Attack-Resistant Frequency Counting

Bo Wu [†] Valerie King [*] Jared Saia [†]

Abstract

We present collaborative peer-to-peer algorithms for the problem of approximating frequency counts for popular items distributed across the peers of a large-scale network. Our algorithms are attack-resistant in the sense that they function correctly even in the case where an adaptive and computationally unbounded adversary causes up to a $1/3$ fraction of the peers in the network to suffer Byzantine faults. Our algorithms are scalable in the sense that all resource costs are polylogarithmic. Specifically, latency is $O(\log n)$; the number of messages and number of bits sent and received by each peer is $O(\log^2 n)$ per item; and number of neighbors of each peer is $O(\log^2 n)$. Our motivation for addressing this problem is to provide a tool for the following three applications: worm and virus detection; spam detection; and distributed data-mining.

To the best of our knowledge, our algorithms are the first attack-resistant and scalable algorithms for this problem. Moreover, surprisingly, our algorithms seem to be the first attack-resistant algorithms for *any* data mining problem.

1 Introduction

In this paper, we address the problem of robustly computing frequency counts for popular items that are distributed across the peers of a large network. Finding frequency counts in large-scale networks is a fundamental problem; an algorithm that solves this problem can be used for many types of applications including the following.

- *Worm and virus detection:* Techniques for automatic generation of worm signatures based on prevalence of portions of flow payloads have already been developed in systems such as Earlybird [34] and Autograph [21]. These systems work well in practice. In fact, Netsift, a company based on the technology of the Earlybird system, was acquired by Cisco systems in 2005 for $30 Million [31].[1] However, to best of our knowledge, there are currently no systems that can generate worm signatures *in a collaborative setting, when a large number of peers in the network are malicious.*

- *Spam detection:* The company Cloudmark maintains a system called Spamnet that is currently used by over two hundred million people [39]. Users that participate in this network mark those emails they receive that they consider spam; fingerprints are then generated for emails that are marked as spam by many users and these fingerprints are distributed to all users in the network.[2] Peer-to-peer systems have been proposed in the academic literature for collaborative spam detection [13, 6]. However, unfortunately, none of these systems are guaranteed to work correctly in the case where a large number of peers in the network are malicious.

[*]Department of Computer Science, University of Victoria; Victoria, BC; Canada V8W 3P6; Email: val@uvic.ca. This research was supported by an NSERC grant.

[†]Department of Computer Science, University of New Mexico, Albuquerque, NM 87131-1386; email: {bowu, saia}@cs.unm.edu. This research was partially supported by NSF CAREER Award 0644058, NSF CCR-0313160, and AFO MURI FA9550-07-1-0532 grant.

[1]Both Earlybird and Autograph make use of Rabin fingerprinting and the technique of breaking flow payloads into smaller strings, and are thus able to generate signatures for a large class of polymorphic worms, with few false positives and few false negatives.

[2]These fingerprints are robust in the sense that they can be used to identify messages that are slight variants of each other.

- *Data Mining:* Finding frequency counts has several important data mining applications including: finding frequent search queries [19, 7]; mining association rules [1]; enabling iceberg queries [5, 20]; and measuring IP traffic for detecting DoS attacks and hot spots [14]. Manku and Motwani give a comprehensive account of how an algorithm for computing approximate frequency counts can be used in the latter three applications [26].

Not surprisingly, the problem of approximating frequency counts has been studied extensively in the data mining community [7, 26, 27]. Moreover, there have been many results on maintaining frequency counts scalably in a distributed setting [25, 8, 10, 9]. However, to the best of our knowledge, there are no results on *fault-tolerant* algorithms for approximating frequency counts. Moreover, even more surprisingly, there do not seem to be any results for fault-tolerant algorithms for *any* data streaming problem. In particular, previous distributed data streaming results are neither tolerant to a certain number of peers dropping out of the network (fail-stop faults), nor tolerant to a collusion of malevolent peers that are trying to skew the results (Byzantine faults). To the best of our knowledge, our work is the first to address the problem of designing scalable data streaming algorithms that are tolerant to these two types of faults.

Our motivation for studying fault-tolerant peer-to-peer algorithms for frequency counts is threefold. First, in client-server solutions to this problem the company (e.g. Cisco or Cloudmark) charges the client, and this cost is a barrier to wide-spread use of the attack-suppression software. The need to charge seems unavoidable since the company must recoup the cost of the computational resources that it dedicates to solving the problem. Second, most companies do not have the same computational resources as can be illegally commandeered by cybercriminals: single botnets[3] have been reported to contain over $100,000$ machines [12]. A peer-to-peer approach can help overcome this mismatch. Finally, the company is a single point of vulnerability. For example, in May of 2006, several hundred junk e-mailers successfully launched an attack that disabled the websites of the anti-spam firm Blue Security [17].

In this paper, we focus on designing fully distributed, scalable and fault-tolerant algorithms to approximate frequency counts. For simplicity of exposition, we focus initially on solving this problem on a static network. Recent results by Scheideler and Awerbuch [32, 3] give general techniques that can be used to adapt our result to a dynamic network, i.e. one that may grow and shrink significantly in size. We also make use of the techniques in [32] to solve this problem on a dynamic network.

1.1 Problem Statement

We consider a model where data is distributed across n peers in a network and we wish to determine frequency counts for that data. The problem is complicated by the fact that a certain number of peers in the network are controlled by a malicious adversary. We assume this adversary is *essentially omniscient* in that it knows everything except for the random bits of the peers. In particular, it knows the topology of the network, the data on all peers, the content of all messages sent, and our algorithms. The adversary is *adaptive* in that it can delay the decision of which peers to take over. In particular, it may carefully watch the course of the algorithm, take over a critical set of peers at some point, wait for some more time, take over additional peers, and so forth until it has taken over its alloted number of peers. We note that an adaptive adversary is stronger than that considered in many recent results on scalable robust collaborative computation [22, 18, 23], but in the context of designing algorithms for large-scale, open access networks, we believe it is crucial to consider an adaptive adversary. Finally, we consider two scenarios for the peers taken over by the adversary. In the *fail-stop* fault model, these peers simply stop working; it is as if they are removed from the network. In the harder *Byzantine* fault model, these peers are all controlled by the adversary and behave in a malevolent way. We will refer to the peers taken over by the adversary as *bad* and the remaining peers as *good*.

We assume that communication among uncorrupted peers can only occur over edges in the network. Communication occurs in rounds. In each round, every peer may send out messages to its neighbors in the

[3]A botnet is a large set of machines that are taken over by a cybercriminal and then used to launch attacks.

network. The corrupted peers are assumed to have received the messages of all the uncorrupted peers before they send out their own messages. The peers are synchronized between rounds so that all messages in round i are assumed to be received before any messages in round $i+1$ are sent out.

We first note that it is impossible to ensure that all peers receive exact frequency counts in our model. In particular there are two main problems. First, since we require a sparse network, each good peer will have only a polylogarithmic number of neighbors. Thus the adversary may target a certain number of good peers, take over all neighbors of these peers, and thereby keep these targeted peers from learning *any* of the frequent items. In other words, it is unavoidable that *some* peers in the network cannot learn the correct output. Second, since our network is sparse and since we require the number of messages sent by each peer to be small, it is inevitable that some good peers will not have their input items counted. Thus, we are unable to guarantee that *any* peers receive the *exact* counts. Note that our result is similar in this way to the result by Manku and Motwani[26].

To address these two problems, we introduce two relaxation parameters ϵ and δ. Instead of requiring every peer to learn the frequent items, we require only all but an ϵ fraction of the peers to learn them. Instead of requiring frequencies to be learned exactly for all but this ϵ fraction of peers, we instead require frequencies to be learned only within a range of $1-\delta$ to $1+\delta$. We can now state our main problem below.

FindFrequentCounts $(S, C, M, t, \epsilon, \delta)$
Given: A set S of n peers; a fixed constant C; integers M and t, where $t/n < 1/3$; and positive reals ϵ and δ. Each peer in S initially has as input up to C items; an adaptive and omniscient adversary controls t of these peers. For an item v, let $c(v)$ be the number of good peers that have item v as an input.
Output: Each peer p outputs a set of items F_p with estimated frequencies $c_p(v)$ for all items v in F_p. For all but ϵn of the good peers, the following is true: 1) F_p contains all items v such that $c(v) \geq M$; 2) F_p contains no item v such that $c(v) < M-(1+\delta)t$; 3) Finally, for any item v in F_p, $c(v)-\delta t \leq c_p(v) \leq c(v)+(1+\delta)t$

Why Robust Distributed Hash Tables Fail: One naive approach to address the above frequency count problems is as follows. We maintain a robust distributed hash table (DHT) over all peers in a network. There are well-known algorithms that provably ensure the functionality of a DHT even in the case where up to a $1/3$ fraction of the peers joining the DHT are controlled by an adversary (see e.g. [3, 32]). Assume we are maintaining such a DHT. Let h be the function hashing items to peers in this distributed hash table. Each peer in the network performs a lookup, using h, for each item it holds. Then, for each item v, the peers at location $h(v)$ are responsible for maintaining counts of the number of unique times that item v has been looked up in the network; if the number of times item v appears is greater than M, these peers broadcast this fact to all the other peers in the network.

This naive scheme suffers from several problems. First, it suffers from load-balancing problems: those peers to which frequent items map may receive $\Theta(n)$ messages while peers to which unpopular items map will receive only $\Theta(1)$ messages. Second, it is vulnerable to attack by peer deletion: to prevent the reporting of some important frequent item v, an adversary need only take over the peers at location $h(v)$ in the DHT. Finally, it is vulnerable to an internal attack: malicious peers can collude to artificially inflate the counts of a very large number of items so that the set of items with a count above a certain threshold is too large to be useful.

To avoid this problem, we need a scheme that ensures: 1) the responsibility for counting any particular item is distributed throughout the network; 2) malicious peers are unable to artificially inflate the counts of benign items.

1.2 Our Results

In this paper, we describe efficient algorithms for solving the *FindFrequentCounts* problem. To the best of our knowledge, our algorithms for this problem are the first that are robust to adversarial attack.

Theoretical Results: Our main theoretical result is given in the following theorem.

Theorem 1.1 *With high probability*[4]*, we are able to solve the* FindFrequentCounts *problem, even in the case where an adaptive adversary causes a $1/3$ fraction of the peers in the network to suffer Byzantine faults. Moreover, the resource costs required by our algorithm are as follows:*

- *Our algorithm has latency $O(\log n)$.*
- *Every peer sends and receives $O(\log^2 n)$ messages.*
- *Every peer has $O(\log^2 n)$ neighbors in the network.*

We measure latency based on the maximum number of links that any message must traverse during the course of our algorithm. We note that in the case of fail-stop faults, we can reduce the number of messages each peer must send and receive from $O(\log^2 n)$ down to $O(\log n)$.

Our results depend critically on the existence of a certain type of butterfly network, where each peer of the butterfly network consists of a subset of $O(\log n)$ peers selected uniformly at random from the set of all peers. This type of network was first described by Fiat and Saia in [15], where it was used to enable secure storage and lookup of data items in a peer-to-peer network. Our contributions over this past result are two-fold. First, we present novel algorithms for using this network to perform *collaborative computation*; in contrast, the result from Fiat and Saia only used the butterfly network for communication. Thus, the distributed algorithms running on top of the network in this paper are completely different from the algorithms in [15]. Second, we design a different type of butterfly network: our new network has each supernode containing exactly the same number of peers, while the Fiat and Saia network requires only that each butterfly peer contain $\theta(\log n)$ peers. Showing that our new network is still robust requires different mathematical tools than from the previous paper.

Empirical Results: We also implement and simulate our algorithms on networks of sizes up to about $53,000$ peers. Our experiments show that we can ensure that almost all peers in the network learn the frequently occurring items with very high accuracy. Moreover, our algorithms for achieving this scale well in terms of bandwidth and latency costs.

The rest of our paper is structured as follows. We review related work in Section 1.3. We present our algorithm in Sections 2 and 3. Our empirical results are in Section 4. Finally, we conclude and give open problems in Section 5. All of our proofs along with variants on our main algorithm are included in the appendix.

1.3 Related Work

As previously mentioned, there has been significant work in the data mining community on counting frequent items [7, 26, 27], and there has also been significant work on the distributed version of this problem [25, 8, 10, 9]. However, to the best of our ability, our work is the first that enables attack-resistant counting of frequent items.

In the peer-to-peer literature, there have been many results on creating attack-resistant distributed hash tables (DHTs) [3, 32, 16, 28, 15]. Unfortunately, as mentioned previously, an attack-resistant DHT does not seem to have the functionality necessary to enable attack-resistant frequency counting. Some recent work has focused on designing robust peer-to-peer systems that solve basic algorithmic problems like Byzantine

[4]Throughout this paper, we will use the phrase with high probability (w.h.p.) to mean with probability $1 - 1/n^c$ for some constant $c > 0$

agreement and leader election [22, 18, 23]. However, these results are not useful for our problem because: 1) they are not robust to an adaptive adversary; and 2) they do not seem useable as building blocks for enabling frequency counts.

Two major motivations for our study of the frequency count problem are worm detection and spam detection. For worm and virus detection, there have been client-server approaches such as Earlybird [34] and Autograph [21] that are based on finding frequently occurring substrings in network traffic. In addition, several peer-to-peer systems have been proposed for collaborative detection of worms and viruses [11, 36, 24]. However, none of these systems are robust to an attack where an adversary takes over peers in the network. For spam detection, systems have been proposed for distributed detection of spam based on: classification and filtering of email that is spam [39, 13, 33, 38]; identification and blacklisting of users that frequently send spam [29]; counting and limiting the number of messages that any user can send [37, 4]. We believe that for all three of these approaches to dealing with spam, a robust tool for distributed frequency counting would be invaluable.

