be within $\frac{3}{16}/(1 + \frac{1}{4}) = \frac{3}{20}$ and $\frac{5}{8}/(1 - \frac{1}{4}) = \frac{5}{6}$. This explains why we used $\frac{3}{20}$ in Lemma 3. On the other hand, $\frac{3}{20}$ is not necessary for the lemma to hold – the lemma actually holds for any positive constant bounded away from 0.

It is critical to notice that the binary search now operates on $\hat{r}_i$'s instead of on the accurate r_i's. Because of estimation error, $\hat{r}_i$'s may not actually be monotonic with i. The following theorem proves that despite non-monotonic $\hat{r}_i$'s, with probability at least $1 - \delta$, r_α is guaranteed to be within $[\frac{3}{20}, \frac{5}{6}]$. The theorem is a special case of a stronger theorem later (Theorem 6). We thus do not provide a redundant proof.

THEOREM 4. *Suppose $b_0 = 1$. With probability at least $1 - \delta$, the binary search is* successful *in the sense that it returns a level α where $r_\alpha \in [\frac{3}{20}, \frac{5}{6}]$.*

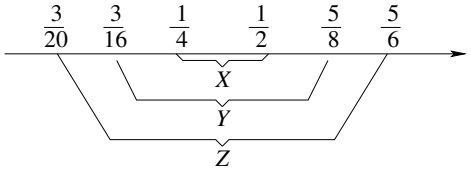

Figure 4: Three intervals $X = [\frac{1}{4}, \frac{1}{2}]$, $Y = [\frac{3}{16}, \frac{5}{8}]$ and $Z = [\frac{3}{20}, \frac{5}{6}]$. With $(1 \pm \frac{1}{4})$ multiplicative estimation error, the estimation of any value in X must be in Y, while the estimation of any value outside of Z can never be in Y.

The total number of samples taken by the binary search is $O(\log \log n \cdot \log \frac{\log \log n}{\delta}) = o(\log n \log \frac{1}{\delta})$. The protocol then (at Step 3) further uses $c_2 = \Theta(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ samples at level α to obtain an (ϵ, δ)-approximation of $\hat{r}_\alpha$ for r_α. This is possible because b_α/n_α is already known to be larger than $\frac{3}{20}$ (assuming the binary search was successful).

6.3 Using Occupancy Bound to Estimate b

The binary search finds a level α where r_α is bounded away from 1. Tree sampling will eventually estimate b based on either $\hat{r}_\alpha$ or some r_i where $i > \alpha$. Notice that since $i > \alpha$ and $r_i \le r_\alpha$, r_i will be guaranteed to be bounded away from 1 as well.

To obtain some intuition for the final estimation, consider any given level i and the n_i subtrees rooted at the n_i keys at that level. The probability that a given sensor is associated with any of the leaves of a given subtree is exactly $\frac{1}{n_i}$. We can thus draw a connection to the classic balls-into-bins problems [19] where each black sensor is a ball and goes into a uniformly random bin out of n_i bins. A key at level i is black iff the bin is not empty. Our task is to estimate the number of balls (b) given the number of occupied bins (b_i).

Given b balls and n_i bins, we have the expected number of occupied bins $E[b_i] = n_i \cdot (1 - (1 - 1/n_i)^b)$. If we substitute $E[b_i]$ with b_i and solve for b, we obtain one possible estimator for b:

$$\begin{aligned} \hat{b} &= \log(1 - b_i/n_i) / \log(1 - 1/n_i) \\ &= \log(1 - r_i) / \log(1 - 1/n_i) \end{aligned} \quad (2)$$

The crux, of course, is to understand how accurate the estimator is. We are able to prove that the above $\hat{b}$ is an (ϵ, δ)-approximation of b if i) r_i is bounded away from 1 by some constant, and ii) $r_i \cdot n_i \ge c_3 = c_3' \cdot \frac{1}{\epsilon^2} \log \frac{1}{\delta'} = \Theta(\frac{1}{\epsilon^2} \log \frac{1}{\delta'})$ for some universal constant c_3'. The proving technique is largely standard except that it uses a Chernoff-type occupancy tail bound from [8]. The complete proof is available as Lemma 11 in the appendix. It is worth noting that c_3 contains the term $\log \frac{1}{\delta'}$ (with $\delta' = \delta / \log \max(4\epsilon^2 n, 2)$) instead of $\log \frac{1}{\delta}$. This is because our proof needs to invoke a union bound across all $O(\log(4\epsilon^2 n))$ levels below level α.

6.4 "Tracking Down" the Tree

After the binary search returns a level α, the protocol first

tries to invoke an occupancy bound on level α (Step 5) to estimate b from $\hat{r}_\alpha$**. If this is not possible because $\hat{r}_\alpha \cdot n_\alpha < c_3$, the protocol exhaustively samples all n_α keys at level α (Step 6). Quite interestingly, we can efficiently do so exactly because $\hat{r}_\alpha \cdot n_\alpha < c_3$, which implies that $n_\alpha < c_3/\hat{r}_\alpha < \frac{20}{3}c_3 = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$.

Only black keys can have black children keys. Thus after knowing the color of all keys at level α, we can "track down" the tree efficiently starting from level α (Step 7). To "track down" to level $\alpha+1$ and reveal the color of all the keys on that level, the protocol simply samples all the children of the level-α black keys, incurring exactly $2b_\alpha \leq 2n_\alpha = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$ samples.

We now know the color of all keys at level $\alpha+1$. If $r_{\alpha+1} \cdot n_{\alpha+1} \geq c_3$, we can output $\hat{b}$ from Equation 2 with $i = \alpha+1$. Otherwise the protocol continues on and tracks down to level $\alpha+2$ by taking $2b_{\alpha+1} = r_{\alpha+1} \cdot n_{\alpha+1} < 2c_3 = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$ samples, and so on. A critical property here is that the number of samples incurred (for each level tracked) is independent of n_i and is always $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$. With $O(\log n)$ levels, this becomes $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log n)$.

6.5 Reaching the Bottom

We may reach the bottom of the tree without being able to invoke the occupancy bound. In such a case, we must have $b_h < c_3 = \Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$. The protocol then directly samples all sensors associated with the b_h leaf black keys (Step 8). Classic balls-into-bins problem [19] tells us that if we throw n balls into $4n$ bins uniformly randomly, with probability at least $1-\frac{1}{n}$, the most loaded bin contains $O(\log n)$ balls. Thus the expected number of balls in the most loaded bin will be at most $(1-\frac{1}{n})\cdot O(\log n) + \frac{1}{n}\cdot n = O(\log n)$. This mean that the b_h leaves will have on expectation $O(b_h \log n)$ sensors associated with them. Sampling all these sensors will take $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log n)$ samples on expectation.

7. TREE SAMPLING – ACCOUNTING FOR MALICIOUS SENSORS

This section models the disruptive behavior of malicious sensors, and then proves the end guarantees of tree sampling despite the malicious sensors. In particular, we will explain why Step 4 and 8 in Figure 2 are needed.

7.1 Modeling the Adversary

We say that a sensor is *black* if it is honest and it satisfies the predicate. It is *white* if it is honest but does not satisfy the predicate. A sensor is *grey* if it is malicious. We use b, w, and g to denote the number of black, white, and grey sensors, respectively. As discussed in Section 3, we do not aim at preventing the malicious sensors from lying about *their own* readings. Thus any result within $[b, b+g]$ is considered as *correct* for the predicate count query.

A key in the sampling tree is *black* if *at least one* black sensor holds it. A key is *white* if *only* white sensors hold it. Finally, a key is *grey* if it is neither black nor white. We let b_i, w_i, and g_i be the number of black, white, and grey keys at level i, respectively. The total number of keys at level i is still n_i. Similar as before, we consider an r_i value as *correct* for level i if $r_i \in [\frac{b_i}{n_i}, \frac{b_i+g_i}{n_i}]$. Namely, any value within the previous range corresponds to some possible "instantiation" of the readings from the malicious sensors. One can trivially prove the following simple *sandwiching property*:

LEMMA 5. *If r_i and r'_i ($r_i < r'_i$) are both correct for level i, then any $r''_i \in [r_i, r'_i]$ is correct for level i as well.*

Now let us consider how the adversary can attack tree sampling. A nice feature of tree sampling is that the protocol is executed locally by the trusted base station, except when the protocol invokes keyed predicate tests to sample keys. Keyed predicate tests always return a binary result. This extremely simple interface allows us to reason about security cleanly.

