DISCOVERY OF FREQUENT ITEM SET IN PEER-TO-PEER NETWORKS

by

Emrah Çem

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical & Computer Engineering

Koç University

October, 2010

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Emrah Çem

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assoc. Prof. Öznur Özkasap (Advisor)

Prof. A. Murat Tekalp

Assoc. Prof. Mine Çağlar

Date:

To my family and cousin Serap Cansu

ABSTRACT

Several peer-to-peer (P2P) applications require a global view of system information such as data access frequencies, item frequencies, query and event counts, that are available locally and partially at peers. Frequent item set discovery (FID) in a distributed environment is a common problem requiring global information computation. Items that globally occur more than a threshold value are referred as *frequent* or *popular* and the number of diverse applications that need globally frequent items is increasing expeditiously in today's P2P networks. Therefore, efficiently discovering frequent items would be a valuable service for peers. Being significant for P2P systems, FID problem is also applicable to distributed database applications, cache management, data replication, sensor networks, and security mechanisms in which identifying frequently occurring items in the entire system is useful.

In this thesis, we propose and develop a gossip-based distributed approach, namely ProFID, for discovering frequent items in unstructured P2P networks. In contrast to the prior studies, our solution progresses in a fully distributed manner using an atomic averaging function to discover frequent items. Utilizing averaging function with gossip-based aggregation in frequent item set discovery problem and a practical convergence rule are novel and beneficial features of our approach. We make the following contributions to the current state of the art. First, we propose a fully distributed Protocol for Frequent Item Discovery (ProFID) where the result is produced at every peer. ProFID uses a novel pairwise averaging function and network size estimation together to discover frequent items in an unstructured P2P network. We also propose a practical rule for convergence of the algorithm. In contrast to previous works, each peer gives local decision for convergence based on the change of updated local state. Moreover, we developed a model of ProFID in PeerSim and performed various experiments to compare and evaluate its efficiency, scalability, applicability. Finally, we compared the accuracy and scalability of ProFID with adaptive Push-Sum algorithm. The comparison results show that ProFID outperforms adaptive Push-sum in terms of accuracy, convergence speed and message overhead.

ÖZETÇE

Bir çok görevdeş ağ uygulaması, sistemdeki düğümlerde kısmi olarak bulunan veri erişim sıklığı, öğe sıklığı, sorgu ve olay sayısı gibi sistem genelindeki bilgiye ihtiyaç duyarlar. Dağıtık bir sistemde sık bulunan öğelerin belirlenmesi problemi sistem genelindeki bir bilginin hesaplanmasını gerektiren yaygın bir problemdir. Sistem genelindeki sıklığı belirli bir eşik değerinin üstünde bulunan öğelere *sık bulunan* veya *yaygın öğeler* denir. Günümüz P2P ağlarında, sistem genelinde sık bulunan öğelerin bilgisine ihtiyaç duyan uygulamaların sayısı hızla artmaktadır. Bu yüzden sık bulunan öğelerin verimli bir şekilde belirlenmesi, bir çok görevdeş ağ uygulamalarında, düğümler için değerli bir servistir. Ayrıca, bu servis dağıtık veri tabanı uygulamalarında, önbellek yönetiminde, veri yineleme yöntemlerinde, algılayıcı ağlarda ve güvenlik mekanizmalarının önemli olduğu sistemlerde de kullanılabilir.