2 Our Algorithm

In this section, we describe our algorithm. For purposes of illustration, we present a naive version of our algorithm first, and then describe our more sophisticated algorithm in Section 2.2.

2.1 Naive Algorithm

A m input butterfly network is a directed graph with $\log m + 1$ levels, each with m nodes. An example is shown in Figure 1. For $i = 1, ..., m$ and $j = 0, ..., \log m$, node i on level j links to two nodes on level $j + 1$: node i and node $(i + 2^j)$ mod 2^j. In the simple version of our protocol, if there are m peers, then each peer i is assigned to position i on every level $i = 0, ..., \log m$ of the butterfly network.

We define $list_p$ for a peer p to be a list of items and a count of each item, i.e. a list of tuples of the form $\{(v, num_p(v)) \mid v$ is an item and $num_p(v)$ is an integer$\}$. Initially, each p is given a list of items, each with count 1. Our naive algorithm is presented below. It makes use of the following subroutine:

- $sum(list_p, list_q)$: returns $\{(v, num_p(v) + num_q(v)) \mid \forall v \; s.t. \; num_p(v) + num_q(v) > 0\}$.

Algorithm 1 FindApproxFrequent

```
1. Peer $j$ is assigned to position $j$ for all levels $i$.
2. For all level $i$, $0 \leq i \leq \log m$, of the butterfly, repeat
    (a) Transfer: Each peer sends its $list$ to its neighbors on level $i + 1$.
    (b) Check: If a list received from level $i$ contains a $num_v > 2^i$, set list to $\emptyset$.
    (c) Combine: Each peer on level $i + 1$ revises its list as follows: $list_p = sum(list_A, list_B)$ where
        $A$ and $B$ are its neighbors on level $i$.
```

2.1.1 Analysis

We now analyze this naive algorithm. We will make use of the following definitions in our proof:

Definition 2.1 *A top node is a level 0 node; a bottom node is a level* $\log m$ *node.*

Definition 2.2 *A node is good if the peer assigned to it is good.*

Definition 2.3 *A good path is a path through the butterfly network from a top node to a bottom node all of whose nodes are good. A path that is not good is bad.*

Definition 2.4 *For any fixed* $\gamma > 0$*, a bottom node is* γ*-expansive if there exist* γm *good paths that end at this node.*

Observation 2.5 *In the butterfly network,*

- *Each node is on exactly m simple paths from the top level to the bottom level.*
- *There is exactly one simple path from a top node to a bottom node.*
- *Each simple path from a top node to a bottom node p contributes either 0 or 1 to $num_p(v)$.*
- *If a good top node contains v and the path from it to a bottom node p contains all good nodes, then this path contributes 1 to $num_p(v)$.*

Suppose there are fewer than t' bad nodes in the network, then fewer than $t'm$ paths contain bad peers. Hence, for any positive $\delta \leq 1$, no more than δm good peers at the bottom level receive more than t'/δ messages passed along bad paths. The following lemma follows easily from Observation 2.5.

Lemma 2.6 *In a standard butterfly network, if the frequency of an item v is $c(v)$ then there are $(1-\delta)m-t'$ good peers that can determine a frequency of v in the range $[c(v)-t'/\delta, c(v)+t'/\delta]$.*

2.2 Our Improved Algorithm

We now describe a way to improve the accuracy of the naive algorithm by using supernodes to reduce the number of corrupt network nodes in a network with n peers. We construct a new butterfly network with $m = n/\log n$ nodes on each level and $\log n - \log\log n$ levels, where each node of the butterfly network is replaced by a set of peers of size $s = k \log n$ called a *supernode*. The following claim is proven in Appendix A.

Claim 2.7 *For any constant δ, there is a constant k such that, the assignment of peers to supernodes has the following properties:*

1. *Each supernode contains $k \log n$ peers.*
2. *In the top and bottom levels, each peer is in exactly k supernodes.*
3. *In the middle levels, all but $\delta^2 t/\log n$ supernodes have a majority of uncorrupt peers.*

The protocol for the supernode network is similar to the algorithm for the standard butterfly network except that there is an initialization process and a modification to the *Transfer* step. In addition, as a peer may appear in several supernodes in the same level, it is important to remember that until the *Final* step, its actions in one supernode on a level are separate and independent of its actions in other supernodes on the same level.

- $Initialize\_total$: When the messages from level 0 supernodes are received, each peer in a level 1 supernode sums up the numbers received for each item, discarding any values not equal to 0 or 1.

For subsequent levels, each peer p in each node constructs its own list for that node, then passes it on. The following subroutine is used to compile lists passed on by the set of peers in a supernode:

- $median(A, i)$: given a collection of lists sent by peers in supernode A, $\{list_p \mid p \in A\}$, this function returns the list f in which $num_f(v)$ is the median value of $num_p(v)$ over all $p \in A$. The function sets to 0 any input values in which $num(v) > 2^i s$, where s is the number of peers in a supernode.

The *Transfer* step 2a is thus modified:

- *Transfer:* Each peer sends its list to every peer in the supernode's neighbors on level $i+1$. Let C be a supernode on level $i+1$ and let A and B be its neighbors on level i. Each peer in C computes $list_A$ and $list_B$, respectively, by applying $median$ to each set of lists sent by peers in A and B, respectively.

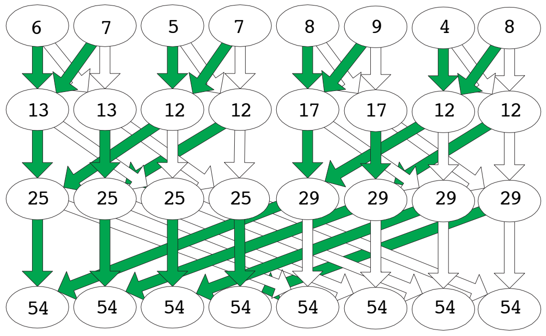


Figure 1: The butterfly network and example run of Algorithm 2. The numbers are the frequency counts for a single item. In this example, there are no adversarial faults, so the bottom nodes have the exact sum of all counts at the top nodes.

A *Final* step is added, in which the various roles of a peer are brought together:

- *Final Step:* For each item v, each peer divides $num(v)$ by k. As a single peer may have several values for $num(v)$ depending on which supernode it is in, it selects the median value as its final value.

Our full algorithm is given below as Algorithm 2. Figure 1 illustrates a run of this algorithm. The numbers in the figure give an example of how frequency counts might be computed for a single item. For ease of presentation, the example in the figure assumes no Byzantine faults.

Algorithm 2 FindApproxFrequent (with Supernodes)

```
1. Each peer is assigned to a supernode in such a way that Claim 2.7 is satisfied (details of how this is
   done are described in Section 3).

2. $Initialize\_total:$ Each peer in a level 0 supernode sends its list to every peer in the supernode's
   neighbors on level 1. When the messages from level 0 supernodes are received, each peer in a level 1
   supernode sums up the numbers received for each item , discarding any values not equal to 0 or 1.

3. For all level $i$, $1 \leq i < \log m$, of the butterfly network, repeat
   (a) Transfer: Each peer sends its list to every peer in the supernode's neighbors on level $i+1$. Let
       $C$ be a supernode on level $i+1$ and let $A$ and $B$ be its neighbors on level $i$. Each peer in $C$
       computes $list_A$ and $list_B$, respectively, by applying $median$ to each set of lists sent by peers in
       $A$ and $B$, respectively.
   (b) Check: If a list received from level $i$ contains a $num_v > 2^i s$, set list to $\emptyset$.
   (c) Combine: Each peer on level $i+1$ revises its list as follows: $list_p = sum(list_A, list_B)$ where
       $A$ and $B$ are its neighbors on level $i$.

4. Final Step: For each item $v$, each peer divides $num(v)$ by $k$. As a single peer may have several values
   for $num(v)$ depending on which supernode it is in, it selects the median value as its final value.
```

2.2.1 Analysis of the Protocol Using Supernodes

We provide a brief sketch of the analysis of our protocol by presenting key definitions and lemmas. All proofs are included in Appendix A.

Definition 2.8 *A supernode is good if it has at least a majority of good peers.*

Definition 2.9 *A good path is a path through the butterfly network from a top supernode to a bottom supernode all of whose supernodes are good supernodes.*

Definition 2.10 *For any fixed $\gamma > 0$, a bottom supernode is γ-expansive if there exist $\gamma n/\log n$ good paths that end at this supernode.*

Observation 2.11 *If the majority of peers in two supernodes transfer num's in the range $[a, b]$ and $[c, d]$ to the peers in some supernode C, then the good peers in C have num's in the range $[a + c, b + d]$.*

The following lemmas hold with high probability in a our network with n peers, where the adversary controls $t \leq n/3$ peers. Recall that $s = k \log n$ is the number of peers in each supernode in our network.

Lemma 2.12 *For any $\delta > 0$, for k and n sufficiently large, all but $\delta n/\log n$ bottom supernodes are $(1 - (\delta t/n))$-expansive with probability $1 - O(1/n^2)$.*

Lemma 2.13 *Let δ, ϵ be fixed constants where $0 < \delta < 1/2$, and $\epsilon > 0$. Let the number of bad supernodes in middle levels be no more than $\delta^2 t/\log n$; let all but $\delta n/\log n$ bottom supernodes be $(1 - (\delta t/n))$-expansive; and let k and n be sufficiently large. Then if the frequency of item v is $c(v)$ we can ensure that there are $(1-\epsilon)(n-t)$ good peers that can determine a frequency of v in the range $[c(v)-\delta t, c(v)+(1+\delta)t]$.*

3 The Existence of the Supernode Network

The static construction of the network is as follows. In Appendix B, we show how the network can be dynamically maintained.

- Let t be the number of bad peers in the network and $t < n/3$. Let M be the threshold such that one item can be considered as frequent item only if its number is greater than M, where $M > t$. We choose the parameters $0 < \epsilon < 1$ and $0 < \delta < 1/2$. As a function of ϵ, δ, we determine constant k. (See Theorem 1.1).
- Use random permutation to construct a random bipartite graph G_{top} with n peers on the left side and $n/\log n$ top supernodes on the right side such that each peer has k neighbors on the right side and each top supernode has $k \log n$ neighbors on the left side.
- Use random permutation to construct a random bipartite graph G_{bottom} with n peers on the left side and $n/\log n$ bottom supernodes on the right side such that each peer has k neighbors on the right side and each bottom supernode has $k \log n$ neighbors on the left side.
- Use random permutation to construct a random $k \log n$-regular bipartite graph G_{middle} with n peers on the left side and n middle supernodes on the right side such that each peer has $k \log n$ neighbors on the right side and each top supernode has $k \log n$ neighbors on the left side.
- Between two sets of peers associated with two supernodes connected in the butterfly network, we will have a full bipartite graph between the peers of these two supernodes.

After the above construction, we can see that in the top and bottom levels, each peer is in exactly k supernodes, and each supernode contains $k \log n$ peers.

4 Experiments

In this section, we present empirical results for our algorithm. Our goal is to measure the tradeoff between (1) resource costs: namely, total bits sent and latency; and (2) accuracy: the fraction of frequent items that are not properly identified as such.

4.1 Experimental Setup

Our focus is on data streams where frequency counts follow a Zipf distribution, i.e. the frequency of the i-th most popular item occurs is proportional to i^{-z} for some fixed value z. Numerous studies on real-world data show that frequency counts typically follow such a distribution, with the exponent z usually in the range from 2 to 3. In our experiments, we consider values of z ranging from 1 to 5. In every experiment, we set the threshold M, that determines whether or not an item is frequent to be $t+1$. This is the smallest threshold possible, since if $M \leq t$, it is possible for the adversarially controlled peers to overwhelm the network by generating essentially an unbounded number of frequent items.

In all of our experiments, we consider the case where the total number of *distinct* items is $10n$, where n is the number of peers in the network, and we consider values of n up to $53,248$. To illustrate, consider the case where $n = 53,248$ and $z = 2$. Then, the number of times the k-th most frequent item occurs is $\frac{(1/k^s)}{\sum_{i=1}^{53248}(1/i^z)}$, where $\sum_{i=1}^{53248}(1/i^z)$ is approximately $\pi^2/6$. Thus, the frequency with which the *most* popular item occurs is about 61%, the frequency of the second most popular item is about 15%, and the frequency of the third most popular item is about 6.8%.

Unfortunately, for typical values of z, the number of items that occur a large number of times is very small. Thus, to more effectively test our algorithms, we add an additional $1,000$ "hidden" items that occur with frequency distributed uniformly at random between M and n, and which are distributed uniformly at random on the peers in the network. These hidden items could represent spam or the signature of an epidemic process. Our experiments were performed using a simple variant of Algorithm 2, described in Appendix E.1, that reduces the total number of bits sent via a thresholding function. The basic idea is to throw out items with frequency counts that fall below a certain threshold, where the threshold increases exponentially as the messages proceed down the butterfly network. We emphasize that although the total number of distinct items is only $10n + 1000$, the number of occurrences of items is actually much higher, since a single distinct item can occur on a large number of peers.