When we sample a key, we say that the key *tests black* if the keyed predicate test succeeds. Otherwise it *tests white*. Applying Theorem 1 immediately tells us that black keys always test black and white keys always test white. Thus the only possible attack on tree sampling is for the adversary to manipulate the (binary) sample results of grey keys. We assume that tree sampling never samples the same key more than once. This can be trivially achieved by the base station remembering the color of the keys already sampled. Now all the adversary can do is to control which grey keys should test black and which should test white.

On a given level i, regardless of the adversary's choices, tree sampling is guaranteed to "observe" some correct r_i when operating on that level. Namely, the r_i "observed" must fall between $\frac{b_i}{n_i}$ (when all grey keys test white) and $\frac{b_i+g_i}{n_i}$ (when all grey keys test black). Thus Lemma 3 still holds, except that it now holds for *some* correct $r_i \in [\frac{b_i}{n_i}, \frac{b_i+g_i}{n_i}]$ instead of for $r_i = \frac{b_i}{n_i}$. Similarly, Equation 2 (i.e., the occupancy bound) can still produce an (ϵ,δ)-approximation, since it is only concerned with the probabilistic property of a single level. On the other hand, the situation can become trickier when tree sampling relies on properties across multiple tree levels. As an example, the grey sensors can make the observed r_i and r_{i+1} to differ by more than a factor of 2, if they let all the level-i grey keys test black and all the level-$(i+1)$ grey keys test white. The next will prove tree sampling's guarantees despite all such attacks.

7.2 Provable Property of the Binary Search

We will first show that despite the existence of grey keys, the binary search from Figure 3 still provides a similar guar-

**Step 5 invokes Equation 2 using $\hat{r}_\alpha$ instead of the accurate r_α. Using the fact that r_α is bounded away from 1, one can easily verify that the ϵ-factor error in $\hat{r}_\alpha$ can be amplified by at most some constant factor when plugged into Equation 2. Thus the asymptotic property will not be affected.

antee as before:

THEOREM 6. *Suppose the root key tests black. Regardless of the behavior of the adversary, with probability at least* $1-\delta$*, the binary search is* successful *in the sense that it returns a level* α *where there exists some correct* $r_\alpha \in [\frac{3}{20}, \frac{5}{6}]$.

To obtain some intuition on why the theorem holds, we will refer to the steps in Figure 3. We say that an $\hat{r}_i$ obtained at Step 2.4 in Figure 3 is *good* if there exists *some* correct r_i satisfying Equation 1. Assuming all $\hat{r}_i$'s being good, if the binary search does find a level α with $\hat{r}_\alpha \in [\frac{3}{16}, \frac{5}{8}]$, then by definition, some correct r_α is guaranteed to be within $[\frac{3}{20}, \frac{5}{6}]$.

The tricky part is that the adversary may prevent the algorithm from finding an $\hat{r}_\alpha \in [\frac{3}{16}, \frac{5}{8}]$, and the binary search may reach two adjacent levels with $\hat{r}_i > \frac{5}{8}$ and $\hat{r}_{i+1} < \frac{3}{16}$. But if the two estimates $\hat{r}_i > \frac{5}{8}$ and $\hat{r}_{i+1} < \frac{3}{16}$ are both good, we must have some correct $r_i > \frac{1}{2}$ and some correct $r_{i+1} < \frac{1}{4}$. Given the correct $r_{i+1} < \frac{1}{4}$, one can show the existence of a second correct $r'_i < \frac{1}{2}$. The sandwiching property then immediately tells us that $r''_i = \frac{1}{2}$ is correct for level i as well. In other words, even if the binary search simply returns level i here (as at Step 2.8), we are still assured that some correct r''_i is within $[\frac{3}{20}, \frac{5}{6}]$.

Formalizing the above intuition will then prove Theorem 6:

Proof for Theorem 6: First of all, the binary search is guaranteed to return after examining $\log\log(4n)$ levels and taking $\log\log(4n)$ estimates. Since each estimate is good with probability at least $1 - \frac{\delta}{\log\log(4n)}$ (by Lemma 3), a simple union bound tells us that the probability that all estimates are good is at least $1-\delta$.

Conditioned upon all estimates being good, if an α is returned at Step 2.7, we know that $\hat{r}_\alpha \in [\frac{3}{16}, \frac{5}{8}]$. Since the estimate $\hat{r}_\alpha$ is good and satisfies Equation 1, we immediately have some correct $r_\alpha \in [\frac{3}{20}, \frac{5}{6}]$. Next consider the α returned from Step 2.8, with three possibilities:

- $x = 0, y = 1$, and the protocol returns $\alpha = 0$. The protocol must have obtained estimate $\hat{r}_1$ earlier and must have had $\hat{r}_1 < \frac{3}{16}$. We already know that the root key tested black and thus $b_0 + g_0 = 1$. If $b_0 = 0$ and $g_0 = 1$, then any $r_0 \in [0, 1]$ is correct and the theorem holds trivially. If $b_0 = 1$ and $g_0 = 0$, then we must have $b_1 \geq 1$ and any correct r_1 must be at least $\frac{1}{2}$. Given that $\hat{r}_1$ is a good estimate, we must have $\hat{r}_1 \geq (1 - \frac{1}{4}) \times \frac{1}{2} = \frac{3}{8}$. This contradicts with $\hat{r}_1 < \frac{3}{16}$.
- $x = h-1, y = h$, and the algorithm returns $\alpha = h-1$. The algorithm must have obtained estimate $\hat{r}_{h-1}$ earlier and must have had $\hat{r}_{h-1} > \frac{5}{8}$. We already know that all correct r_h's are at most $\frac{1}{4}$ and thus all correct r_{h-1}'s are at most $\frac{1}{2}$. Given that $\hat{r}_{h-1}$ is a good estimate, we must have $\hat{r}_{h-1} \leq (1 + \frac{1}{4}) \times \frac{1}{2} = \frac{5}{8}$. This contradicts with $\hat{r}_x > \frac{5}{8}$.
- $0 < x = y - 1 < y < h$, and the algorithm returns $\alpha = x$. Since $x > 0$ and $y < h$, the value assignments to x and y at Step 2.5 and 2.6 must both have been executed at least once. Then at Step 2.8, the protocol always maintains the invariant $\hat{r}_x > \frac{5}{8}$ and $\hat{r}_y < \frac{3}{16}$. We argue that there must exists a correct $r''_x = \frac{1}{2}$, as follows. Since $\hat{r}_x$ is good, there exists some correct $r_x > \frac{5}{8}/(1 + \frac{1}{4}) = \frac{1}{2}$. Since $\hat{r}_{x+1}$ (i.e., $\hat{r}_y$) is good, there exists some correct $r_{x+1} < \frac{3}{16}/(1 - \frac{1}{4}) = \frac{1}{4}$. This in turn means there must exist some correct $r'_x \leq 2r_{x+1} < \frac{1}{2}$. Given $r_x > \frac{1}{2}$ and $r'_x < \frac{1}{2}$, the sandwiching property tells us that there must exist a correct $r''_x = \frac{1}{2}$.

□

7.3 Main Theorem on Tree Sampling

We can now prove the main theorem on tree sampling regarding its communication complexity and approximation error. We first provide some intuition while referring to Figure 2. Tree sampling may return an estimation for b at Step 1, 5, 9, or 10. The result returned at Step 1 and 10 is always correct regardless of the behavior of the grey sensors. In particular for Step 10, when tracking down the tree, the algorithm will always track down black keys since the parent of a black key must be black as well. Step 5 and 9 invoke the occupancy bound. As explained earlier, the approximation error guarantee of the occupancy bound relies on the probabilistic property of a single level, and thus will still hold despite the grey keys.

The only tricky part is that the occupancy bound invoked by Step 5 implicitly requires that $\hat{r}_\alpha$ be bounded away from 1. Without grey keys and after a successful binary search, this requirement will always be met because a successful binary search guarantees $r_\alpha \leq \frac{5}{6}$. With grey keys, the binary search still guarantees that some correct r_α is no larger than $\frac{5}{6}$. But the grey keys on level α may interfere by testing black at Step 3[††]. This may cause the $\hat{r}_\alpha$ obtained at Step 3 to be close to 1.