Bu tezde, ProFID olarak adlandırdığımız, yapılandırılmamış görevdeş ağlarda sık bulunan öğelerin belirlenmesi için epidemik tabanlı yöntemi kullanan tam dağıtık bir yöntem sunulmuştur. Ortalama yönteminin epidemik tabanlı yöntem ile birlikte sık bulunan öğelerin tespit edilmesinde kullanılması ve pratik yakınsama yöntemi önerdiğimiz yöntemin yenilikçi ve yararlı özellikleridir. Bu konudaki çalışmalara katkılarımızı şu şekilde sıralayabiliriz. İlk olarak, sık bulunan öğe setinin bütün düğümlerde hesaplandığı tam dağıtık bir yöntem önerisi sunulmuştur. Bu yöntem yapılandırılmamış ağlarda ikili gruplar halinde ortalama yöntemi ile ağ boyutu tahminini kullanarak sık bulunan öğeleri tespit eden ilk yöntemdir. İkinci olarak, algoritmanın sonlandırılması için pratik yakınsama yöntemi sunulmuştur. Bu yöntem ile düğümler yerel durumlarındaki değişimleri değerlendirerek, diğer düğümlerden bağımsız olarak yakınsama kararı alırlar. Ayrıca, PeerSim simülatörü ile elde edilen kapsamlı sonuçlar kullanılarak önerilen yöntemin verimliliği, ölçeklenebilirliği ve uygulanabilirliği ölçülmüştür. Son olarak, uyarlanmış Push-Sum ile ProFID yöntemleri karşılaştırılmıştır. Bu karşılaştırma sonuçlarında, ProFID hem doğruluk hem de ölçeklenebilirlik açısından uyarlanmış Push-Sum yöntemine göre daha iyi sonuçlar vermiştir.

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Öznur Özkasap for her great support, encouragement, and understanding as well as her stimulating ideas. Besides my advisor, I am also grateful to members of my thesis committee for their valuable and insightful comments.

Special thanks to my colleagues Yusuf Sahillioglu (for his motivating acts and mimics), M. Ali Yatbaz (for his motivating jokes), C. Goktug Gurler (for his motivating ideas), Burak Gorkemli (for his motivating life experience), Görkem Saygılı (for his motivating friendship), and many others in College of Engineering.

I am sincerely and heartily grateful to my future wife, Sevinç Tekin, for her moral support she showed me throughout my dissertation writing. I am sure it would have not been possible without her support.

My deepest gratitude goes to my family for their unflagging love and support throughout my life. My father, Yunus Çem, has always supported me in pursuit of my dreams and motivated me from an early age. My mother, Nazife Çem, is the most altruistic mother in the world and she has done her best, and I am sure she will; this dissertation would be impossible without them.

The financial support of the Turkish National Science Foundation (TÜBİTAK) is also sincerely acknowledged.

Finally, I offer my regards and blessings to all of those who supported me in any respect during the completion of my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Recent years have witnessed an extraordinary growth of P2P network services that have a very dynamic structure since peers may join or leave the system at any time. On the other hand, advantages of these type of systems are the enhanced scalability and service robustness due to their distributed architectures. Because of P2P systems' dynamic and scalable nature, centralized approaches are not as functional and reliable as decentralized approaches. In decentralized approaches, there is no central administration, so peers need to communicate with each other to perform various tasks such as searching and indexing. Furthermore, peers may need a system-wide information such as network size, system load, query/event counts, or mostly contacted peers for specific files in order to perform various tasks such as load-balancing and topology optimization [1]. Database applications, wireless sensor networks, and security applications can also make use of frequent item set discovery (FID) service, as well as P2P applications. Hence, efficient discovery of frequent items would be a valuable service for distributed systems.

There are various P2P applications such as cache management, search technique design, query refinement, content mirroring, network topology optimization, denial of service attack and internet worm detections that can utilize FID service (see Table 1.1). For example, in cache management application, peers can put frequent item set into a cache to access them faster. This will reduce the average access time of peers significantly, since the probability of accessing frequent items is more probable than accessing non-frequent items in the near future. Other than P2P networks, FID protocol can also be used in sensor networks to detect anomalies and attacks. For example, assume that there are many movement sensors implanted in the ground which can detect whether there is a moving object around it or not. If most of the sensors detect a moving object, then there might be the possibility of an attack. It can also be used to detect anomalies in atmospheric conditions via temperature sensors

Application Area	Application	Meaning of an item
Database	Cache management	A keyword used in queries
Sensor Networks	Military attack detection	A sensed soldier trying to pass through the border
Internet Security	Worm detection	A similar byte sequence
Internet Security	DoS attack detection	A destination address of a packet
P2P Networks	Most downloaded files detection	A shared file in the network
P2P Networks	Topology optimization	A contacted peer ip address

Table 1.1: Usage of FID in some applications

[2, 3]. If average temperature of a region is measured above/below a certain threshold, then that might be an indication of an anomaly in atmospheric condition such as fire. Moreover, that service can be used in database applications such as maintenance [4], auto-administration [5], and computation of iceberg (hot list) queries [6, 7, 8].