We assume the following strategy for the adversary. The adversary takes over supernodes starting at the top left of the butterfly network and moving right and then down. While the adversary still has peers that it can take over, it takes over a majority of the peers in each supernode in order to control the entire supernode. In this way, the adversary will be able to control at least $O(n/\log n)$ supernodes. We experimented with various other simple adversarial strategies (e.g. targeting random peers, targeting random supernodes) and this seemed to be the most effective one for the adversary.

4.2 Results

Our empirical results are shown in Figures 2 and 3. Both plots in Figure 2, give results for Algorithm 2 with networks of size $n = 53,248$; in both plots the y-axis measures the fraction of frequent items that are not correctly identified as such by the algorithm. The left plot gives results when $k = 4$ (recall that k is the robustness parameter for Algorithm 2) as the exponent z of the Zipf distribution varies from 1 to 5. The fraction of nodes controlled by the adversary, t, forms the contours in the left plot. As expected, higher values of z result in fewer errors, all else being equal. Moreover, our algorithm seems to do quite well even for moderately high values of z. The right plot gives results for $z = 2$, when robustness parameter, k, varies on the x-axis. Again different values of t form the contours in the plot. We note that $z = 2$ is smaller than we might expect to see in practice. However, with larger values of z, our experiments showed that essentially no frequent items are missing, so we used a smaller value of z to show the tradeoff between k and the error rate.

Both plots in Figure 3 give results for Algorithm 2 with $t = n/8$ and $z = 2$. The left plot shows how the total number of bits sent varies with n (x-axis), and k (contours). Without thresholding, for $n = 53,248$, our analysis predicts that the number of bits sent by a node would be roughly $48 * 532480 * k^2 \log^2 n$. This follows since the maximum frequency for an item is n, and we assume each items needs 32 bits to be

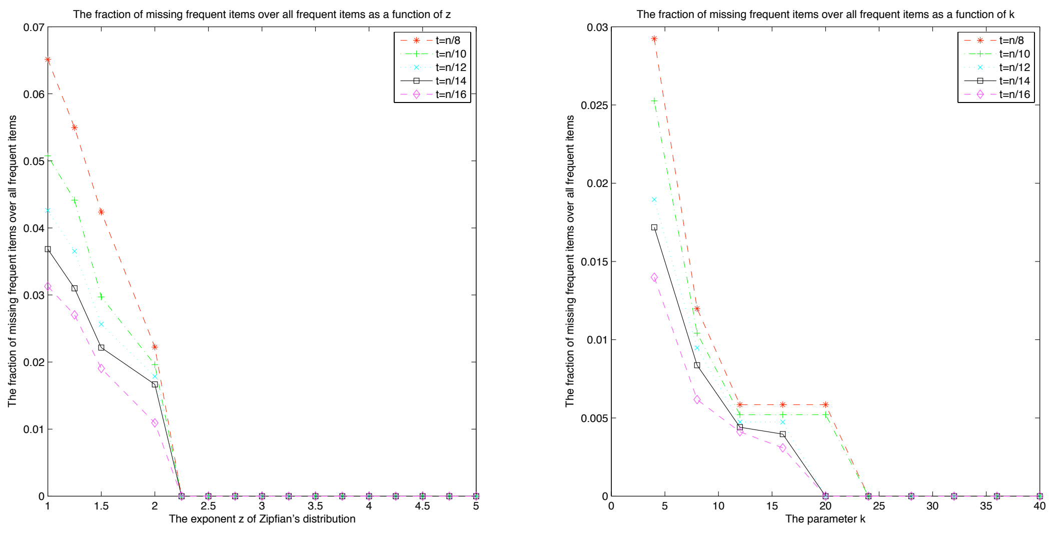


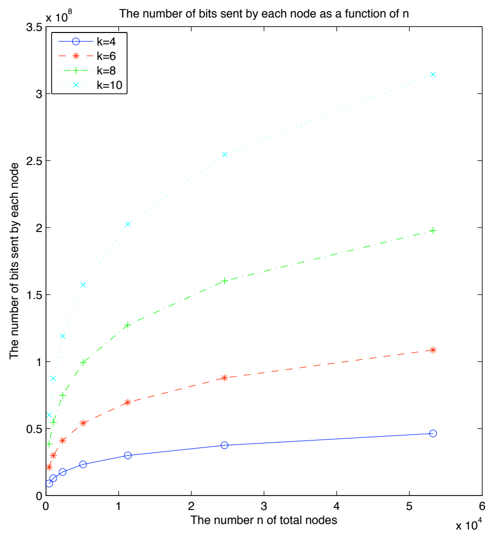
Figure 2: Error Rates for Algorithm 2

identified, so the total bits to identify and count an item is $\log n + 32 = 48$. This is multiplied by the total number of items that are counted and by the $k^2 \log^2 n$ factor that comes from our algorithm. For $k = 4$, the above formula predicts about $9 * 10^9$ bits will be sent without thresholding, while our experiments show about $5 * 10^7$ bits or about 6 megabytes being sent per processor with thresholding. By way of comparison, a one hour voice call on Skype requires from 10 to 56 megabytes [35]. In some situations, the bandwidth cost of our algorithm may be significant. Some techniques to further reduce bandwidth, at the expense of decreased accuracy, are discussed in Appendix E. The right plot shows the relationship between the latency of our algorithm (number of hops) and the parameter n. The right plot is exactly same for different k. For the right plot, our theoretical results predict a latency that is logarithmic in network size and this is also approximately matched by our empirical results.

5 Conclusion and Future Work

We have described distributed algorithms for approximating frequency counts of popular items in a large-scale peer-to-peer network. Our algorithms are attack-resistant in the sense that they functions correctly even if a malicious adversary controls up to a constant fraction of the peers in the network. Our algorithms are also scalable in terms of latency and bandwidth costs. We have proven that our algorithms are correct, and have presented empirical evidence showing that their resource costs scale well.

Several open problems remain including the following. First, can we further reduce bandwidth costs without sacrificing accuracy? Second, can use the techniques described in this paper for designing robust distributed algorithms for other computational problems? Finally, would it be possible to deploy these algorithms to solve the problem of approximating frequency counts in an actual distributed network?



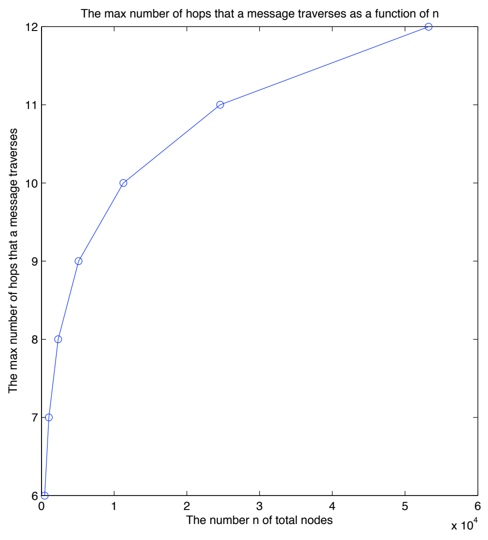


Figure 3: Resource Costs for Algorithm 2

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the International Conference on Very Large Databases*, 2002.

[2] Noga Alon and Joel Spencer. *The Probabilistic Method, 2nd Edition*. John Wiley & Sons, 2000.

[3] B. Awerbuch and C. Scheideler. Towards a Scalable and Robust DHT. In *Symposium on Parallel Algorithms and Architecture (SPAA)*, 2006.

[4] Hari Balakrishnan and David Karger. Spam-I-am: A Proposal for Spam Control using Distributed Quota Management. In *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[5] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1999.

[6] P. Oscar Boykin and Vwani P. Roychowdhury. Leveraging social networks to fight spam. *IEEE Computer*, 38(4):61–68, Apr 2005. cond-mat/0402143.

[7] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the International Colloquium on Automata, Languages and Programming(ICALP)*, 2002.

[8] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the International Conference on Very Large Data Bases Conference (VLDB)*, 2005.

[9] G. Cormode and M. Garofalakis. Streaming in a connected world: Querying and tracking distributed data streams. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2006. Tutorial Session.

[10] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *Proceedings of the International Conference on Management of Data(SIGMOD)*, 2005.

[11] Manuel Costa, Jon Crowcroft, Miguel Castro, Anthony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: End-to-end containment of internet worms. In *Symposium on Operating System Principles (SOSP)*, 2005.

[12] Drew Cullen. Dutch smash 100,000-strong zombie army, 2005. The Register.

[13] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. P2p-based collaborative spam detection and filtering. In *Proceedings of the Fourth International Conference on Peer-to-peer Computing*, 2004.

[14] C. Estan and G. Verghese. New directions in trafc measurement and accounting: Focusing on the Elephants, Ignoring the Mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[15] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the Thirteenth ACM Symposium on Discrete Algorithms (SODA)*, 2002.

[16] Amos Fiat, Jared Saia, and Maxwell Young. Making Chord Robust to Byzantine Attacks. In *Proceedings of the European Symposium on Algorithms(ESA)*, 2005.

[17] Joanna Glasner. I'm the Blue Security Spammer, May 2006. Wired Magazine.

[18] Shafi Goldwasser, Elan Pavlov, and Vinod Vaikuntanathan. Fault-tolerant distributed computing in full-information networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 15–26, 2006.

[19] Google. Google zeitgeist - search patterns, trends, and surprises according to google. http://www.google.com/press/zeitgeist.htm.

[20] J. Han, J. Pei, G. Dong, and K. Wang. Bottom-up computation of sparse and iceberg cubes. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2001.

[21] H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the 13th Usenix Security Symposium (Security 2004)*, 2004.

[22] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the Symposium on Discrete Algorithms(SODA)*, 2006.

[23] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Towards secure and scalable computation in peer-to-peer networks. In *To Appear in Foundations of Computer Science(FOCS)*, 2006.

[24] David J. Malan and Michael D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *ACM Workshop on Rapid Malcode*, 2005.

[25] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2005.

[26] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the International Conference on Very Large Databases Conference(VLDB)*, 2002.

[27] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers, Inc, 2005.

[28] Moni Naor and Udi Wieder. A simple fault tolerant distributed hash table. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[29] Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2007.

[30] Jared Saia and Maxwell Young. Reducing communication costs in robust peer-to-peer networks. *Information Progressing Letters*, 2007.

[31] Peter Sayer. Cisco buys Netsift, June 2005. Infoworld.

[32] C. Scheideler. How to Spread Adversarial Nodes? Rotate! In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing(STOC)*, 2005.

[33] Richard Segal. Combining global and personal anti-spam filtering. In *Proceedings of the 4th Conference on Email and Anti-Spam(CEAS2007)*, 2007.

[34] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[35] skype. https://support.skype.com/faq/fa151/how-much-bandwidth-does-skype-use-while-i-m-in-a-call.

[36] V. Vlachos, S. Androutsellis-Theotokis, and D. Spinellis. Security applications of peer-to-peer networks. *Computer Networks*, 45(2):195–205, 2004.

[37] Michael Walfish, J.D. Zamfirescu, Hari Balakrishnan, David Karger, and Scott Shenker. Distributed quota enforcement for spam control. In *Proceedings of 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[38] Zhe Wang, William Josephson, Qin Lv, Moses Charikar, and Kai Li. Filtering image spam with near-duplicate detection. In *Proceedings of the 4th Conference on Email and Anti-Spam(CEAS2007)*, 2007.

[39] Cloudmark Website. http://cloudmark.com/.

APPENDIX

A Proofs

A.1 Proof Overview

Technically, our proofs make extensive use of random constructions and the Probabilistic Method[2]. The overview of the proof is as follows. First, in Section A.2, we show that for an arbitrarily small constant δ, all but $\delta^2 t/\log n$ supernodes are good with high probability. Second, we show that all but $\delta n/\log n$ bottom supernodes are $(1-(\delta t/n))$-expansive with high probability in Section A.3. This implies that each of these expansive bottom supernodes has at least $(1-(\delta t/n))(n/\log n)$ good paths starting from $(1-(\delta t/n))(n/\log n)$ good top supernodes. Third, in Section A.4, we show that with high probability, at least $(1-\epsilon)(n-t)$ good peers appear in these $(1-(\delta t/n))(n/\log n)$-expansive bottom supernodes at least $k/2$ times. Finally, we show that with high probability, for any item v, if the frequency of v in the network is $c(v)$, at least $(1-\epsilon)(n-t)$ good peers can determine the frequency of the item v in the range $[c(v)-\delta t, c(v)+(1+\delta)t]$.

A.2 Good Supernodes

The three lemmas in this section establish Claim 2.7. In fact, for technical reasons, they establish a slightly stronger result.

Lemma A.1 *Let n be the number of peers. Let $t < n/3$ be the number of bad peers. Let δ, k be constants where $0 < \delta < 1/2$. Consider the random bipartite graph $G_{top}(G_{bottom})$ with n peers on the left side L and $n/\log n$ top(bottom) supernodes on the right side R constructed in Section 3. Each peer participates in k random top(bottom) supernodes and each top(bottom) supernode contains exactly $k\log n$ peers in it. Then with probability at least $1-1/n^2$, all but $\delta^2 t/(3\log n)$ top(bottom) supernodes have at least $(2/3)k\log n$ good peers.*

Proof. Let $|L| = l$, $|R| = r$. Let L' be the set of bad peers, R' be the set of top(bottom) supernodes with at least $(k/3)\log n$ bad peers, where $|L'| = l'$ and $|R'| = r'$. Thus $l = n$, $l' = t$ and $r = n/\log n$. Assume $r' = \delta^2 t/(3\log n)$, then we will fix a set $L' \subset L$ of size l' and a set $R' \subset R$ of size r' and compute the probability that all peers in R' have more than $k\log n/3$ neighbors in L'. If this occurs then the total number of edges shared between R' and L' must be more than $(\delta^2 tk)/9$. To do this, we will make use of the random graph $G_{top}(G_{bottom})$ defined in the following way.