The key observation here is that this close-to-one $\hat{r}_\alpha$ is entirely artificial, and is caused by the grey sensors testing black. Given that a successful binary search already guarantees the existence of some correct $r_\alpha \leq \frac{5}{6}$, if Step 3 obtains an $\hat{r}_\alpha > \frac{5}{6}$, we can simply ignore that $\hat{r}_\alpha$ and use $\frac{5}{6}$ instead. This is exactly what Step 4 does. By a similar argument, Step 4 can safely substitute overly small $\hat{r}_\alpha$ with $\frac{3}{20}$. The purpose of doing so is to ensure that the occupancy bound can be successfully invoked when n_α is $\Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$. Otherwise we might be forced to sample $\Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$ keys in Step 6, which would disrupt the asymptotic performance of the protocol.

A similar issue can arise when Step 9 invokes the occupancy bound, if the grey sensors cause the observed r_i to be close to 1. Similar as before, we already know that there is

[††]Step 3 takes a larger number of samples on level α than Step 2. Thus even though we never sample the same key twice, Step 3 may still encounter grey keys that are sampled for the very first time.

some correct $r_\alpha \le \frac{5}{6}$. This means there exists some correct $r'_i \le \frac{5}{6}$ (since $i > \alpha$ at Step 9). If we observe an $r_i > \frac{5}{6}$, there must exist another correct $r''_i = \frac{5}{6}$ (from the sandwiching property). This means that we can safely ignore the r_i obtained and use $\frac{5}{6}$ instead (Step 8).

The theorem below summarizes these arguments:

THEOREM 7. *Consider any given $0 < \delta_0 \le 0.5$ and $0 < \epsilon_0 \le 0.5$. (We use δ_0 and ϵ_0 to avoid notation collision.) Regardless of the behavior of the adversary, tree sampling in Figure 2 outputs a predicate count after taking on expectation $O(\frac{1}{\epsilon_0^2}\log\frac{1}{\delta'_0}\log n)$ samples, where $\delta'_0 = \delta_0/\log\max(4\epsilon_0^2 n, 2)$. Furthermore, with probability at least $1-\delta_0$, the output $\hat{b}$ is guaranteed to be within $(1\pm\epsilon_0)$ multiplicative factor of some x where $x \in [b, b+g]$.*

Proof sketch: We will refer to Figure 2, which annotates the number of samples taken in each step. The binary search deterministically takes $O(\log\log n \cdot \log\frac{\log\log n}{\delta}) = o(\log n \log\frac{1}{\delta})$ samples. If the algorithm reaches Step 6, then Step 4 ensures that $\hat{r}_\alpha \ge \frac{3}{20}$ and Step 5 implies that $\hat{r}_\alpha n_\alpha < c_3 = \Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$. This means that Step 6 needs to take $n_\alpha = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$ samples. For each of the $O(\log n)$ levels tracked in Step 7, the number of samples taken is $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$. Finally, Step 10 takes $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log n)$ samples on expectation (as explained earlier).

Next we prove the approximation error in $\hat{b}$. As explained earlier, we do not intend to prevent malicious sensors from reporting arbitrary values themselves, so any value $x \in [b, b+g]$ is considered correct. Tree sampling may output a $\hat{b}$ in Step 1, 5, 9, and 10. Consider these four cases one by one. In Step 1, if the root tests white, then we are guaranteed to have no black sensors and thus $\hat{b} = 0 \in [0, 0+g]$. In Step 10, we trivially have $\hat{b} \le n - w = b + g$, since white sensors can never test black. Furthermore, when tracking down the tree, we will never miss any black sensor since the parent (and ancestors) of a black sensor must be black. Thus $\hat{b} \ge b$ and $\hat{b} \in [b, b+g]$. For Step 5 and 9, define event $\mathcal{B}$ to be the event that the binary search is successful and the $\hat{r}_\alpha$ obtained in Step 3 is good. Obviously, the probability of $\mathcal{B}$ happening is at least $1-2\delta$. Conditioned upon $\mathcal{B}$, Lemma 8 below will show that i) the $\hat{r}_\alpha$ used in Step 5 is within $(1\pm\epsilon)$ factor of some correct r_α (even if $\hat{r}_\alpha$ has been modified in Step 4), and ii) the r_i used in Step 9 is correct (even if r_i has been modified in Step 8). Finally, conditioned upon $\mathcal{B}$, the occupancy bound tells us that the $\hat{b}$ returned in Step 5 or 9 is within $(1\pm\epsilon)$ factor of some $x \in [b, b+g]$ with probability at least $1-\delta$.

So far, the probability of $\hat{b}$ not falling within $(1\pm\epsilon)$ factor of x is at most 4δ (one δ introduced by each of Step 2, 3, 5, and 9). Taking $\delta = \delta_0/4$ and $\epsilon = \epsilon_0$ completes the proof. □

LEMMA 8. *To avoid notation collision, in the following we let $\hat{r}'_\alpha$ be the $\hat{r}_\alpha$ obtained in Step 3 and $\hat{r}_\alpha$ be the $\hat{r}_\alpha$ used by Step 5. With probability at least $1-2\delta$:*

- *$\hat{r}_\alpha$ (used by Step 5) is within $(1\pm\epsilon)$ factor of some correct r_α.*
- *r_i (used by Step 9) is correct.*

Proof sketch: We will prove that the two claims hold conditioned upon the binary search being successful and $\hat{r}'_\alpha$ being within $(1\pm\epsilon)$ factor of some correct r'_α. Obviously, these pre-conditions hold with probability at least $1-2\delta$.

If $\hat{r}'_\alpha \in [\frac{3}{20}, \frac{5}{6}]$, then $\hat{r}_\alpha = \hat{r}'_\alpha$ and we are done. Next consider the case of $\hat{r}'_\alpha > \frac{5}{6}$ and $\hat{r}_\alpha = \frac{5}{6}$. We know that $\hat{r}'_\alpha$ is within $(1\pm\epsilon)$ factor of some correct r'_α. If $r'_\alpha \le \frac{5}{6}$, then $\hat{r}_\alpha$ is a more accurate approximation (of r'_α) than $\hat{r}'_\alpha$. If $r'_\alpha > \frac{5}{6}$, notice that since the binary search is successful, some correct r''_α exists in $[\frac{3}{30}, \frac{5}{6}]$. From the sandwiching property, $r_\alpha = \frac{5}{6}$ must be correct as well, and obviously $\hat{r}_\alpha$ exactly equals this r_α. The case for $\hat{r}'_\alpha < \frac{3}{20}$ and $\hat{r}_\alpha = \frac{3}{20}$ is similar.

Next consider the r_i used by Step 9. Since the binary search is successful, some correct r''_α exists in $[\frac{3}{30}, \frac{5}{6}]$. This in turn means that some correct $r''_i \le \frac{5}{6}$ exists for any $i \ge \alpha$. Thus if the r_i obtained in Step 8 is larger than $\frac{5}{6}$, the sandwiching property tells us that some correct r_i must equal $\frac{5}{6}$. □

8. GENERALIZING TO SUM / AVERAGE

The tree sampling protocol for predicate count can be easily generalized to sum and average. Without loss of generality, we will assume that the reading on each sensor is an integer within $[1, m]$. The simplest way to compute the sum is to do a predicate count for each of the $\log_2 m$ bits in the binary form of readings. A much better approach, however, is for each sensor to "emulate" v virtual sensors. Predicate count can then be invoked on the $v \cdot n$ virtual sensors. At each invocation, a (physical) sensor can report a value between $[0, v]$. If we represent the sensor readings in base-$(v+1)$ format, we will only need to invoke predicate count $\log_{v+1} m$ times.

In each keyed predicate test, the communication complexity of a physical sensor is independent of the number of virtual sensors it emulates, since the physical sensor only needs to locally broadcast/forward the reply $\text{MAC}_K(N)$ at most once. Thus the only cost of emulating v virtual sensors is to store $O(\log(vn))$ keys for each of them. In most cases, we can afford to have $v = \sqrt{m}$, and thus only need to invoke predicate count $\log_{v+1} m = O(1)$ times. The resulting communication complexity is then $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log(nm))$, which is the same as $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log n)$ if $m = O(n)$.