1.1 Scope

The goal of this research is to develop a distributed approach for efficient discovery of frequent items in unstructured networks. The main motive in this thesis is to provide an efficient and robust distributed approach which does not use any central administration as in hierarchical approaches and global information such as network size or topology information.

FID problem can be decomposed into 3 phases. First phase is the initialization of the local state, which has no network overhead. Second phase is the computation of aggregates of items in the system, which is the phase that this thesis mostly focuses on. Efficient aggregate computation is the key point in FID problem since robustness, network overhead, and speed of the algorithm mostly depends on the efficiency of this phase. In fact, what differentiates a FID problem from an aggregate computation problem is that it is not necessary to compute the frequencies of all items until the end of the algorithm. Last phase is to distinguish frequent items from infrequent items. Last phase also does not have any network cost. It is only a local computation of information. There are various studies focusing on each phase separately and they will be discussed in Chapter 2.

1.2 Contributions

We address the FID problem in unstructured P2P networks. Our contributions are as follows:

1. We propose a fully distributed gossip-based approach named ProFID using pairwise averaging function which is novel in frequent item set discovery problem.

2. We introduce a practical convergence algorithm. In contrast to previous works, each peer gives local decision for convergence based on the change of updated local state. Converged peers may list frequent items independently from other peers in the network.

3. We develop a model of ProFID in PeerSim [9], and perform various experiments to compare and measure its efficiency and scalability.

4. We develop a model of adaptive Push-Sum protocol in PeerSim and compare it with ProFID in terms of accuracy and scalability.

1.3 Overview

In this thesis, a distributed approach using atomic pairwise averaging with gossip-based aggregation technique to compute frequent item set is proposed. Details of the thesis are presented in respective chapters as described here:

Chapter 2 presents the literature review. First, FID problem is divided into 2 subproblems, namely aggregate computation problem and threshold mechanisms. Then, in Sect. 2.1, aggregate computation studies are discussed in detail. In Sect. 2.2, existing threshold techniques are discussed. Finally, in Sect. 2.3, studies focusing on FID are reviewed.

Chapter 3 presents the details of ProFID. Firstly, the system model and the formal definition of the FID problem is stated in Sect. 3.1 and 3.2, respectively. The network details such as peer characteristics, network structure, and communication style is given. Then, problem is defined formally with system parameters composed of local and global parameters. Moreover, an illustrative example is given to clarify the problem and its parameters. Then, the reason of using average aggregation instead of sum aggregation is explained. Then, algorithm is given with parameters and their descriptions in Sect. 3.3. Then, system size estimation technique used in ProFID is given in Sect. 3.4. Next, the details of atomic pairwise averaging is given as well as the convergence rule of ProFID in Sect. 3.5 and 3.6, respectively. Next, discussion of analytical findings is given in Sect. 3.7. Lastly, threshold determination mechanism we proposed is presented in Sect. 3.8.

Sect. 4.1 presents the details of how the network environment is set up. Then, in Sect. 4.1.1, review of popular P2P networks simulators is introduced. Then, Sect. 4.1.2 gives the details of the PeerSim simulation environment we used to model ProFID and obtain experimental results. Finally, in Sect. 4.1.3, we give implementation details of ProFID in PeerSim.

Chapter 4 presents the simulation results of ProFID. First, Sect. 4.1 presents the details of how the network environment is set up. Then, different performance metrics that were used for analysis is presented, as well as algorithm parameters. Then, we evaluated the effect of those metrics in both atomic pairwise averaging operation and ProFID in two different sections. We also compared the time and message complexity of ProFID with a well known Push-Sum protocol that we modified to adapt it to the FID problem.