The process of constructing such a random bipartite graph $G_{top}(G_{bottom})$ is equivalent to the process of constructing a random 1-regular bipartite graph $G'_{top}(G'_{bottom})$ with two set of peers A and B, each of size $n' = Cn$ (making each peer C replicas and each top(bottom) supernode $C\log n$ replicas).

The construction of graph $G'_{top}(G'_{bottom})$ is as follows. Define a set, $X_1, X_2, \ldots, X_{n'}$, of independent random variables which determine the matching. For all i between 1 and n', X_i takes on an integer value between i and n', each with equal probability. These random variables determine a permutation in the following way. Start with the ordered list $1, 2, \ldots, n'$, then perform the following swaps on the list: For each i between 1 and n', swap the element at the i-th position in the list with the element at the X_i-th position in the list. Let the resulting permutation be $P = (p_1, p_2, \ldots, p_{n'})$. The matching is built by creating an edge between each peer $i \in B$ and peer $p_i \in A$.

In the random 1-regular bipartite graph $G'_{top}(G'_{bottom})$ with two sets of peers A and supernodes B, we let A' be the set of peers in $G'_{top}(G'_{bottom})$ that corresponds to the set L' in $G_{top}(G_{bottom})$ and B' be the set of peers in $G'_{top}(G'_{bottom})$ that corresponds to the set R' in $G_{top}(G_{bottom})$.

Let X be a random variable giving the number of edges shared between L' and R', then X is also the random variable giving the number of edges shared between A' and B'. We consider the following two cases.

1. Case 1: $t = o(n)$.
   Consider the probability that a single edge falls from B' in A'. Since there are always at most kt peers in A that are bad, and there are always at least $kn - k\log n(\delta^2 t/(3\log n)) = kn - (\delta^2 kt)/3$ peers left in A that can be chosen by the peers in B', the probability that a single edge falls from B' in A' is less than

$$\frac{kt}{kn - \frac{\delta^2 tk}{3}} = \frac{3t}{3n - \delta^2 t} \leq \frac{9t}{8n}.$$

   The inequality in the above follows since $t < n/3$ and $0 < \delta < 1/2$.
   The total number of all edges from B' is

$$\frac{\delta^2 t}{3\log n} k\log n = \frac{\delta^2 kt}{3}.$$

The number of ways to choose a set of more than $\delta^2 kt/9$ edges from $\delta^2 kt/3$ is

$$\begin{pmatrix} \frac{\delta^2 kt}{3} \\ \frac{\delta^2 kt}{9} \end{pmatrix} \leq (3e)^{\frac{\delta^2 kt}{9}}.$$

Thus we can say that:

$$\begin{aligned} \Pr\left(X \geq \frac{\delta^2 kt}{9}\right) &\leq (3e)^{\frac{\delta^2 kt}{9}} \left(\frac{9t}{8n}\right)^{\frac{\delta^2 kt}{9}} \\ &= \left(\frac{8n}{27et}\right)^{-\frac{\delta^2 kt}{9}}. \end{aligned}$$

The number of ways to choose a set L' of the appropriate size is no more than $(le/l')^{l'}$ and the number of ways to choose a set R' of the appropriate size is no more than $(re/r')^{r'}$. So the probability that for any set L' of size t, there exists one subset R' of size $\delta^2 t/(3\log n)$ such that all supernodes in R' share more than $k\log n/3$ neighbors in L' is no more than

$$\left(\frac{le}{l'}\right)^{l'} \left(\frac{re}{r'}\right)^{r'} \left(\frac{n}{27et}\right)^{-\frac{\delta^2 kt}{9}} = \left(\frac{ne}{t}\right)^{t} \left(\frac{3ne}{\delta^2 t}\right)^{\frac{\delta^2 t}{3\log n}} \left(\frac{8n}{27et}\right)^{-\frac{\delta^2 kt}{9}}.$$

Below we solve for appropriate k such that this probability is less than $1/n^2$:

$$\left(\frac{ne}{t}\right)^{t} \left(\frac{3ne}{\delta^2 t}\right)^{\frac{\delta^2 t}{3\log n}} \left(\frac{8n}{27et}\right)^{-\frac{\delta^2 kt}{9}} \leq \frac{1}{n^2}.$$

Isolating k in the above equation, we get that:

$$k \geq \frac{9}{\delta^2 \log\left(\frac{8n}{27et}\right)} \left(\log\left(\frac{ne}{t}\right) + \frac{\delta^2}{3\log n}\log\left(\frac{3ne}{\delta^2 t}\right) + \frac{2\log n}{t}\right).$$

Since $t < n/3$, the right side of the above inequality is no more than a fixed constant k_0, then k can be a constant greater than or equal to k_0.

2. Case 2: $t = \Theta(n)$
The probability that a single edge from L' falls in R' is $r'/r = \delta^2 t/(3n)$ so by linearity of expectation, $E(X) = r'l'k/r = (\delta^2 kt^2)/(3n)$. In graph G', let g be a function such that $X = g(X_1, X_2, \ldots, X_{|B|})$. We note that g satisfies the Lipchitz condition, i.e. $|g(X_1, X_2, \ldots, X_j, \ldots, X_{|B|}) - g(X_1, X_2, \ldots, X_{j'}, \ldots, X_{|B|})| \leq 1$.
Hence, we can use Azuma's Inequality to say that for any fixed positive λ:

$$\Pr(X - E(X) \geq \lambda E(X)) \leq e^{-\frac{\lambda^2 E^2(X)}{2|B|}}.$$

Let

$$\lambda = \frac{n - 3t}{3t},$$

then

$$\begin{aligned} \Pr(X \geq \frac{\delta^2 kt}{9}) &= \Pr(X - E(X) \geq \lambda E(X)) \\ &\leq e^{-\frac{(n-3t)^2}{9t^2}\frac{\delta^4 t^4 k^2}{9n^2}\frac{1}{2kn}} \\ &= e^{-\frac{\delta^4 kt^2 (n-3t)^2}{162n^3}}. \end{aligned}$$

The number of ways to choose a set L' of the appropriate size is no more than $(le/l')^{l'}$, and the number of ways to choose a set R' of the appropriate size is no more than $(re/r')^{r'}$.

So the probability that for any set L' of size t, there exists one subset R' of size $\delta^2 t/(3\log n)$ such that all supernodes in R' share more than $k\log n/3$ neighbors in L' is no more than

$$\left(\frac{le}{l'}\right)^{l'}\left(\frac{re}{r'}\right)^{r'} e^{-\frac{\delta^4 kt^2(n-3t)^2}{162n^3}} = \left(\frac{ne}{t}\right)^{t}\left(\frac{3ne}{\delta^2 t}\right)^{\frac{\delta^2 t}{3\log n}} e^{-\frac{\delta^4 kt^2(n-3t)^2}{162n^3}}.$$

Below we solve for appropriate k such that this probability is less than $1/n^2$

$$\left(\frac{ne}{t}\right)^{t}\left(\frac{3ne}{\delta^2 t}\right)^{\frac{\delta^2 t}{3\log n}} e^{-\frac{\delta^4 kt^2(n-3t)^2}{162n^3}} \leq \frac{1}{n^2}.$$

Isolating k in the above equation, we get that

$$k \geq \frac{162n^3}{\delta^4 t^2(n-3t)^2}\left(t\log\left(\frac{ne}{t}\right) + \frac{\delta^2 t\log\left(\frac{3ne}{\delta^2 t}\right)}{3\log n} + 2\log n\right).$$

Since $t < n/3$ and $t = \Theta(n)$, the right side of the above inequality is no more than a fixed constant k_0, then k can be a constant greater than or equal to k_0.

Thus with probability at most $1/n^2$, after controlling any set L' of t peers, there exists one subset R' of size $\delta^2 t/(3\log n)$ such that all supernodes in R' have more than $k\log n/3$ neighbors in L'. Thus with probability at least $1 - 1/n^2$, controlling any set of t peers by the adversary still leaves all but $\delta^2 t/(3\log n)$ top(bottom) supernodes all with at least $2k\log n/3$ good peers.

□

Lemma A.2 *Let n be the number of peers. Let t be the number of bad peers. Let δ, k be constants where $0 < \delta < 1/2$. Consider the random k-bipartite graph G_{middle} with n peers on the left side L and n middle supernodes on the right side R constructed in Section 3: each peer participates in k random middle supernodes and each middle supernode contains exactly $k\log n$ peers in it. Then with probability at least $1 - 1/n^2$, controlling any set of t peers by adversary still leaves all but $\delta^2 t/(3\log n)$ middle supernodes with at least $k\log n/2$ good peers.*

Proof. Let $|L| = l$, $|R| = r$. Let L' be the set of bad peers, R' be the set of bad supernodes with at least $k\log n/2$ bad peers, where $|L'| = l'$ and $|R'| = r'$. Thus $l = n$, $l' = t$ and $r = n$. Assume $r' = \delta^2 t/(3\log n)$, then we will first fix a set $L' \subset L$ of size l' and a set $R' \subset R$ of size r' and compute the probability that all peers in R' have more than $k\log n/2$ neighbors in L'. If this occurs then the total number of edges shared between R' and L' must be more than $(\delta^2 tk)/6$. To do this, we will make use of the random graph G_{middle} defined in the following way.

The process of constructing such a random $C\log n$-regular bipartite graph G_{middle} is equivalent to the process of constructing a random 1-regular bipartite graph G'_{middle} with two sets of peers A and B, each of size $n' = Cn\log n$(make each peer $C\log n$ replicas and each middle supernode $C\log n$ replicas).

The construction of graph G'_{middle} is as follows. Define a set, $X_1, X_2, \ldots, X_{n'}$, of independent random variables which determine the matching. For all i between 1 and n', X_i takes on an integer value between i and n', each with equal probability. These random variables determine a permutation in the following way. Start with the ordered list $1, 2, \ldots, n'$, then perform the following swaps on the list: For each i between 1 and n', swap the element at the i-th position in the list with the element at the X_i-th position in the list. Let the resulting permutation be $P = (p_1, p_2, \ldots, p_{n'})$. The matching is built by creating an edge between each peer $i \in B$ and peer $p_i \in A$.

In the random 1-regular bipartite graph G'_{middle}, we let A' be the set of peers in G'_{middle} that corresponds to the set L' in G_{middle} and B' be the set of peers in G'_{middle} that corresponds to the set R' in G_{middle}.

Let X be a random variable giving the number of edges shared between L' and R', then X is also the random variable giving the number of edges shared between A' and B'. Now we consider the following two cases.

1. Case 1: $t = o(n)$.
   Consider the probability that a single edge falls from B' in A'. Since there are always at most $kt\log n$ peers in

A that are bad, and there are always at least $kn \log n - k \log n(\delta^2 t/(3 \log n)) = kn \log n - (\delta^2 kt)/3$ peers left in A that can be chosen by the peers in B', the probability that a single edge falls from B' in A' is less than

$$\frac{kt \log n}{kn \log n - \frac{\delta^2 tk}{3}} = \frac{3t \log n}{3n \log n - \delta^2 t} \leq \frac{9t}{8n}.$$

The inequality in the above follows since $t < n/3$ and $0 < \delta < 1/2$.
The total number of all edges from B' is

$$\frac{\delta^2 t}{3 \log n} k \log n = \frac{\delta^2 kt}{3}.$$

The number of ways to choose a set of more than $\delta^2 kt/6$ edges from $\delta^2 kt/3$ is

$$\binom{\frac{\delta^2 kt}{3}}{\frac{\delta^2 kt}{6}} \leq (2e)^{\frac{\delta^2 kt}{6}}.$$

Thus we can say that:

$$\begin{aligned} \Pr\left(X \geq \frac{\delta^2 kt}{6}\right) &\leq (2e)^{\frac{\delta^2 kt}{6}} \left(\frac{9t}{8n}\right)^{\frac{\delta^2 kt}{6}} \\ &= \left(\frac{9et}{4n}\right)^{\frac{\delta^2 kt}{6}} \\ &= \left(\frac{4n}{9et}\right)^{-\frac{\delta^2 kt}{6}}. \end{aligned}$$

The number of ways to choose a set L' of the appropriate size is no more than $(le/l')^{l'}$ and the number of ways to choose a set R' of the appropriate size is no more than $(re/r')^{r'}$. So the probability that for any set L' of size t, there exists one subset R' of size $\delta^2 t/(3 \log n)$ such that all supernodes in R' share more than $k \log n/2$ neighbors in L' is

$$\left(\frac{le}{l'}\right)^{l'} \left(\frac{re}{r'}\right)^{r'} \left(\frac{4n}{9et}\right)^{-\frac{\delta^2 kt}{6}} = \left(\frac{ne}{t}\right)^t \left(\frac{3ne \log n}{\delta^2 t}\right)^{\frac{\delta^2 t}{3 \log n}} \left(\frac{4n}{9et}\right)^{-\frac{\delta^2 kt}{6}}.$$

Below we solve for appropriate k such that this probability is less than $1/n^2$:

$$\left(\frac{ne}{t}\right)^t \left(\frac{3ne \log n}{\delta^2 t}\right)^{\frac{\delta^2 t}{3 \log n}} \left(\frac{4n}{9et}\right)^{-\frac{\delta^2 kt}{6}} \leq \frac{1}{n^2}.$$

Isolating k in the above equation, we get that:

$$k \geq \frac{6}{\delta^2 \log\left(\frac{4n}{9et}\right)} \left(\log\left(\frac{ne}{t}\right) + \frac{\delta^2}{3 \log n} \log\left(\frac{3ne \log n}{\delta^2 t}\right) + \frac{2 \log n}{t}\right).$$

Since $t < n/3$, the right side of the above inequality is no more than a fixed constant k_0, then k can be a constant greater than or equal to k_0.