Finally, average is simply sum divided by count. The generalization of the formal arguments on approximation error to sum and average is straightforward.

9. IMPLEMENTATION ISSUES

Despite the subtlety in some of its algorithmic concepts, tree sampling is rather straightforward to implement. Our simulator later implements the protocol from Figure 2 in less

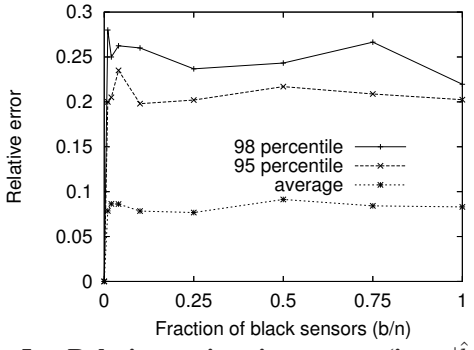

Figure 5: Relative estimation error (i.e., $|\hat{b} - b|/b$) achieved by tree sampling. The "x percentile" value means that x% of the 200 trials have an error below that value.

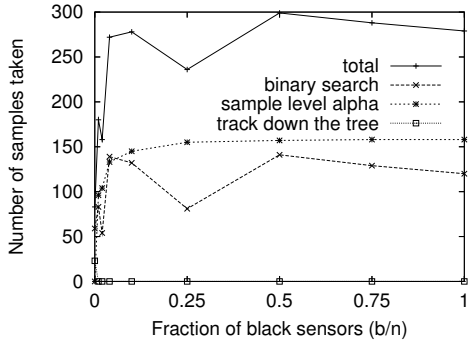

Figure 6: Average number of samples taken by tree sampling. In addition to the "total" number of samples, we also plot the number of samples taken during "binary search", for "sampling level α", and also for "tracking down the tree". The number of samples taken in all other steps are negligible.

than 200 lines of Java code. A salient feature of tree sampling is that the resource-constrained sensors only needs to implement the simple keyed predicate test from Section 5. The main protocol (i.e., Figure 2) is only on the base station. Also, only the base station needs to store the sampling tree and each sensor only stores $O(\log n)$ keys.

Several important optimizations below should be used in a real implementation. Each sample in the protocol involves a keyed predicate test. The predicate and the nonce N in the test are the same for all samples, and thus only need to be disseminated once. The base station does not need to indicate the name of the key either, since a sensor only has a limited number of keys and it can simply try all of them. Thus each sample will only involve the base station broadcasting H(MAC$_K(N)$), and then waiting for the potential reply of MAC$_K(N)$. This means that every sample will require each sensor to send and receive two 8-byte payloads. Tree sampling only has $O(\log n)$ stages, where all samples within a stage can be taken in parallel and can thus be combined into smaller number of message.

10. SIMULATION RESULTS

Our simulation aims to better understand the hidden constants in the asymptotic communication complexity of tree sampling, under some example parameter values. Our simulator does not simulate malicious sensors because simulation experiments fundamentally cannot cover all possible strategies of the adversary. Thus ultimately, the only way to reason about the impact of malicious sensors is via rigorous formal proofs, as in Section 7.

For all experiments, we use $n = 10,000$. This n value is intentionally chosen to be large so that it can capture most sensor networks today. Smaller n only makes our results better. We aim to achieve an ϵ of 0.2 and a δ of either 0.05 or 0.02. The average ϵ will be lower and will be around 0.08. Our choice of ϵ is consistent with prior work [12, 20]. We incorporate the following natural optimization: If the protocol intends to take s samples on a level (on the sampling tree) with less than s keys, it simply samples all keys on that level. We use $c_1 = 40$, $c_2 = 200$, and $c_3 = 30$ (as in Table 1) in our simulation. We have derived the asymptotic values of these three parameters earlier. One can directly plug in the constants derived in our proofs and the values of ϵ, δ, and n to obtain these parameters. Doing so is guaranteed to produce an (ϵ, δ)-approximation for b. On the other hand, since the proofs can be pessimistic, one can often use slightly better constants in practice. Tree sampling's estimation error is only affected by the probabilistic properties of the sampling tree, and is independent of the actual sensor network deployment. Thus to use the most appropriate constants in practice, one can always use simulation to determine such constants. Such simulation only need to be done once and *before* the sensor network is deployed. This is how we obtain the above c_1, c_2, and c_3 for our experiments.

For a given b, we perform 200 simulation trials, and calculate the relative approximation error (i.e., $|\hat{b} - b|/b$) in every trial. Figure 5 plots the average, 95-percentile, and 98-percentile approximation error achieved under different b values. The average error is around 0.08, while the 95-percentile is about 0.2. This means that the quality of the final estimate is roughly an (ϵ, δ)-approximation with $\epsilon = 0.2$ and $\delta = 0.05$. Figure 6 further quantifies the average number of samples taken by the algorithm. The total number of samples is roughly between 250 and 300, for all b values. These samples are taken in around 5–15 sequential stages.

As a quick comparison, our simulation also shows that naive sampling with 300 samples can provide a similar approximation error guarantee (i.e., $\epsilon = 0.2$ and $\delta = 0.05$) only when b is above $2,000$ (i.e., $n/b < 5$). For $b < 2,000$, the lower bound of $\Omega(\frac{n}{b}\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ starts to become prominent.

We explained earlier that while we do not aim at better performance, tree sampling's asymptotic communication complexity is similar to some previous aggregation-based approaches [12, 23]. These two approaches [12, 23] are rather

similar and both use Flajolet-Martin sketches with signatures on the bits. They thus are likely to have similar communication complexity. We will compare against proof sketches [12] since their paper provides detailed performance results. Proof sketches can achieve roughly $(\epsilon = 0.1, \delta = 0.05)$ using 256 Flajolet-Martin sketches. Each sketch contains up to $\log_2 n$ signatures. If each signature is a MAC, then sending a signature back (8 bytes) will incur half of the communication complexity as taking a sample (8×2 bytes) in our protocol. Using public key signatures will simply make the signature size larger and make our results better. Under $n = 10,000$, proof sketches will incur a communication complexity of roughly $256 \times \log_2 10^5 \approx 3,328$ signatures. One would expect that under $(\epsilon = 0.2, \delta = 0.05)$, the number of signatures will be reduced by a factor of $(0.2/0.1)^2 = 4$. This yields 832 signatures, which is comparable to taking 416 samples. Remember that tree sampling achieves $(\epsilon = 0.2, \delta = 0.05)$ with below 300 samples. This means that tree sampling's communication complexity is at least comparable to proof sketches.

11. CONCLUSION

This paper proposes a novel tree sampling protocol with provable guarantees to always correctly answer aggregation queries in sensor networks despite adversarial interference. As a sharp contrast to conventional approaches, tree sampling directly uses sampling to answer these queries. It leverages a novel set sampling technique to overcome a fundamental linear lower bound in traditional sampling, and takes only logarithmic number of samples (with respect to the number of sensors). We believe that the concept of set sampling and tree sampling can be rather general, and thus can potentially be applied to other problems as well.

12. ACKNOWLEDGMENTS

I thank Phillip B. Gibbons and Suman Nath for answering my questions about synopsis diffusion. I thank Aldar Chun-fai Chan, Mun Choon Chan, Ben Leong, and the anonymous reviewers for many helpful comments on this paper. I thank Andreas Terzis for shepherding this paper. This work is partly supported by NUS Young Investigator Award R-252-000-334-123.

13. REFERENCES

[1] Haowen Chan, Adrian Perrig, Bartosz Przydatek, and Dawn Song. SIA: Secure Information Aggregation in Sensor Networks. *Journal of Computer Security (Special Issue on Security of Ad Hoc and Sensor Networks)*, 15(1), 2007.

[2] Haowen Chan, Adrian Perrig, and Dawn Song. Secure Hierarchical In-network Aggregation for Sensor Networks. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2006.

[3] Zehua Chen, Zhidong Bai, and Bimal K. Sinha. *Ranked Set Sampling: Theory and Applications*. Springer, 2003.

[4] Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. An Optimal Algorithm for Monte Carlo Estimation. *SIAM Journal on Computing*, 29(5), 2000.

[5] J. Deng, R. Han, and S. Mishra. Limiting DoS Attacks During Multihop Data Delivery In Wireless Sensor Networks. *International Journal of Security and Networks, Special Issue on Security Issues in Sensor Networks*, 2006.