Chapter 5 gives the comparison results of ProFID and adaptive Push-Sum protocols. First, in Sect. 5.1, the details of how Push-Sum algorithm is adapted to FID problem are given. Then, comparison results of ProFID and adaptive Push-Sum are presented in Sect. 5.2.

Finally, in Chapter 6, this thesis is concluded and future directions are stated.

Chapter 2

RELATED WORK

This section describes related work on FID and two sub-problems of it. Different approaches will be compared and contrasted in those fields. As illustrated in Fig. 2.1, FID problem can be divided into two parts, namely aggregate computation and threshold mechanism. There exist hierarchical and gossip based approaches for aggregate computation, and static(absolute) and adaptive(relative) schemes for threshold mechanisms. We first review the studies on aggregate computation and threshold mechanisms, and then the studies focusing on FID problem.

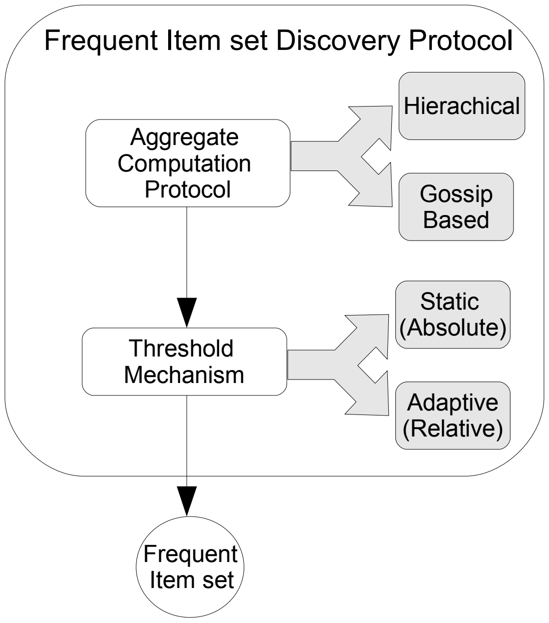


Figure 2.1: Subproblems of FID problem.

2.1 Aggregate Computation

Aggregate computation is a common name for operations computing global information, such as sum, average, max, and min, in distributed systems. It is an important step in popular item identification, since an item is identified as popular if its global value (sum aggregate) is above a threshold value. Aggregate computation protocols need to have the following properties to be applicable in real networks [10]:

- *Scalable* : Today's P2P networks support a large number of nodes at any specific time; hence an aggregation computation protocol should have an efficient memory usage and a reasonable completion time in such a large network.

- *Robust* : P2P networks are very dynamic and they show churn characteristics since peers enter/leave the system whenever they want. Moreover, message loss events can be frequent due to network characteristics. Thus, a solution should be resilient to node failures and message losses.

- *Communication efficient* : A method should compute the result with small number of messages in order to use bandwidth efficiently, and run with other applications in parallel.

In general, there are two main approaches in aggregate computation techniques, namely *hierarchical* and *gossip-based*. In hierarchical approaches, peers form a structure such as a tree to communicate with each other. These approaches are generally communication efficient but not robust against peer failures. In dynamic networks, they have the problems of structure maintenance cost as well as the single point of failure. On the other hand, gossip-based approaches are robust against peer and link failures. Moreover, system load is distributed fairly and there is no single point of failure. Furthermore, result is obtained at every peer instead of only at the initiator peer. However, communication overhead is higher due to random communication and the result is probabilistic. The longer the protocol runs, the better approximation is obtained.