2. Case 2: $t = \Theta(n)$
The probability that a single edge from L' falls in R' is $r'/r = \delta^2 t/(3n \log n)$ so by linearity of expectation, $E(X) = r'l'k \log n/r = (\delta^2 kt^2)/(3n)$. In graph G', let g be a function such that $X = g(X_1, X_2, \ldots, X_{|B'|})$.

We note that g satisfies the Lipchitz condition, i.e. $|g(X_1, X_2, \ldots, X_j, \ldots, X_{|B'|}) - g(X_1, X_2, \ldots, X_{j'}, \ldots, X_{|B'|})| \leq 1$.

Hence, we can use Azuma's Inequality to say that for any fixed positive λ:

$$\Pr(X - E(X) \geq \lambda E(X)) \leq e^{-\frac{\lambda^2 E^2(X)}{2|B'|}}.$$

Let

$$\lambda = \frac{n-2t}{2t},$$

then

$$\begin{aligned}\Pr(X \geq \frac{\delta^2 kt}{6}) &= \Pr(X - E(X) \geq \lambda E(X)) \\ &\leq e^{-\frac{(n-2t)^2}{4t^2} \frac{\delta^4 t^4 k^2}{9n^2} \frac{3}{\delta^2 tk}} \\ &= e^{-\frac{\delta^2 kt(n-2t)^2}{12n^2}}.\end{aligned}$$

The number of ways to choose a set L' of the appropriate size is no more than $(le/l')^{l'}$, and the number of ways to choose a set R' of the appropriate size is no more than $(re/r')^{r'}$.

So the probability that for any set L' of size t, there exists one subset R' of size $\delta^2 t/(3 \log n)$ such that all supernodes in R' share more than $k \log n/2$ neighbors in L' is no more than

$$\left(\frac{le}{l'}\right)^{l'} \left(\frac{re}{r'}\right)^{r'} e^{-\frac{\delta^2 kt(n-2t)^2}{12n^2}} = \left(\frac{ne}{t}\right)^t \left(\frac{3en \log n}{\delta^2 t}\right)^{\frac{\delta^2 t}{3 \log n}} e^{-\frac{\delta^2 kt(n-2t)^2}{12n^2}}.$$

Below we solve for appropriate k such that this probability is less than $1/n^2$

$$\left(\frac{ne}{t}\right)^t \left(\frac{3en \log n}{\delta^2 t}\right)^{\frac{\delta^2 t}{3 \log n}} e^{-\frac{\delta^2 kt(n-2t)^2}{12n^2}} \leq \frac{1}{n^2}.$$

Isolating k in the above equation, we get that

$$k \geq \frac{12n^2}{\delta^2 t(n-2t)^2} \left(t \log\left(\frac{ne}{t}\right) + \frac{\delta^2 t \log\left(\frac{3en \log n}{\delta^2 t}\right)}{3 \log n} + 2 \log n \right).$$

Since $t < n/3$ and $t = \Theta(n)$, the right side of the above inequality is no more than a fixed constant k_0, then k can be a constant greater than or equal to k_0.

Thus with probability at most $1/n^2$, after controlling any set L' of t peers, there exists one subset R' of size $\delta^2 t/(3 \log n)$ such that all peers in R' have more than $k \log n/2$ neighbors in L'. Thus with probability at least $1 - 1/n^2$, controlling any set of t peers by the adversary still leaves all but $\delta^2 t/(3 \log n)$ middle supernodes all with at least $k \log n/2$ good peers.

□

Lemma A.3 *Let n be the number of peers. Let t be the number of bad peers. Let δ, k be values where* $0 < \delta < 1/2$. *Let k, n be sufficiently large. Consider the butterfly network constructed in Section 3. Then after controlling any set of t peers by the adversary, all but $\delta^2 t/\log n$ supernodes are good with probability* $1 - O(1/n^2)$.

Proof. Since k is sufficiently large, we can use k in Lemma A.1, then we know that no more than $\delta^2 t/(3 \log n)$ top supernodes and no more than $\delta^2 t/(3 \log n)$ bottom supernodes have less than $2k \log n/3$ good peers in them. Next using k into Lemma A.2 gives no more than $\delta^2 t/(3 \log n)$ middle supernodes have less than $k \log n/2$ good peers in them. Thus in the butterfly network no more than $\delta^2 t/\log n$ supernodes are bad with probability $1 - O(1/n^2)$.

□

A.3 Expansive Bottom Supernodes

Lemma A.4 *Let n be the number of peers. Let t be the number of bad peers. Let δ, k be values where $0 < \delta < 1/2$. Let k, n be sufficiently large. Consider the butterfly network constructed in Section 3. Then after controlling any set of $t \leq n/3$ peers, all but $\delta n/\log n$ bottom supernodes are $(1 - (\delta t/n))$-expansive with probability $1 - O(1/n^2)$.*

Proof. Assume that for some particular k more than $\delta n/\log n$ bottom supernodes are not $(1 - (\delta t/n))$-expansive. Then each of these not $(1 - (\delta t/n))$-expansive bottom supernodes has more than $(\delta t/n)(n/\log n) = \delta t/\log n$ paths that are not good. So the total number of paths that are not good is more than

$$\frac{\delta t}{\log n}\frac{\delta n}{\log n} = \frac{\delta^2 nt}{\log^2 n}.$$

Since k be sufficiently large, then we know by Lemma A.3 that with high probability $1 - O(1/n^2)$, there are no more than $\delta^2 t/\log n$ supernodes that are bad. We also know that each of these bad supernodes which causes at most $n/\log n$ paths in the butterfly to be bad. Hence the number of paths that are not good is no more than $\delta^2 nt/\log^2 n$ which is what we wanted to show. □

A.4 Good Peers in $(1 - (\delta t/n))$-Expansive Bottom Supernodes

Lemma A.5 *Let n be the number of peers. Let t be the number of bad peers. Let ϵ, δ, k be constants where $0 < \epsilon < 1$ and $0 < \delta < 1/2$. Consider the random bipartite graph G_{bottom} constructed in Section 3: each peer participates in k random bottom supernodes and each bottom supernode contains exactly $k \log n$ peers in it. Then after controlling any set of t peers, with probability $1 - O(1/n^2)$, in any $(1-\delta)n/\log n$ bottom supernodes, there are at least $(1-\epsilon)(n-t)$ good peers appearing more than $k/2$ times.*

Proof. Controlling any set of t peers will still leave remaining $n - t$ good peers. Consider the bipartite subgraph $G_{subbottom}$ of graph G_{bottom}: $G_{subbottom}$ consists of $n-t$ good peers on the left side L and $n/\log n$ bottom supernodes on the right side R. Then we can consider that the corresponding bipartite subgraph $G'_{subbottom}$ of graph G'_{bottom}: $G'_{subbottom}$ consists of two sets A, B, where A corresponds to the set L in $G_{subbottom}$ and B corresponds to the set R in $G_{subbottom}$.

We will fix a set $R' \subset R$ of size $\delta n/\log n$ and a set $L' \subset L$ of size $\epsilon(n-t)$ and compute the probability that all peers in L' have more than $k/2$ neighbors in R'. If this occurs, then the total number of edges shared between L' and R' must be more than $\epsilon k(n-t)/2$.

In the bipartite subgraph $G'_{subbottom}$, we let A' be the set of peers in $G'_{subbottom}$ that corresponds to the set L' in $G_{subbottom}$ and B' be the set of peers in $G'_{subbottom}$ that corresponds to peers in R' in $G_{subbottom}$. Let X be a random variable giving the number of edges shared between L' and R', then X is also the random variable giving the number of edges shared between A' and B'. The probability that a single edge from A' falls in B' is δ so by linearity of expectation, $E(X) = \delta(\epsilon(n-t)k) = \delta\epsilon k(n-t)$.

In graph $G'_{subbottom}$, let g be a function such that $X = g(X_1, X_2, \ldots, X_{|B|})$. We note that g satisfies the Lipchitz condition, i.e. $|g(X_1, X_2, \ldots, X_j, \ldots, X_{|B|}) - g(X_1, X_2, \ldots, X_{j'}, \ldots, X_{|B|})| \leq 1$. Hence, we can use Azuma's Inequality to say that for any fixed positive λ:

$$\Pr(X - E(X) \geq \lambda E(X)) \leq e^{-\frac{\lambda^2 E^2(X)}{2|B|}}.$$

Let

$$\lambda = \frac{1}{2\delta} - 1 = \frac{1-2\delta}{2\delta},$$

then

$$\begin{aligned}\Pr(X \geq \frac{\epsilon k(n-t)}{2}) &= \Pr(X - E(X) \geq \lambda E(X)) \\ &\leq e^{-\frac{(1-2\delta)^2\epsilon^2 k(n-t)^2}{8n}} \\ &\leq e^{-\frac{(1-2\delta)^2\epsilon^2 kn}{16}}.\end{aligned}$$

The last line in the above follows since $t < n/3$.

The number of ways to choose a set L' of the appropriate size is no more than $(e/\epsilon)^{\epsilon(n-t)}$, and the number of ways to choose a set R' of the appropriate size is no more than $(e/\delta)^{\delta n/\log n}$. So the probability that in graph $G_{subbottom}$, for any set R' of size $\delta n/\log n$, there exists a set L' of size $\epsilon(n-t)$ such that all peers in L' have more than $k/2$ neighbors in R' is no more than:

$$\left(\frac{e}{\epsilon}\right)^{\epsilon(n-t)} \left(\frac{e}{\delta}\right)^{\frac{\delta n}{\log n}} e^{-\frac{(1-2\delta)^2\epsilon^2 kn}{16}}.$$

The number of ways to remove any set of t peers is no more than $(ne/t)^t$. So the probability that after controlling any set of t peers, for any set R' of size $\delta n/\log n$, there exists a set L' of size $\epsilon(n-t)$ such that all peers in L' have more than $k/2$ neighbors in R' is no more than:

$$\left(\frac{ne}{t}\right)^{t} \left(\frac{e}{\epsilon}\right)^{\epsilon(n-t)} \left(\frac{e}{\delta}\right)^{\frac{\delta n}{\log n}} e^{-\frac{(1-2\delta)^2\epsilon^2 kn}{16}}.$$

Below we solve for appropriate k such that this probability is less than $1/n^2$:

$$\left(\frac{ne}{t}\right)^{t} \left(\frac{e}{\epsilon}\right)^{\epsilon(n-t)} \left(\frac{e}{\delta}\right)^{\frac{\delta n}{\log n}} e^{-\frac{(1-2\delta)^2\epsilon^2 kn}{16}} \leq \frac{1}{n^2}.$$

Isolating k in the above equation, we get that:

$$k \geq \frac{16}{(1-2\delta)^2\epsilon^2}\left(\frac{\log\left(\frac{ne}{t}\right)}{\frac{n}{t}} + \frac{\epsilon\,(n-t)\log\left(\frac{e}{\epsilon}\right)}{n} + \frac{\delta\log\left(\frac{e}{\delta}\right)}{\log n} + \frac{2\log n}{n}\right).$$

Since $t < n/3$, the right side of the above inequality is no more than a fixed constant k_0, then k can be a constant greater than or equal to k_0.

Thus, with probability at most $1/n^2$, after controlling any set of t peers, for any set R' of $\delta n/\log n$ bottom supernodes there exists one set L' of $\epsilon(n-t)$ good peers such that all good peers in L' have more than $k/2$ neighbors in R'. Hence, with probability at least $1-O(1/n^2)$, after controlling any set of t peers, in any $(1-\delta)n/\log n$ bottom supernodes, there are at least $(1-\epsilon)(n-t)$ good peers appearing more than $k/2$ times. □

Recall that $s = k\log n$ is the size of a supernode in our network, and that there are n peers total, $t < n/3$ of which are bad.

Lemma A.6 *Let δ, ϵ be fixed constants where $0 < \delta < 1/2$, and $0 < \epsilon < 1$. Let the number of bad supernodes in middle levels be no more than $\delta^2 t/\log n$; let all but $\delta n/\log n$ bottom supernodes be $(1-(\delta t/n))$-expansive; and let k and n be sufficiently large. Then if the frequency of item v is $c(v)$ we can ensure that there are $(1-\epsilon)(n-t)$ good peers that can determine a frequency of v in the range $[c(v)-\delta t, c(v)+(1+\delta)t]$.*

Proof. Let s be the size of a supernode. We first prove the following claim.

Claim: For any level $i > 0$, let p be a peer in a supernode C on level i such that there are no more than r bad paths passing from level 0 to C, the total number of times the item v appears in a list of a good peer in level 0 supernodes on good paths to C is $d(v)$, and the number of bad peers in the top supernodes having paths to C is t', then when p computes $num(v)$ in C, $num(v) \in [d(v)-sr, d(v)+sr+t']$.

The proof of the claim is by induction on the level i of C. With the initial transfer from 2 supernodes on level 0 to level 1, each good peer on level 1 receives a $(v,1)$ from every good peer which has v and possibly t' extra $(v,1)$'s, from the bad peers in the supernodes. Hence every good peer on level 1 has a $num(v)$ in the range $[d'(v), d'(v)+t']$ where $d'(v)$ is the number of good peers in the supernodes which hold v, and t' is the number of bad peers in the supernodes.