[6] W. Du, J. Deng, Y. Han, and P. K. Varshney. A Witness-based Approach for Data Fusion Assurance in Wireless Sensor Networks. In *Proceedings of Global Telecommunications Conference*, 2003.

[7] Wenliang Du, Ronghua Wang, and Peng Ning. An Efficient Scheme for Authenticating Public Keys in Sensor Networks. In *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing*, 2005.

[8] D. Dubhashi. Simple Proofs of Occupancy Tail Bounds. *Random Structructures and Algorithms*, 11(2), 1997.

[9] P. Flajolet and G.N. Martin. Probabilistic Counting Algorithms for Database Applications. *Journal of Computer and System Sciences*, 1985.

[10] K. Frikken and J. Dougherty. An Efficient Integrity-preserving Scheme for Hierarchical Sensor Aggregation. In *Proceedings of the ACM Conference on Wireless Network Security*, 2008.

[11] Sorabh Gandhi, Subhash Suri, and Emo Welzl. Catching elephants with mice: sparse sampling for monitoring sensor networks. In *SenSys*, 2007.

[12] Minos Garofalakis, Joseph M. Hellerstein, and Petros Maniatis. Proof Sketches: Verifiable In-Network Aggregation. In *Proceedings of the International Conference on Data Engineering*, 2007.

[13] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*, 2004.

[14] L. Hu and D. Evans. Secure Aggregation for Wireless Networks. In *Proceedings of Workshop on Security and Assurance in Ad hoc Networks*, 2003.

[15] P. Jadia and A. Mathuria. Efficient Secure Aggregation in Sensor Networks. In *Proceedings of the 11th International Conference on High Performance Computing*, 2004.

[16] A. Kamath, R. Motwani, K. Palem, and P. Spirakis. Tail Bounds for Occupancy And the Satisfiability Threshold Conjecture. *Random Structures Algorithms*, 7(1), 1995.

[17] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2002.

[18] A. Mahimkar and T. Rappaport. SecureDAV: A Secure Data Aggregation and Verification Protocol for Sensor Networks. In *Proceedings of the IEEE Global Telecommunications Conference*, 2004.

[19] Michael Mitzenmacher and Eli Upfal. *Probability and Computing – Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[20] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis Diffusion for Robust Aggregation in Sensor Networks. In *SenSys*, 2004.

[21] Suman Nath and Haifeng Yu. Secure Outsourced Aggregation via One-way Chains. In *ACM SIGMOD*, June 2009.

[22] Peng Ning, An Liu, and Wenliang Du. Mitigating DoS Attacks against Broadcast Authentication in Wireless Sensor Networks. *ACM Transactions on Sensor Networks*, 2008.

[23] Sankardas Roy, Sanjeev Setia, and Sushil Jajodia. Attack-resilient Hierarchical Data Aggregation in Sensor Networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2006.

[24] David Wagner. Resilient Aggregation in Sensor Networks. In *Proceedings of the ACM Workshop on Security of Ad hoc and Sensor Networks*, 2004.

[25] Ronghua Wang, Wenliang Du, and Peng Ning. Containing Denial-of-Service Attacks in Broadcast Authentication in Sensor Networks. In *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing*, 2007.

[26] Y. Wang, G. Attebury, and B. Ramamurthy. A Survey of Security

Issues in Wireless Sensor Networks. *IEEE. Communications Surveys & Tutorials*, 8(2), 2006.

[27] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks. In *Proceedings of ACM Symposium on Mobile Ad hoc Networking and Computing*, 2006.

APPENDIX

A. PROOF FOR THE OCCUPANCY BOUND USED IN SECTION 6.3

LEMMA 9. **Chernoff-type occupancy tail bound [8].** *Consider an experiment where we throw b balls into n_i bins. Let random variable Z denote the number of empty bins and $b_i = n_i - Z$, where $E[Z] = n_i(1 - \frac{1}{n_i})^b$ and $E[b_i] = n_i - n_i(1 - \frac{1}{n_i})^b$. Then we have:*

$$Pr[|Z - E[Z]| > t] \quad < \quad 2exp(-2t^2/b)$$

Comments on Lemma 9. There are other forms of occupancy tail bounds [16]. The bound in Lemma 9 is not the strongest one. However, its form is convenient for our purpose.

Introducing Lemma 10. To be as strong as possible, Lemma 10 below will assume that the binary search returns a level α where $r_\alpha \leq a < 1$. In other words, we will show that r_α being bounded away from 1 is sufficient. Theorem 6 proves that $r_\alpha \in [\frac{3}{20}, \frac{5}{6}]$, which is obviously more than sufficient to satisfy the condition in the lemma.

LEMMA 10. *For any given constant $0 < a < 1$, we can find a universal constant $c_3' > 1$ with the following property. Define:*

$$\begin{aligned} \delta' &= \frac{\delta}{\log \max(4\epsilon^2 n, 2)} \\ \hat{b} &= \frac{\log(1 - r_i)}{\log(1 - 1/n_i)} \\ c_3 &= c_3' \cdot \frac{1}{\epsilon^2} \log \frac{1}{\delta'} \end{aligned}$$

For any $0 \leq i \leq h$, let $\mathcal{E}_i$ be the indicator random variable denoting the event of:

$$(b_i/n_i \leq a) \text{ and } (b_i \geq c_3) \text{ and } (|\hat{b} - b| > \epsilon b)$$

Then for any $0 \leq i \leq h$:

$$Pr[\mathcal{E}_i] \quad \leq \quad \delta'$$

Proof sketch: Consider any given i and define:

$$d = \frac{\log(0.5 - 0.5a)}{\log 0.37} > 0.5$$

We prove the lemma for two different regions of b values: $b \in (d \cdot n_i, n]$ and $b \in [0, d \cdot n_i]$.

Case 1: $b \in (dn_i, n]$**.** We have the following relation:

$$\begin{aligned} Pr[\mathcal{E}_i] &\leq Pr[(b_i \leq an_i) \text{ and } (b_i \geq c_3)] \\ &\leq Pr[(b_i \leq an_i) \text{ and } (n_i \geq c_3)] \end{aligned}$$

Thus it suffices to prove:

$$Pr[(b_i \leq an_i) \text{ and } (n_i \geq c_3)] < \delta'$$

Notice that for any given i, n_i is not a random variable. If for the given i we have $n_i < c_3$, then the above event has 0 probability. Thus we only need to prove that for any given i where $n_i \geq c_3$, $Pr[b_i \leq an_i] < \delta'$. Obviously, the probability of $b_i \leq an_i$ will only decrease as b increase. Thus it suffice to show that for $b = d \cdot n_i$, $Pr[b_i \leq an_i] < \delta'$. When $b = d \cdot n_i$, we have for any $c_3' \geq x_1 = 4d/(1-a)^2$:

$$\begin{aligned} Pr[b_i \leq an_i] &= Pr[b_i - E[b_i] \leq an_i - E[b_i]] \\ &< Pr[E[b_i] - b_i > n_i(1 - 0.37^{b/n_i}) - an_i] \\ &= Pr[Z - E[Z] > (0.5 - 0.5a)n_i] \\ &< 2exp(-0.5(1-a)^2 n_i^2/b) \\ &= 2exp(-0.5(1-a)^2 n_i/d) \\ &< 2exp(-2\log(1/\delta')) < \delta' \end{aligned}$$

Case 2: $b \in [0, dn_i]$**.** All the discussion below is conditioned upon $b_i \geq c_3$, which implies $n_i \geq c_3$ as well as $b \geq c_3$. Let $p_1 = Pr[\hat{b} > (1+\epsilon)b]$ and $p_2 = Pr[\hat{b} < (1-\epsilon)b]$. We will show that $p_1 \leq \delta'/2$ and $p_2 \leq \delta'/2$, respectively. Consider p_1 first:

$$\begin{aligned} p_1 &= Pr[\hat{b} > (1+\epsilon)b] \\ &= Pr\left[b_i - E[b_i] > n_i(1 - (1 - 1/n_i)^{(1+\epsilon)b}) - E[b_i]\right] \\ &= Pr\left[E[Z] - Z > n_i(1 - (1 - 1/n_i)^{(1+\epsilon)b}) - E[b_i]\right] \end{aligned}$$

Let:

$$\begin{aligned} t &= n_i(1 - (1 - 1/n_i)^{(1+\epsilon)b}) - E[b_i] \\ &= n_i((1 - 1/n_i)^b - (1 - 1/n_i)^{(1+\epsilon)b}) \\ &= n_i(1 - 1/n_i)^b(1 - (1 - 1/n_i)^{\epsilon b}) \end{aligned}$$

For any $c_3' \geq x_2 = 24$, we have $n_i \geq b_i \geq c_3' \frac{1}{\epsilon^2} \log \frac{1}{\delta'} \geq 24 \log 2 > 12$ and thus $(1 - 1/n_i)^{n_i} \geq 0.35$. This means:

$$(1 - 1/n_i)^b \geq (1 - 1/n_i)^{dn_i} \geq 0.35^d$$

Also, we have:

$$\begin{aligned} 1 - (1 - 1/n_i)^{\epsilon b} &= 1 - ((1 - 1/n_i)^{n_i})^{\epsilon b/n_i} \\ &> 1 - exp(-\epsilon b/n_i) \end{aligned}$$

The above three equations imply:

$$t > n_i \times 0.35^d \times (1 - exp(-\epsilon b/n_i))$$

If $b < 0.5n_i$, then we can show that $1 - exp(-\epsilon b/n_i) > 0.75\epsilon b/n_i$ as follows. Define $x = \epsilon b/n_i$ where $0 \leq x \leq 0.5$, and $f(x) = 1 - exp(-x) - 0.75x$. One can easily verify that $f(x) \geq 0$ for all $x \in [0, 0.5]$. Thus we have

$$t > n_i \times 0.35^d \times 0.75\epsilon b/n_i = 0.35^d \times 0.75 \times \epsilon b$$

Applying Lemma 9 thus shows that for any $c'_3 \geq x_3 = 3/(1.125 \times 0.35^{2d})$:

$$\begin{aligned} p_1 &< 2exp(-2t^2/b) \\ &\leq 2exp(-1.125 \times 0.35^{2d} \times \epsilon^2 b_i) \\ &\leq 2exp(-3\log(1/\delta'))) \leq \delta'/2 \end{aligned}$$

If $0.5n_i \leq b \leq dn_i$, then we have $1 - exp(-\epsilon b/n_i) > 1 - exp(-\epsilon/2) > 0.375\epsilon$ and $t > 0.35^d \times 0.375 \times \epsilon n_i$. Applying Lemma 9 shows that for any $c'_3 \geq x_4 = 3d/(0.28 \times 0.35^{2d})$:

$$\begin{aligned} p_1 &< 2exp(-2t^2/b) \\ &< 2exp(-0.28 \times 0.35^{2d} \times \epsilon^2 n_i/d) \\ &< 2exp(-3\log(1/\delta'))) < \delta'/2 \end{aligned}$$

We move on to p_2:

$$\begin{aligned} p_2 &= Pr[\hat{b} < (1-\epsilon)b] \\ &= Pr\left[\frac{\log(1-b_i/n_i)}{\log(1-1/n_i)} < (1-\epsilon)b\right] \\ &= Pr\left[b_i < n_i\left(1-(1-1/n_i)^{(1-\epsilon)b}\right)\right] \\ &= Pr\left[E[b_i] - b_i > E[b_i] - n_i(1-(1-1/n_i)^{(1-\epsilon)b})\right] \\ &= Pr\left[Z - E[Z] > E[b_i] - n_i(1-(1-1/n_i)^{(1-\epsilon)b})\right] \end{aligned}$$

Let:

$$\begin{aligned} t &= E[b_i] - n_i(1-(1-1/n_i)^{(1-\epsilon)b}) \\ &= n_i((1-1/n_i)^{(1-\epsilon)b} - (1-1/n_i)^b) \\ &= n_i(1-1/n_i)^b((1-1/n_i)^{-\epsilon b} - 1) \end{aligned}$$

We have:

$$\begin{aligned} (1-1/n_i)^{-\epsilon b} - 1 &= ((1-1/n_i)^{n_i})^{-\epsilon b/n_i} - 1 \\ &> exp(\epsilon b/n_i) - 1 > \epsilon b/n_i \end{aligned}$$

Thus we have $t > 0.35^d \epsilon b$. Applying Lemma 9 shows that for any $c'_3 \geq x_5 = 3/(2 \times 0.35^{2d})$, we have:

$$\begin{aligned} p_2 &< 2exp(-2t^2/b) < 2exp(-2 \times 0.35^{2d}\epsilon^2 b) \\ &< 2exp(-3\log(1/\delta'))) < \delta'/2 \end{aligned}$$

Finally, letting $c'_3 = max(1, x_1, x_2, x_3, x_4, x_5)$ will then complete the proof. □

LEMMA 11. *For any given constant $0 < a < 1$, we can find a universal constant $c'_3 \geq 1$ with the following property. Define:*

$$\begin{aligned} \delta' &= \frac{\delta}{\log\max(4\epsilon^2 n, 2)} \\ \hat{b} &= \frac{\log(1-r_i)}{\log(1-1/n_i)} \\ c_3 &= c'_3 \cdot \frac{1}{\epsilon^2}\log\frac{1}{\delta'} \end{aligned}$$

Let $\mathcal{E}$ be an indicator random variable denoting the event that there exists some level $0 \leq i \leq h$ on the sampling tree such that:

$$(b_i/n_i \leq a) \text{ and } (b_i \geq c_3) \text{ and } (|\hat{b} - b| > \epsilon b)$$

Then:

$$Pr[\mathcal{E}] \leq \delta$$

Proof sketch: We applying Lemma 10 and set c'_3 the same as the c'_3 from Lemma 10. There are total $\log(4n)$ levels on the sampling tree, out of which at most $\log(4\epsilon^2 n)$ levels have at least c_3 nodes. For the remaining levels, it is impossible for the event $b_i \geq c_3$ to occur, and thus we do not need to worry about them. Applying a union bound across the $\log(4\epsilon^2 n)$ levels then immediately proves the lemma. □

Comments on Lemma 11. Let $\mathcal{E}'$ denote the event that our tree sampling algorithm returns an estimation via the occupancy bound and the estimation has larger than ϵ multiplicative error. Then trivially, we have $Pr[\mathcal{E}'] \leq Pr[\mathcal{E}]$ where $\mathcal{E}$ is defined as in Lemma 11. Namely, if $\mathcal{E}$ does not occur, then it is impossible for $\mathcal{E}'$ to occur. As a result, Lemma 11 upper bounds $Pr[\mathcal{E}']$ as well.

Also notice that it is necessary for us to reason about all levels in Lemma 11, instead of just the level from which the tree sampling algorithm returns a final result. This is because the algorithm might return from the level that provides the largest approximation error. We are not able to precisely reason about such probability, and as a result, we apply a pessimistic union bound across all levels in Lemma 11.

B. FURTHER REDUCING THE NUMBER OF SAMPLES TAKEN

The tree sampling protocol described in this paper so far incurs $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log n)$ samples for predicate count. This section presents some further techniques that can reduce the number of samples down to $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\min(\frac{1}{\epsilon}, \epsilon^2 n) + \frac{1}{\epsilon}\log\frac{1}{\delta'}\log\max(\epsilon^2 n, 2))$. To achieve this reduced number of samples, we will reduce/bound the number of samples taken during the following steps in tree sampling:

- Currently the binary search at Step 2 (Figure 2) takes $O(\log\log n \cdot \log\frac{\log\log n}{\delta})$ samples. Section B.1 will prove that $(\log\log n) \cdot (\log\log\log n) = O\left(\frac{1}{\epsilon} \cdot \log\max(\epsilon^2 n, 2)\right)$, which in turn implies $O(\log\log n \cdot \log\frac{\log\log n}{\delta}) = O(\frac{1}{\epsilon}\frac{1}{\delta'}\log\max(\epsilon^2 n, 2))$.
- Currently Step 10 (Figure 2) samples all sensors associated with the $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$ black leaves, where on expectation $O(\log n)$ sensors may be associated with each black leave. Section B.2 will show that a minor optimization can drive the total number of samples taken down to $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$.
- Current Step 7 (Figure 2) tracks down the $O(\log n)$ levels of the tree, where each level incurs $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$ samples. Section B.3 will explain how to reduce the number of samples taken down to $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\min(\frac{1}{\epsilon}, \epsilon^2 n) + \frac{1}{\epsilon}\log\frac{1}{\delta'}\log\max(\epsilon^2 n, 2))$.