Let C be a supernode on level $i > 1$ with fewer than r bad paths passing through it, and t' bad peers in the top supernodes having paths to C. Then there are links from supernodes A, B on level $i-1$ to C, with $r', r"$ bad paths passing through A, B respectively, and $r'+r" < r$. Also there are t'_1, t'_2 bad peers in the top supernodes having paths to supernodes A, B, respectively. By induction, for any good peer in A, B, $num(v) \in [d'(v)-sr', d'(v)+sr'+t'_1]$, $num(v) \in [d"(v)-sr", d"(v)+sr"+t'_2]$, respectively. If both A and B are good supernodes, then the median $num(v)$ for each is also in these ranges. Hence for each peer in C, when the sum is calculated, it is in the range $[d'(v)+d"(v)-s(r'+r"), d'(v)+d"(v)+s(r'+r")+(t'_1+t'_2)] \subseteq [d(v)-sr, d(v)+sr+t']$. Assume A is a bad supernode, then r' is equal to 2^i, the upper bound of the range does not need to count t'_1, and each peer in C by default sets the value of $num(v)$ from the bad supernode to be in the range of $[0, s2^i]$, giving a sum of $[d"(v) - sr", d"(v) + s(2^i + r") + t'_2] \subseteq [d(v) - sr, d(v) + sr + t']$. The analysis for the case where B or both A and B are bad is similar and omitted. This concludes the proof of the Claim.

Following the proof for the theorem for the standard butterfly network, there are no more than $\delta^2 t/\log n$ bad supernodes in the middle. Then no more than $\delta n/\log n$ supernodes on the bottom level have more than $\delta t/\log n$ paths containing at least one bad supernode. The remaining supernodes are $(1 - \delta t/n)$-expansive. Each bad path can affect the count of s good peers in a single level 0 supernode. Thus each peer represented in more than $k/2$ expansive supernodes receives a value in the range $[d(v) - s(\delta t/\log n), d(v) + s(\delta t/\log n) + kt]$. Dividing by k yields $[c(v) - \delta t, c(v) + (1 + \delta)t]$. There can be more than $(1 - \epsilon)(n - t)$ peers appearing more than $k/2$ times in $(1 - \delta)n/\log n$ supernodes on the bottom level which are $(1 - \delta t/n)$-expansive. Hence, $(1 - \epsilon)(n - t)$ peers receive values in this range. □

A.5 Putting it All Together

Now, we give the proof of Theorem 1.1.

Proof. By Lemma A.6, for any item v, if the frequency of item v is $c(v)$, there are $(1 - \epsilon)(n - t)$ good peers which can determine a frequency of v is in the range $[c(v) - \delta t, c(v) + (1 + \delta)t]$.

Each peer requires $2k \log n$ links for each of the kn top supernodes it plays a role in; $2k \log n$ links for each of the $C \log n$ middle supernodes it plays a role in. So each peer has $O(\log^2 n)$ neighbors in the network. Algorithm 2 will transmit the value in parallel and each of these paths has less than $\log n$ hops in it so the latency of Algorithm 2 is $O(\log n)$.

Each supernode contains exactly $k \log n$ peers and each peer in top supernodes and middle supernodes will send $2k \log n$ message. So the total number of messages transmitted in Algorithm 2 is no more than $(2k \log n)(k \log n)(n - n/\log n) = 2kn \log^2 n - 2kn \log n$. Hence, for each peer, it only needs to send and receive $O(\log^2 n)$ messages. □

B Dynamic Networks

In a real-world peer-to-peer network, nodes are constantly entering and leaving the system, which is a dynamic network. Therefore, it will be very useful to solve the problem in dynamic network. We make use of the cuckoo rule in [3] to handle the situation that peers join and leave the network. At first, we will construct three networks with help of the cuckoo rule and the three networks correspond to the top level of butterfly network, middle levels of butterfly network and bottom level of butterfly network respectively. Secondly, we are able to apply the algorithm in the statistic network to solve the problem in the dynamic network.

B.1 Cuckoo Rule for Dynamic Network

Cuckoo rule is introduced in [3]. In the dynamic network, assume at any time, n is the maximum number of good nodes and ϵn is the maximum number of bad nodes for some $\epsilon < 1$. A $region$ is an interval of size $1/2^r$ in $[0, 1)$ for some integer r that starts at an integer multiple of $1/2^r$. A $k - region$ is a region of size (closet from above to) k/n, and for any point $x \in [0, 1)$, the k-region $R_k(x)$ is the unique $k - region$ containing x.

Cuckoo rule is described as follows: if a new node v wants to join the network, pick a random $x \in [0, 1)$. Place v into x and move all nodes in $R_k(x)$ to points in $[0, 1)$ chosen uniformly and independently at random(without replacing any further nodes). In our method, an interval $I \subset [0, 1)$ of size $\log n/n$ can be considered as a supernode.

With the above definition, we will have the following theorem in [3].

Theorem B.1 *Let n be the maximum number of good nodes in the network and ϵn is the maximum number of bad nodes. Let $\epsilon < 1$ and k be constant with $\epsilon < 1 - 1/k$. For each interval I of size $\log n/n$(I is considered as a supernode), with high probability $1 - 1/n$, the cuckoo rule with parameter k satisfies the following two conditions for a polynomial number of rounds, with any adversarial strategy:*

1. *Balancing condition: I contains $\Theta(|I| * n)$ nodes.*
2. *Majority condition: the good nodes in I are in the majority.*

Therefore, we can make use of cuckoo rule to construct the following networks: the network N_{top} corresponds to the top level of the butterfly network, the network N_{middle} corresponds to the middle level of the butterfly network, and the network N_{bottom} corresponds to the bottom level of the butterfly network.

Consider the network N_{top}, the number of nodes is n and each node makes C replicas. Thus the total number of nodes is Cn if we consider each replica as a node. Notice that each node will have C replicas, so the algorithm needs to use cuckoo rule to handle each replica joining the network. ϵ is the fraction of the bad nodes. We can choose an interval of size $\log n/n$ such that for each interval I of size $\log n/n$, with high probability $1 - 1/n$, the cuckoo rule with parameter k satisfies Balancing condition and Majority condition for a polynomial number of rounds, with any adversarial strategy. Each interval is considered as a supernode, thus the number of supernodes in N_{top} is just $n/\log n$. Moreover, each supernode will contain $\Theta(\log n)$ nodes. Also, the network N_{bottom} can be constructed by the same process.

Consider the network N_{middle}, the number of nodes is n and each node makes $C \log n$ replicas. Thus the total number of nodes is $Cn \log n$ if we consider each replica as a node. Notice that each node will have $C \log n$ replicas, so the algorithm needs to use cuckoo rule to handle each replica joining the network. ϵ is also the fraction of the bad nodes. Let $N = n \log n$. This time, we can choose an interval of size $\log N/N$, and for each interval I of size $\log N/N$, with high probability $1 - 1/N$, the cuckoo rule with parameter k satisfies Balancing condition and Majority condition for a polynomial number of rounds, with any adversarial strategy. Each interval is considered as a supernode, thus the number of supernodes in N_{middle} is just $N/\log N = n \log n/\log(n \log n) = n$. Moreover, each supernode will contain $\Theta(\log n)$ nodes.

With the networks N_{top}, N_{bottom}, and N_{middle}, we can construct the butterfly network B consisting of these three networks. Therefore, we can make use of Algorithm 2 to count the frequencies of items in the butterfly network. When some peer joins the network, we make use of cuckoo rule to handle it; when some peer leaves the network, we just let it leave freely. With Theorem B.1, all the supernodes satisfy the balancing condition and majority condition with any adversarial strategy.

B.2 Proofs for Dynamic Network

In the dynamic network, we can relax the requirement that each supernode have exactly $k \log n$ peers in it as follows.

Lemma B.2 *Let l, r, r', d, β' and n be any positive values where $l' \leq l$, $\beta' > 1$ and*

$$d \geq \frac{4r}{r'l(\beta'-1)^2}\left(r' \ln\left(\frac{re}{r'}\right) + 2\ln n\right).$$

Let G be a random bipartite multigraph with left side L and right side R where $|L| = l$ and $|R| = r$ and each node in L has edges to d random neighbors in R. Then with probability at least $1 - 1/n^2$, there is no set $R' \subset R$, where $|R'| = r'$ such that all nodes in R' have degree greater than $\beta' ld/r$.

Proof. We will again use the probabilistic method to show this. We will first fix a set $R' \subset R$ of size r' and compute the probability that all nodes in R' have degree greater than $\beta' ld/r$. If this bad event occurs then the total number of edges shared between L and R' must be at least $\beta' r' ld/r$. Let X be a random variable giving the number of edges shared between L and R'. The probability that a single edge from L falls in R' is r'/r so by linearity of expectation, $E(X) = r'ld/r$.

We can then say that:

$$\Pr\left(X \geq \frac{\beta' r' ld}{r}\right) = \Pr(X \geq (1+\delta)E(X)) \leq e^{-E(X)\delta^2/4}.$$

Where $\delta = \beta' - 1$ and the last equation follows by Chernoff bounds if $1 < \beta' < 2e - 1$.

The number of ways to choose a set R' of the appropriate size is no more than $(re/r')^{r'}$. So the probability that no subset R' of size r' has this bad event occur is

$$\left(\frac{re}{r'}\right)^{r'} \cdot e^{-\frac{r'l'd\delta^2}{4r}}.$$

Below we solve for appropriate d such that this probability is less than $1/n^2$.

$$\left(\frac{re}{r'}\right)^{r'} \cdot e^{-\frac{r'ld\delta^2}{4r}} \leq 1/n^2 \quad (1)$$

$$r' \ln\left(\frac{re}{r'}\right) - \frac{r'ld\delta^2}{4r} \leq -2\ln n \quad (2)$$

$$\frac{4r}{r'l(\beta'-1)^2}\left(r' \ln\left(\frac{re}{r'}\right) + 2\ln n\right) \leq d$$

We get step (2) from step (1) in the above by taking the logarithm of both sides.

□

Lemma B.3 *Let β, δ', n, k be values such that $\beta > 1$, $\delta' > 0$ and assume n is sufficiently large. Let each node participate in $k \ln n$ of the middle supernodes, chosen uniformly at random. Then all but δ' middle supernodes have less than $\beta k \ln n$ participating nodes with probability at least $1 - 1/n^2$.*

Proof. For simplicity, we will assume there are n middle supernodes (we can throw out any excess supernodes and the lemma will still hold). Let $l = n$, $r = n$, $r' = \delta'$, $d = k \ln n$ and $\beta' = \beta$ in Lemma B.2. Then the statement in this lemma holds provided that:

$$\begin{aligned} k \ln n &\geq \frac{4}{\delta'(\beta-1)^2}\left(\delta' \cdot \ln\left(\frac{ne}{\delta'}\right) + 2\ln n\right); \\ k &\geq \frac{4}{(\beta-1)^2 \ln n} \cdot \left(\delta' \cdot \ln\left(\frac{ne}{\delta'}\right) + 2\ln n\right). \end{aligned}$$

The right hand side of this equation goes to some constant as n goes to infinity.

□

Lemma B.4 *Let β, δ', n, k be values such that $\beta > 1$, $\delta' > 0$ and n is sufficiently large. Let each node participate in k of the top (bottom) supernodes (chosen uniformly at random). Then all but δ' top (bottom) supernodes consist of less than $\beta k \ln n$ nodes with probability at least $1 - 1/n^2$.*

Proof. Let $l = n$, $r = n/\ln n$, $r' = \delta'$, $d = k$ and $\beta' = \beta$ in Lemma B.2. Then the statement in this lemma holds provided that:

$$k \geq \frac{4}{\delta' \ln n(\beta-1)^2}\left(\delta' \cdot \ln\left(\frac{ne}{\delta' \ln n}\right) + 2\ln n\right).$$

The right hand side of this equation goes to some constant as n goes to infinity.

□

We can see that the number of supernodes whose nodes is greater than $\beta k \ln n$ is very small. Thus we can make use of $\beta k \ln n$ as threshold in Algorithm 2. In the butterfly network B, the Lemma A.1, Lemma A.2, Lemma A.4, Lemma A.6, Lemma A.5 and Theorem 1.1 will hold.

Thus we have the following Lemma.

Lemma B.5 *Let n be the number of peers. Let t be the number of bad peers, where $t < n/3$. Let δ, ϵ, k, β be constants where $0 < \delta < 1/2$, $0 < \epsilon < 1$, and $\beta > 1$. Let $s = \beta k \log n$ be the maximum size of a supernode. Let $\delta^2 t/\log n$ be the total number of supernodes which are bad in middle levels. If the frequency of item v is $c(v)$, then there are $(1-\epsilon)(n-t)$ good peers which can determine a frequency of v is in the range $[c(v) - \delta\beta t, c(v) + (1+\delta)\beta t]$.*

Proof. Let s be the maximum size of a supernode. We first prove the following claim.

Claim: For any level $i > 0$, let p be a peer in a supernode C on level i such that there are no more than r bad paths passing from level 0 to C, the total number of times the item v appears in a list of a good peer in level 0 supernodes on good paths to C is $d(v)$, and the number of bad peers in the top supernodes having paths to C is t', then when p computes $num(v)$ in C, $num(v) \in [d(v) - sr, d(v) + sr + t']$.