B.1 Number of Samples Taken During Binary Search in Step 2

LEMMA 12. *For any* $0 < \epsilon \leq 1$*, we have:*

$$(\log\log n) \cdot (\log\log\log n) = O\left(\frac{1}{\epsilon} \cdot \log\max(\epsilon^2 n, 2)\right)$$

Proof: We will prove that $(\log\log n)\cdot(\log\log\log n) \leq 2/\epsilon \cdot \log\max(\epsilon^2 n, 2)$. If $\epsilon^2 \leq 2/n$, we need to prove:

$$(\log\log n) \cdot (\log\log\log n) \leq 2/\epsilon$$

The above equation holds because $2/\epsilon \geq \sqrt{2n}$ and because $\sqrt{2n} \geq (\log\log n) \cdot (\log\log\log n)$ holds for all n. Now consider $\epsilon^2 > 2/n$, and we need to prove:

$$(\log\log n) \cdot (\log\log\log n) \leq \frac{2}{\epsilon} \cdot \log(\epsilon^2 n)$$

Let $x = \epsilon$ and define $f(x) = (\log\log n) \cdot (\log\log\log n) - (2/x) \cdot \log(x^2 n)$. One can easily verify that $f'(x) \leq 0$ for $\sqrt{2/n} \leq x \leq \sqrt{4/n}$ and $f'(x) \geq 0$ for $\sqrt{4/n} \leq x \leq 1$. This means that the maximum of $f(x)$ is reached either when $x = \sqrt{2/n}$ or when $x = 1$. We have $f(\sqrt{2/n}) = (\log\log n) \cdot (\log\log\log n) - \sqrt{2n} < 0$ for all n, and also $f(1) = (\log\log n) \cdot (\log\log\log n) - 2\log n < 0$ for all n. Thus $f(x)$ is negative for all x and n. □

B.2 Reducing Samples Taken When Sampling All Black Leaves in Step 10

Reducing the number of samples taken in Step 10 is simple. Instead of using a sampling tree with $4n$ leaves, we can use a sampling tree with $4n^{1.5}$ leaves. This will make the expected number of sensors associated with any leave to be $O(1)$. On the other hand, The $n^{1.5}$ term will only appear in logarithmic form in other steps of the protocol, which is still $O(\log n)$. This technique thus reduces the number of samples taken in Step 10 to $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$, without affecting the asymptotic number of samples taken in other steps.

It may appear that a sampling tree with $4n^{1.5}$ leaves can be quite large, and even the resourceful base station may not be able to easily store it. Interestingly, even if the sampling tree has $4n^{1.5}$ leaves, every sensor will still be loaded with only $\log(4n^{1.5})$ keys. With total n sensors, the total number of keys that are actually loaded onto the sensors is at most $n\log(4n^{1.5}) = O(n\log n)$. This means that a lot of the keys in the sampling tree will not actually be used. The base station can thus create keys only as needed in a lazy fashion, and the total number of keys that the base station needs to store is $O(n\log n)$ instead of $O(n^{1.5})$. To be concrete, even with $n = 10,000$, the number of keys needed to be stored by the base station is only $220,000$, which is trivial compared to even a commodity PC's disk space.

B.3 Reducing Samples Taken When Tracking Down the Tree in Step 7

To reduce the number of samples taken in Step 7, we will reduce the number of samples taken at each level. The number of samples taken at each level is determined by the occupancy bound, which can only be invoked when $b_i = \Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta'})$. We have developed another *small occupancy bound* that can be invoked when $b_i = \Theta(\frac{1}{\epsilon}\log\frac{1}{\delta'})$ and $r_i \leq \epsilon/2$. This allows us to stop tracking down the tree whenever either the occupancy bound condition is met or the small occupancy bound condition is met. (If both conditions are met, either of them can be used.) Using these two bounds, we will be able to eventually reduce the number of samples taken in Step 7 to $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\min(\frac{1}{\epsilon}, \epsilon^2 n) + \frac{1}{\epsilon}\log\frac{1}{\delta'}\log\max(\epsilon^2 n, 2))$.

Small occupancy bound. We continue our analogy to the balls-into-bins problem from Section 6.3. When the fraction of occupied bins is small, one would imagine that b is not much larger than b_i. Lemma 13 below formalizes such intuition, and proves that $\hat{b} = b_i$ is already an (ϵ, δ)-approximation of b if $b_i/n_i \leq \frac{\epsilon}{2}$ and if $b_i \geq c_4 = c_4' \cdot \frac{1}{\epsilon}\log\frac{1}{\delta'} = \Theta(\frac{1}{\epsilon}\log\frac{1}{\delta'})$ for some universal constant c_4'. Compared to the occupancy bound in Lemma 10, both require a large enough b_i, but the requirement here is relaxed by a factor of $\frac{1}{\epsilon}$.

LEMMA 13. *There exists a universal constant* $c_4' > 2$ *with the following property. Define*

$$\begin{aligned} \delta' &= \frac{\delta}{\log\max(4\epsilon^2 n, 2)} \\ \hat{b} &= b_i \\ c_4 &= c_4' \cdot \frac{1}{\epsilon}\log\frac{1}{\delta'} \end{aligned}$$

For any $0 \leq i \leq h$*, let* $\mathcal{F}_i$ *be the indicator random variable denoting the event of:*

$$(b_i/n_i \leq \epsilon/2) \text{ and } (b_i \geq c_4) \text{ and } (|\hat{b} - b| > \epsilon b)$$

Then for any $0 \leq i \leq h$*:*

$$Pr[\mathcal{F}_i] \leq \delta'$$

Proof: Obviously, $\hat{b} = b_i$ can never be larger than b, thus it suffices to prove that

$$Pr[(b_i/n_i \leq \epsilon/2) \text{ and } (b_i \geq c_4) \text{ and } (b - b_i > \epsilon b)] \leq \delta'$$

Consider the b balls, and define the indicator random variable X_j (for $1 \leq j \leq b$) to be 0 if and only if the jth ball goes into a previously empty bin. In other words, if $X_j = 1$, it means that the ball "collides" with some previous ball. Notice that X_j's are all correlated. Nevertheless, since at most $\epsilon/2$ fraction of the bins are occupied at any point of time, we always have $Pr[X_j] \leq \epsilon/2$. Define independent indicator random variables Y_j (for $1 \leq j \leq b$) where $Pr[Y_j] = \epsilon/2$ for all i. Let $X = \sum X_j$ and $Y = \sum Y_j$, and we will always have $X \leq Y$ (after properly defining the joint distribution). Next we invoke a standard Chernoff bound [19] on Y, and

for any $c_4' \geq 8$, we have:

$$\begin{aligned} Pr[\mathcal{F}_i] &\leq Pr[X > \epsilon b] \leq Pr[Y > \epsilon b] \\ &\leq exp(-\epsilon b/8) \leq \delta' \end{aligned}$$

□

LEMMA 14. *There exists a universal constant c_4' with the following property. Define*

$$\begin{aligned} \delta' &= \frac{\delta}{\log \max(4\epsilon^2 n, 2)} \\ \hat{b} &= b_i \\ c_4 &= c_4' \cdot \frac{1}{\epsilon} \log \frac{1}{\delta'} \end{aligned}$$

Let $\mathcal{F}_i$ be an indicator random variable denoting the event that there exists some level $0 \leq i \leq h$ on the sampling tree such that:

$$(b_i/n_i \leq \epsilon/2) \text{ and } (b_i \geq c_4) \text{ and } (|\hat{b} - b| > \epsilon b)$$

Then:

$$Pr[\mathcal{F}] \leq \delta$$

Proof: We applying Lemma 13 and set c_4' the same as the c_4' from Lemma 13. There are total $\log(4n)$ levels on the sampling tree, out of which at most $\log(4\epsilon^2 n)$ levels have at least $2c_4/\epsilon$ nodes. For the remaining levels, it is impossible for the events $b_i \geq c_4$ and $b_i/n_i \leq \epsilon/2$ to both occur. Thus we do not need to worry about them. Applying a union bound across the $\log(4\epsilon^2 n)$ levels then immediately proves the lemma. □

Comments on Lemma 14. Similar as for Lemma 11, here Lemma 14 bounds the probability that the tree sampling algorithm returns an estimate with large error when invoking the small occupancy bound.

Using both occupancy bound and small occupancy bound. It remains non-trivial to prove exactly how much improvement the small occupancy bound can help us to achieve. We first provide some intuitions. When we track down the tree starting from level α, intuitively r_i should decrease quickly. The reason is that n_i always doubles when i increments. But b_i is much less likely to double, given that some fraction of the bins at level α is already empty. Let β to be the highest level that satisfies $r_\beta \leq \frac{\epsilon}{2}$. The above intuition then suggests that $(\beta - \alpha)$ can be quite small. Further notice that for any level i below level β, we must have $r_i \leq \frac{\epsilon}{2}$ as well. Thus based on small occupancy bound, starting from level β, we can output $\hat{b} = b_i$ as long as $b_i \geq c_4 = \Theta(\frac{1}{\epsilon} \log \frac{1}{\delta'})$. This means that the number of samples taken for each level will be $O(\frac{1}{\epsilon} \log \frac{1}{\delta'})$. The total number of samples taken from level β to potentially the bottom is then $O(\frac{1}{\epsilon} \log \frac{1}{\delta'} \log \max(\epsilon^2 n, 2))$. (There are only $\log \max(\epsilon^2 n, 2)$ levels below level β.)

Reasoning about the number of samples needed to go from level α to level β is more complex, and Lemma 15 below proves that conditioned upon the binary search being successful, the number is $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log \frac{1}{\epsilon})$ with probability at least $1 - \delta$. For theoretical interest, one can further make this number deterministic by generating an arbitrary output if the number of samples taken exceeds $c\frac{1}{\epsilon^2} \log \frac{1}{\epsilon} \log \frac{1}{\delta'}$ for some constant c. Also, before tracking down the tree, the algorithm must have already exhaustively sampled level α and can tell whether the binary search was successful. If the binary search was not successful, the algorithm can simply stop and generate an arbitrary output. Finally, obviously $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log \max(\epsilon^2 n, 2))$ upper bounds the number of samples taken from level α to level β as well. This results in the term $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log \max(\frac{1}{\epsilon}, \epsilon^2 n))$.

LEMMA 15. *Conditioned upon the binary search being successful, the number of samples taken by tree sampling at Step 7, starting from level α and until the algorithm either reaches level β or returns (whichever occurs earlier), is $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log \frac{1}{\epsilon})$ with probability at least $1 - \delta$.*

Proof: Based on the occupancy bound, the algorithm takes $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'})$ samples for each level tracked. Thus it suffices to prove that $(\beta - \alpha)$ is $O(\log \frac{1}{\epsilon})$ with probability $1 - \delta$. However, this turns out to be true only when n_α is $\Omega(\frac{1}{\epsilon^2} \log \frac{1}{\delta'})$. A second complication is that both α and β are random variables, which can make the arguments a little convoluted.

To circumvent these problems, suppose that a successful binary search always returns a level α such that $0 < a' \leq r_\alpha \leq a < 1$. Here a' and a are constants and Theorem 6 guarantees the existence of a' and a. we define level γ to be the highest level in the tree with at least c_3/a' nodes, where c_3 is defined as in Lemma 11. Notice that γ is not a random variable. Lemma 16 will prove that there exists some universal constant c such that:

$$Pr\left[(b_\gamma/n_\gamma \leq a) \text{ and } \left(\beta - \gamma > c \log \frac{1}{\epsilon}\right)\right] < \delta$$

Conditioned upon the binary search being successful, we consider two cases. If $\alpha \geq \gamma$, then we must have $b_\alpha \geq a' \cdot n_\alpha \geq a' \cdot n_\gamma \geq c_3$. This means that b_α is already large enough to invoke the occupancy bound. The algorithm will then return a $\hat{b}$ at level α and will never need to track down the tree.

If $\alpha < \gamma$, we must have $b_\gamma/n_\gamma = r_\gamma \leq r_\alpha \leq a$. This means:

$$\begin{aligned} & Pr\left[\left(\beta - \gamma > c \log \frac{1}{\epsilon}\right)\right] \\ = \; & Pr\left[(b_\gamma/n_\gamma \leq a) \text{ and } \left(\beta - \gamma > c \log \frac{1}{\epsilon}\right)\right] \\ < \; & \delta \end{aligned}$$

This shows that tracking from level γ to level β will take $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'} \log \frac{1}{\epsilon})$ samples with probability at least $1 - \delta$. On the other hand, the total number of tree nodes between level α and level γ is at most $2n_\gamma = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta'})$. Thus

to track from level α to level β will take $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}) + O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\frac{1}{\epsilon}) = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\frac{1}{\epsilon})$ samples with probability at least $1-\delta$. □

LEMMA 16. *Consider any given constant $0 < a' < a < 1$. Define γ as in Lemma 15. We can always find universal constants c (which is uniquely determined by a' and a) such that*

$$Pr\left[(b_\gamma/n_\gamma \le a) \text{ and } \left(\beta - \gamma > c\log\frac{1}{\epsilon}\right)\right] \quad < \quad \delta$$

Notice that here b_γ and β are random variables, while γ, n_γ, and a are not.

Proof: Define:

$$\begin{aligned} c &= \frac{\log(2d)}{\log 2} + 1 \ge \frac{\log(2d)}{\log\frac{1}{\epsilon}} + 1 \ge \frac{\log\frac{2d}{\epsilon}}{\log\frac{1}{\epsilon}} \\ \beta' &= \gamma + c\log\frac{1}{\epsilon} \ge \gamma + \log\frac{2d}{\epsilon} \\ d &= \frac{\log(0.5 - 0.5a)}{\log 0.37} \end{aligned}$$

We consider two cases based on the value of b. First, if $b > d\cdot n_\gamma$, then applying the same technique as in Lemma 10 will show that $Pr[b_\gamma/n_\gamma \le a] < \delta$.

Second, for $b \le d \cdot n_\gamma$, we have:

$$\begin{aligned} n_{\beta'} &= n_\gamma \cdot 2^{\beta'-\gamma} \ge \frac{2d}{\epsilon}\cdot n_\gamma \\ b_{\beta'} &\le b \le d\cdot n_\gamma \\ r_{\beta'} &= b_{\beta'}/n_{\beta'} < \epsilon/2 \end{aligned}$$

We then deterministically have $\beta - \gamma \le \beta' - \gamma = c\log\frac{1}{\epsilon}$. □

THEOREM 17. *Conditioned upon the binary search being successful, with probability at least $1-\delta$, the number of samples taken by tree sampling at Step 7 is $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\min(\frac{1}{\epsilon}, \epsilon^2 n) + \frac{1}{\epsilon}\log\frac{1}{\delta'}\log\max(\epsilon^2 n, 2))$.*

Proof: First, the total number of samples taken from level β to potentially the bottom is $O(\frac{1}{\epsilon}\log\frac{1}{\delta'}\log\max(\epsilon^2 n, 2))$. Lemma 16 tells us that the number of samples taken from level α to level β is $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\frac{1}{\epsilon})$ with probability at least $1-\delta$. Add up the two terms completes the proof. □

Adversary's impact on performance guarantees. Finally, by always testing black after the binary search, the grey sensors may prevent the condition of $r_i \le \frac{\epsilon}{2}$ at Step 6.2 from being met quickly. This will not affect security but will increase the asymptotic number of samples taken. To preserve our previous performance guarantees, we know from Lemma 15 that once the total number of samples reaches $c\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\frac{1}{\epsilon}$ for some universal constant c, there must exist some correct $r_i' \le \frac{\epsilon}{2}$ (with probability at least $1-\delta$). Again from the sandwiching property, if we observe $r_i > \frac{\epsilon}{2}$, then there must exist a correct $r_i'' = \frac{\epsilon}{2}$. Thus all we need to do is to replace r_i with $\min(r_i, \frac{\epsilon}{2})$ at Step 6.2 once the total number of samples taken reaches $c\frac{1}{\epsilon^2}\log\frac{1}{\delta'}\log\frac{1}{\epsilon}$.