The proof of the claim is by induction on the level i of C. With the initial transfer from 2 supernodes on level 0 to level 1, each good peer on level 1 receives a $(v, 1)$ from every good peer which has v and possibly t' extra $(v, 1)$'s, from the bad peers in the supernodes. Hence every good peer on level 1 has a $num(v)$ in the range $[d'(v), d'(v) + t']$ where $d'(v)$ is the number of good peers in the supernodes which hold v, and t' is the number of bad peers in the supernodes.

Let C be a supernode on level $i > 1$ with fewer than r bad paths passing through it, and t' bad peers in the top supernodes having paths to C. Then there are links from supernodes A, B on level $i-1$ to C, with r', r" bad paths passing through A, B respectively, and $r' + r" < r$. Also there are t'_1, t'_2 bad peers in the top supernodes having paths to supernodes A, B, respectively. By induction, for any good peer in A, B, $num(v) \in [d'(v) - sr', d'(v) + sr' + t'_1]$, $num(v) \in [d"(v) - sr", d"(v) + sr" + t'_2]$, respectively. If both A and B are good supernodes, then the median $num(v)$ for each is also in these ranges. Hence for each peer in C, when the sum is calculated, it is in the range $[d'(v) + d"(v) - s(r' + r"), d'(v) + d"(v) + s(r' + r") + (t'_1 + t'_2)] \subseteq [d(v) - sr, d(v) + sr + t']$. Assume A is a bad supernode, then r' is equal to 2^i, the upper bound of the range does not need to count t'_1, and each peer in C by default sets the value of $num(v)$ from the bad supernode to be in the range of $[0, s2^i]$, giving a sum of $[d"(v) - sr", d"(v) + s(2^i + r") + t'_2] \subseteq [d(v) - sr, d(v) + sr + t']$. The analysis for the case where B or both A and B are bad is similar and omitted. This concludes the proof of the Claim.

Following the proof for the theorem for the standard butterfly network, there are no more than $\delta^2 t/\log n$ bad supernodes in the middle. Then no more than $\delta n/\log n$ supernodes on the bottom level have more than $\delta t/\log n$ paths containing at least one bad supernode. The remaining supernodes are $(1 - \delta t/n)$-expansive. Each bad path can affect the count of s good peers in a single level 0 supernode. Thus each peer represented in more than $k/2$ expansive supernodes receives a value in the range $[d(v) - s(\delta t/\log n), d(v) + s(\delta t/\log n) + \beta kt]$. Dividing by k yields $[c(v) - \delta\beta t, c(v) + (1 + \delta)\beta t]$. There can be more than $(1 - \epsilon)(n - t)$ peers appearing more than $k/2$ times in $(1 - \delta)n/\log n$ supernodes on the bottom level which are $(1 - \delta t/n)$-expansive. Hence, $(1 - \epsilon)(n - t)$ peers receive values in this range. □

Thus the Algorithm 2 in the dynamic network can achieve the same result in the static network except that the approximation guarantees on the estimated counts of each item will be weakened by a constant factor β.

C Lowerbound

Theorem C.1 *In a peer-to-peer network, let n be the number of peers. Let t be the number of bad peers. Consider any algorithm that solves the* FindFrequentCounts *problem. If in this algorithm, each peer has only $O(\log^q n)$ links to other peers, where $q > 0$ is some constant, then for any item v with frequency $c(v)$ in the network, the adversary can ensure that at most $n - O(t/\log^q n)$ peers get an estimate of the frequency of v in the range $[c(v) - O(t/\log^q n), c(v) + O(t/\log^q n) + t]$.*

Proof. For any scalable algorithm, if all peers have only $O(\log^q n)$ neighbors, the adversary can easily target a set, T, of $O(t/\log^q n)$ peers and then delete all the peers that are neighbors of any peer in the set T. The peers in T will then be completely isolated and unable to receive any messages. Similarly, the items held by these peers will not be counted by any other peers. Consider the case where the adversary targets all peers that hold some item v. Then the lower bound for the approximated frequency counts of item v is $c(v) - O(t/\log^q n)$. On the other hand, consider the case where the adversary targets a set T where none of the peers in T hold the item v. Then, all the t bad peers can pretend they have the item v and the adversary can pretend that all the peers in T have item v. In this way, the upper bound for the approximated frequency count of item v is $c(v) + O(t/\log^q n) + t$. Thus at most $n - O(t/\log^q n)$ peers can get the frequency of item v in the range $[c(v) - O(t/\log^q n), c(v) + O(t/\log^q n) + t]$ after an attack by the adversary. □

By Theorem C.1, we can see that Algorithm 2 is optimal up to a polylogarithmic factor among all algorithms that are *scalable*, in the sense that the each peer has a polylogarithmic number of neighbors.

D Reducing the Number of Messages

In this section, we sketch how to improve message passing between two successive supernodes so that only $\Theta(\log n)$ messages are sent in expectation. This allows us to ensure that each peer in the network only needs to send $\Theta(\log n)$ in expectation. Our techniques in this section are based on past work by Saia and Young in [30].

Assume that h_1 is a function that maps a peer uniformly to an integer in the range 1 to $\log n$ and h_2 is a function that maps a peer to D integers selected uniformly and independently in the range 1 to $\log n$ for some constant D. Further assume that h_1 and h_2 are known to all the peers in the network. Instead of having a full bipartite graph between two supernodes connected in the butterfly network, we will have a constant degree bipartite expander between the peers of these two supernodes(see Figure 4).

The method of constructing the constant degree bipartite expander between two successive supernodes is as follows. For any two neighboring supernodes Q_i at level i and Q_{i+1} at level $i + 1$, each peer q in Q_i will link to peer p in

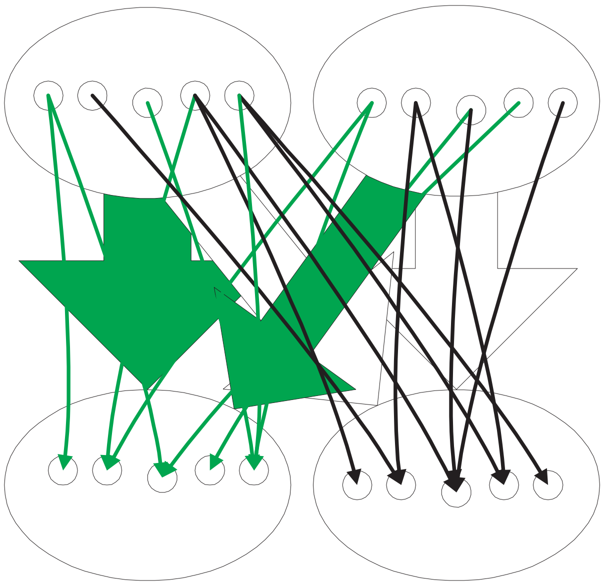


Figure 4: The constant degree bipartite expander graphs between supernodes.

Q_{i+1} if and only if $h_1(q) = h_2(p)$. We can think of this process as assigning each peer q in Q_i to a bin numbered between between 1 and $\log n$ whose number is given by $h_1(p)$, and each peer q in Q_{i+1} is assigned to D bins numbered between 1 and $\log n$ whose numbers are given by $h_2(p)$.

In the whole butterfly network, we keep the same full bipartite graph between the peers on the top level 0 supernodes and level 1 supernodes. However, we make use of the above constant degree bipartite expander between all the other neighbored supernodes. The new protocol, which we refer to as Algorithm 2.1, is exactly the same as Algorithm 2, except for the following single change. Let A and B be supernodes at level i, C be a supernode at level $i+1$ that is a neighbor of A and B in the butterfly, and p be a peer in C. During the *Transfer* protocol, instead of p receiving messages from all peers in A and all peers in B, it receives messages only from the peers in A and the peers in B that are neighbors in the bipartite expander graphs constructed above. The peer p ignores messages received from all other peers. The rest of Algorithm 2.1 is equivalent to Algorithm 2. The following theorem follows from the results in [30].

Theorem D.1 *Let n be the number of peers. Let t be the number of bad peers and $t < n/4$. Then for any fixed $0 < \epsilon < 1$, $0 < \delta < 1/2$, Algorithm 2.1 solves the* FindFrequentCounts *problem, even in the case where an adaptive adversary causes a up to t of the peers in the network to suffer Byzantine faults. Moreover, the resource costs required by Algorithm 2.1 are as follows.*

- *Latency is $O(\log n)$*
- *Every peer sends and receives $O(\log n)$ messages in expectation*
- *Every peer has $O(\log n)$ neighbors in expectation*

E Reducing the Number of Bits

In Algorithm 2.1, the size of the messages increase as we progress down the butterfly network. We now analyze the number of bits sent in Algorithm 2.1. If the total number of items is some constant, then the maximum number of bits of any message is $O(\log n)$. Thus in this case, each peer sends $O(\log n) * O(\log n) = O(\log^2 n)$ bits. However, consider the following worse case. Assume that initially, any peer in the network has at most some fixed constant c items on it and every item has a constant number of bits. Then in each top supernode, there are at most $ck \log n$ distinct items contained. Consider the messages generated at level i for all $i = 1, \ldots, \log n - \log\log n$ levels of the butterfly network. A message generated at level i can contain at most $(2^i ck) \log n$ items and their frequencies. Thus, the number of bits in one message can be $O(2^i \log n)$. Since for any level i, the number of messages transmitted from level i to level $i+1$ is $O(n)$, the number of bits transmitted from level i to level $i+1$ is $O(2^i n \log n)$. The total number of bits transmitted from the top level to bottom level is thus $O(n^2 \log n)$. Thus each peer sends $O(n \log n)$ bits in this worse case. In this section, we try to reduce the number of bits sent to polylogarithmic even in the worse case. We solve the problem in two different cases that are described in Section E.1 and Section E.2, respectively.

E.1 General Distribution and Constant Items Stored in Each Peer

In this section, we handle the case that the items in the network follows any general distribution and each peer contains only constant distinct items. We can reduce the number of bits transmitted if we allow the user to introduce an error tolerance threshold during transmission. In particular, instead of estimating counts in the range $[c(v) - \delta t, c(v) + (1 + \delta)t]$, we will instead provide estimates that are in the range $[c(v) - \delta(t + M), c(v) + (1 + \delta)t]$.

Therefore, we need to improve Algorithm 2.1: for any peer at level i, after receiving two messages from its neighbors in the two supernodes above, it will generate a new message containing only a part of items with frequency more than an error tolerance threshold other than generating a new message containing all the items and their frequency in the two messages. In this way, we can reduce the number of bits during Algorithm 2.1. The crucial point is to find a reasonable error tolerance threshold.

The construction of the butterfly network is the same as the one in Section D. For any supernode Q at level i, we choose the error tolerance threshold for this supernode is $(2^i \delta k M \log n)/n$, so that only items with frequency more than $(2^i \delta k M \log n)/n$ can be transmitted to Q's neighbors at level $i + 1$. The protocol for the supernode network is similar to the Algorithm 2.1 for butterfly network with supernode except that we modify the function sum.

The new subroutine $sum(list_q, list_r)$ is defined as follows:

- $sum(list_q, list_r)$: for a peer p in a supernode Q at level i, where $1 \le i \le \log n - \log\log n$, it returns $\{(v, num_q(v) + num_r(v)) \mid \forall v \ s.t.\ num_q(v) + num_r(v) > \frac{2^i \delta k M \log n}{n}\}$.

The new algorithm, which we call Algorithm 2.2 is equivalent to Algorithm 2.1 except it makes use of this new subroutine sum. The properties of Algorithm 2.2 are described in Theorem E.1.

Theorem E.1 *Let n be the number of peers. Let t be the number of bad peers and $t < n/3$. Let M be the threshold such that one item can be considered as frequent item only if its frequency is greater than M, where $M > t$ and M is $\Theta(n)$. Then for any fixed $0 < \epsilon < 1$, $0 < \delta < 1/2$, we are able to solve the* FindFrequentCounts *problem, even in the case where an adaptive adversary causes a $1/3$ fraction of the peers in the network to suffer Byzantine faults. Moreover, the resource costs required by our algorithm are as follows.*

- *Algorithm 2.2 has latency $O(\log n)$.*
- *Every peer sends and receives $O(\log n)$ messages.*
- *Every peer has $O(\log n)$ neighbors in the network.*
- *Every peer sends $O(\log^2 n)$ bits.*

Proof. The framework of proof for the Theorem E.1 is almost the same as that of Theorem D.1. However, in Algorithm 2.2, for the message transmitted from any supernode Q_i at level i to Q_{i+1}'s neighbor at level $i + 1$, it only contains the items with frequency more than $(2^i \delta k M \log n)/n$. Thus for any item v, its frequency may loose during the transmission. The maximum frequency lost in the transmission is $\frac{2^i \delta k M \log n}{n} \frac{n}{2^i \log n} = \delta k M$. Hence the frequency of item v received by the good peers at the bottom level is in the range $[c(v) - \delta(t + M), c(v) + (1 + \delta)t]$.

Since there are at most c distinct items in each peer, then there are at most $ck \log n$ total items in one top supernode. Then for any supernode Q at level i, the frequency of items it counts is at most $2^i ck \log n$. Note that the error tolerance threshold for Q is $(2^i \delta k M \log n)/n$. Thus there are at most $(2^i ck \log n)/(2^i \delta k M \log n/n) = cn/(\delta M)$ distinct items that can be transmitted to Q's neighbor. Since the bits of one item and its frequency is at most $O(\log n)$, the number of bits transmitted from Q to its neighbor is at most $(cn/(\delta M))O(\log n) = O(\frac{n \log n}{M})$. So the total number of bits transmitted in Algorithm 2.2 is no more than $n(2D)(C \log n)O(\frac{n \log n}{M}) = O(\frac{n^2 \log^2 n}{M})$, where D is the constant defined in Section D. For each peer, it only needs to send $O(\frac{n \log^2 n}{M})$ bits in expectation. Since M is $\Theta(n)$, each peer only needs to send $O(\log^2 n)$ bits. $\square$

E.2 Zipfian Distribution and Any Number of Items Stored in Each Peer

In this section, we consider an even worse case: the total number of distinct items is m and each peer can hold any number of distinct items. However, in this case, the items in the network follow Zipfian distribution. Also, we define another approximating frequent item problem as follows.

FindApproxTop $(S, m, k, t, q, \epsilon, \delta)$
Given: A set S of n peers; total m distinct items in the network; integers k and t, where $t/n < 1/3$; and positive reals ϵ and δ. Each peer in S initially has as input up to any number items; an adaptive and omniscient adversary controls t of these peers. For an item v, let $c(v)$ be the number of good peers that have item v as an input.
Output: Each peer p has as output a set of items F_p with estimated frequencies $c_p(v)$ for all items v in F_p. For all but ϵn of the good peers, the following is true: F_p contains all items v such that $c_p(v) \geq (1-\delta)c(v_q) - t$, where v_q is the top qth frequent item.

For a Zipfian distribution with parameter z, $c(v_q) = \frac{c}{q^z}$ for some scaling factor constant c, where v_q is the top qth frequent item. We give an algorithm that uses hash function to transmit the frequency of the items so that if $z > 1/2$, only $O(\log^3 n + \log^2 n \log m)$ bits are sent by each peer in expectation. The construction of the butterfly network is also the same as the one in Section D. Our techniques in this section are based on past work in [30, 7].

Let r, b be parameters with values to be determined later. Let $h_1, \ldots, h_r$ be hash functions that map objects to an integer in the range 1 to b, and $s_1, \ldots, s_r$ be hash functions that map objects to an integer -1 or 1. Then we will have the following data structure: a $CountSketch$ data structure C consists of these hash functions along with a $r \times b$ array of counters, which will be interpreted as an array of r hash tables, each containing b bins. The data structure supports two operations for ont item q:

- $Add(C, q)$: for $i \in [1, r]$, $h_i[q]$ += $s_i[q]$.
- $Estimate(C, q)$: return $|median\{h_i[q]* s_i[q]\}|$.

In the new algorithm, the transmission between the peers in two successive supernodes is the same as Algorithm 2.1. However, instead of transmitting the lists, each peer in the network will transmit the $CountSketch$ to its neighbors. Thus we need to make some changes as follows.

We define C_p for a peer p to be a $CountSketch$ data structure. Initially, each p is given a list of items, each with $num = 1$. The following subroutine is used:

- $sum(C_p, C_q)$: returns $\{C_p + C_q\}$, where $+$ is the add operator on matrix.

The subroutine $Initialize\_total$ is changed as follows:

- $Initialize\_total$: when the messages from level 0 supernodes are received, each peer in a level 1 supernode constructs the $CountSketch$ C by the operator $Add(C, q)$ for each item q in the messages.

The subroutine $median(A, i)$ is changed as follows:

- $median(A, i)$: given a collection of $CountSketch$ sent by peers in supernode A, $\{C_p \mid p \in A\}$, this function returns the $CountSketch$ f in which each element of f is the median value of the corresponding element over all $p \in A$. The function sets $2^i s$ to any input values in which $num(v) > 2^i s$, where s is the number of peers in a supernode.

  We also need to define a new subroutine $major\_topqlist(A)$ as follows:

- $major\_topqlist(A)$: given a collection of top q frequent item lists sent by peers in supernode A, $A_{top} = \{l_p \mid p \in A\}$, and the number of elements in A_{top} is T this function returns a top q frequent item list l_q in which each jth frequent item of l_q occurs in the jth frequent position of the top q frequent item lists more than $T/2$ times.

Lemma E.2 *Let n be the number of peers and let m be the number of distinct items in the whole network, and let N be the total frequency of all items in the whole network, where N is at most mn. Let α be a constant where $\alpha > 0$. Let r, b be the parameters for the $CountSketch$, where r is $O(\log N)$, and $b \geq 8k$. Let $\beta = \sqrt{\frac{\sum_{q'=k+1}^{m} c^2(v_{q'})}{b}}$, where $v_{q'}$ is the top q'th frequent item. If there is no bad peer in the network, then after running Algorithm 3, we can get that with high probability $1 - \alpha/N$, for any item v, if the frequency of item v is c(v), then $||median\{h_i[v]s_i[v]\}| - c(v)| \leq 8\beta$.*

The proof of Lemma E.2 can be seen in [7].

Lemma E.3 *Let n be the number of peers and let each peer have constant distinct items. Let m be the number of distinct items in the netwrok. Let N be the total frequency of all items in the whole network, where N is at most mn. Let t be the number of bad peers, where $t < n/3$. Let δ, ϵ, k be constants where $0 < \delta < 1/2$, and $0 < \epsilon < 1$. Let $s = k \log n$ be the size of a supernode. Let α be a constant where $0 < \alpha < 1/100$. Let r, b be the parameters for the*

Algorithm 3 FindApproxTop (with Supernodes)

1. Phase 1
   (a) Each peer is assigned to a supernode in such a way that Claim 2.7 is satisfied (details of how this is done are described in Section 3).
   (b) $Initialize\_total$: Each peer on a level 0 supernode sends its list of items to each peer in the supernode's neighbors on level 1. When the messages from level 0 supernodes are received, constructs the $CountSketch\ C$ by the operator $Add(C, q)$ for each item in the messages.
   (c) For all level i, $1 \leq i < \log n - \log\log n$, of the butterfly network, repeat
      - *Transfer:* Each peer p sends its $CountSketch$ to each of p's neighbored peers in the supernode's neighbors on level $i + 1$. Let Q be a supernode on level $i + 1$ and let A and B be its neighbors on level i. Each peer p in Q computes C_A and C_B, respectively, by applying $median$ to each set of $CountSketch$ sent by p's neighbored peers in A and B, respectively.
      - *Check:* If any element of $CountSketch$ received from level i contains a $num_v > 2^i s$, set this element of the $CountSketch$ to $2^i s$.
      - *Combine:* Each peer p in supernode Q on level $i + 1$ revises its list as follows: $C_p = sum(C_A, C_B)$, where A and B are Q's neighbors on level i.
   (d) *Final Step:*As a single peer may have several $CountSketch$ with different value depending on which supernode it is in, it selects the median value as its final $CountSketch$.

2. Phase 2
   (a) $Initialize$: Each peer on a level 0 supernode sends its list of items to each peer in the supernode's neighbors on level 1. When the messages from level 0 supernodes are received, then makes use of $Estimate$ function on the $CountSketch$ generated in phase 1 to determine the list l_q of the top q frequent items in all the items in the messages.
   (b) For all level i, $1 \leq i < \log n - \log\log n$, of the butterfly network, repeat
      - *Transfer:* Each peer p sends its list l_q of the top q frequent items to each of p's peers in the supernode's neighbors on level $i + 1$. Let Q be a supernode on level $i + 1$ and let A and B be its neighbors on level i. Each peer in Q computes l_{Aq} and l_{Bq}, respectively, by applying $major\_topqlist$ to each set of lists sent by peers in A and B, respectively.
      - *Filter:* Each peer makes use of $Estimate$ function on its $CountSketch$ generated in phase 1 to determine the list l_q of the top q frequent items in the two lists l_{Aq} and l_{Bq}.
   (c) *Final Step:*As a single peer may have several different top q frequent items list depending on which supernode it is in, it applies $major\_topqlist$ to the set of lists it received at bottom level supernodes.

CountSketch, where r is $O(\log N)$, and $b \geq 8k$. Let $\beta = \sqrt{\frac{\sum_{q'=k+1}^{m} c^2(v_{q'})}{b}}$, where $v_{q'}$ is the top q'th frequent item. If the frequency of item v is $c(v)$, then there are $(1-\epsilon)(n-t)$ good peers which can determine that v's frequency $c_p(v)$, and $-8\beta - \delta t \leq c_p(v) - c(v) \leq 8\beta + (1+\delta)t$

Proof. By Lemma E.2, if there is no bad peer in the network, then $||median\{h_i[v] * s_i[v]\}| - c(v)| \leq 8\beta$, i.e. $-8\beta \leq |median\{h_i[v] * s_i[v]\}| - c(v) \leq 8\beta$. Now consider the t bad peers. By Lemma A.6, if there are t bad peers controlled by the adversary in the network, then $c_p(v)$ will be in the range $[|median\{h_i[v] * s_i[v]\}| - \delta t, |median\{h_i[v] * s_i[v]\}| + (1+\delta)t]$, i.e. $-\delta t \leq c_p(v) - |median\{h_i[v] * s_i[v]\}| \leq (1+\delta)t$. Thus we can get that $-8\beta - \delta t \leq c_p(v) - c(v) \leq 8\beta + (1+\delta)t$. □

Lemma E.4 *Let n be the number of peers and let m be the number of distinct items in the whole network. Let N be the total frequency of all items in the whole network, where N is at most mn. Let t be the number of bad peers, where $t < n/3$. Let δ, ϵ, k be constants where $0 < \delta < 1/2$, and $0 < \epsilon < 1$. Let $s = k \log n$ be the size of a supernode. Let $\beta = \sqrt{\frac{\sum_{q'=k+1}^{m} c^2(v_{q'})}{b}}$, where $v_{q'}$ is the top q'th frequent item. Let r, b be the parameters for the CountSketch, where r is $O(\log n)$, and $b \geq max\left(8k, \frac{256\sum_{q'=k+1}^{m} c^2(v_{q'})}{(\delta c(v_q) - 2\delta t)^2}\right)$. Then there are $(1-\epsilon)(n-t)$ good peers which can get the estimated top q items occurring at least $(1-\delta)c(v_q) - t$ times in the network.*

Proof. By Lemma E.3,for any item v, the estimate for the frequencies are within an additive factor of the true frequencies, which is in the range $[c(v) - 8\beta - \delta t, c(v) + 8\beta + (1+\delta)t]$. Thus for two items whose true frequencies differ by more than $16\beta + (1+2\delta)t$, the estimates correctly identify the more frequent item. By setting $16\beta + (1+2\delta)t \leq \delta c(v_q) + t$, we ensure that the only items that can replace the true most frequent items in the estimated top q list are items with true frequencies at least $(1-\delta)c(v_q) - t$.

$$16\beta + (1+2\delta)t = 16\sqrt{\frac{\sum_{q'=k+1}^{m} c^2(v_{q'})}{b}} + (1+2\delta)t \leq \delta c(v_q) + t.$$

Isolating b in the above equation, we get that:

$$b \geq \frac{256\sum_{q'=k+1}^{m} c^2(v_{q'})}{(\delta c(v_q) - 2\delta t)^2}.$$

Thus, we have proved the lemma. □

Theorem E.5 *Let n be the number of peers. Let t be the number of bad peers and $t < n/3$. Let m be the number of distinct items in the whole network, and let N be the total frequency of all items in the network, where N is at most mn. Let r, b be the parameters for the CountSketch, where r is $O(\log N)$, and $b \geq max\left(8k, \frac{256\sum_{q'=k+1}^{m} c^2(v_{q'})}{(\delta c(v_q) - 2\delta t)^2}\right)$. Assume that the items in the network distribute in Zipfian distribution with parameter $z > 1/2$, $c(v_q) = \frac{c}{q^z}$ for some scaling factor constant c, where v_q is the top qth frequent item. Then for any fixed $0 < \epsilon < 1$, $0 < \delta < 1/2$, we are able to solve the* FindApproxTop *problem, even in the case where an adaptive adversary causes a $1/3$ fraction of the peers in the network to suffer Byzantine faults. Moreover, the resource costs required by our algorithm are as follows.*

- *Algorithm 3 has latency $O(\log n)$.*
- *Every peer sends and receives $O(\log n)$ messages.*
- *Every peer has $O(\log n)$ neighbors in the network.*
- *Every peer sends $O(\log^3 n + \log^2 n \log m)$ bits.*

Proof. We only need to prove the number of bits sent by every peer is $O(\log^3 n + \log^2 n \log m)$. For the Zipfian distribution with parameter $z > 1/2$, we can get that

$$\sum_{q'=k+1}^{m} c^2(v_{q'}) = \sum_{q'=k+1}^{m} \frac{1}{(q')^{2z}} = \begin{cases} O(\log m), z = \frac{1}{2} \\ O(k^{1-2z}), z > \frac{1}{2} \end{cases}$$

Thus if $z > 1/2$, b = k. In Phase 1 of Algorithm 3, the size of the $CountSketch$ is $r \times b \leq O(\log N * k) \leq O(\log(mn))$ and the number of bits of each element in $CountSketch$ is $O(\log n)$. Thus the number of bits for each $CountSketch$ is $O(\log n * \log(mn))$. Each peer is assigned to $O(\log n)$ supernodes, thus each peer sends $O(\log n * \log(mn) * \log^n) = O(\log^2 n \log(mn)) = O(\log^3 n + \log^2 n \log m)$ bits in the first phase. In Phase 2, each peer only sends q items to its neighbors at the successive level, thus each peer sends $O(\log n)$ bits in the second phase.

Therefore, in Algorithm 3, each peer sends $O(\log^3 n + \log^2 n \log m)$ bits. $\square$