2.1.1 Gossip-based aggregate computation

Gossip-based protocols emerged for the maintenance of replicated database systems [4] and then they have been used for various applications such as reliable data dissemination [11, 12], membership maintenance [13, 14], overlay topology construction [15, 16, 17], failure detection [18, 19], P2P streaming [20, 21] as well as data aggregation [2, 22, 23, 10, 24, 25, 26]. Even though gossip-based protocols are used in different application areas, they have some common properties. Gossip-based algorithms consist of rounds, which are the time intervals that nodes periodically communicate among each other. In each round, each peer contacts one or a few nodes, called neighbors, to exchange state. The more rounds, the closer the calculated aggregate to true aggregate. Algorithm finishes in multiple rounds, and data is disseminated to the network like an epidemic disease. A peer infects its neighbor, and then neighbor infects its neighbors, and so on. An advantages of this approach is its robustness against peer failures. Removal, or a failure of a peer does not affect the dissemination speed significantly. It is also very simple and there is no single point of failure. Moreover, all peers have equal responsibilities; hence the system is inherently load-balanced. However, in practice, peer capabilities such as computation power and bandwidth might be significantly different, which may make load-balance undesirable property [27] . Consider a video streaming application, which requires fast data dissemination [1]. In such an environment, system utilization must be maximized, instead of giving equal responsibilities to all peers. Thus, we can infer that if time is critical, gossip-based approaches may fall behind. Another disadvantage of gossip-based approaches is high communication overhead since any two neighbors may communicate multiple times during the algorithm, hence resulting in redundant information exchange when compared to hierarchical approaches. Furthermore, result is probabilistic in gossip-based approaches due to random communication of peers. However, it is more convenient to use gossip-based algorithms in P2P, wireless and sensor networks with high link-failure probability, since deterministic algorithms generally use a hierarchy which may fail even with a little disruption such as node or link failures [23].

In gossip-based protocols, there are three communication styles used to exchange states as depicted in Fig. 2.2:

[1]fast data dissemination means *within a given time window* in video streaming context

- *Push based*: peer chooses random peer(s) to send its state
- *Pull based*: peer chooses random peer(s) to receive their states
- *Push-pull based*: peer chooses random peers both to send and receive states.

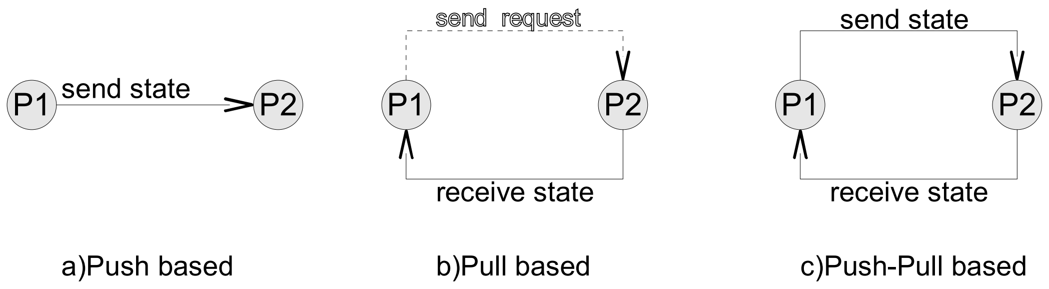


Figure 2.2: Communication styles in gossip-based approaches

Aggregate computation operation is generally based on obtaining an information sample and then combining the values in order to calculate global values [28]. Several researchers have proposed decentralized gossip-based protocols to compute aggregates and most relevant ones are described next.

Push-Sum Protocol

A well-known gossip-based data aggregation protocol is named Push-Sum, as described in study [23]. In this study, Kempe et al. propose a push-synopses protocol for aggregate computation and analyze the scalability, reliability and efficiency of their approach. According to this protocol, each peer i initially keeps a single value $s_{0,i} = x_i$ and a weight $w_{0,i} = 1$. It is assumed that, each peer sends $(s_{0,i}, w_{0,i})$ tuple to itself at t_0. Then in each round t, every peer performs the actions given in Alg. 1.

Push-Sum protocol computes average aggregate values in $O(logN)$ rounds with $O(NlogN)$ messages, where N is the network size. Relative error is bounded by ε after $O(logN + log\frac{1}{N} + log\frac{1}{\delta})$ rounds with probability 1-δ. However, algorithm uses a single item and assumes that all peers are aware of that item initially, meaning that even a peer, which does not have the

Algorithm 1: *Push-Sum Protocol*

```
Input: prevIncomingPairs: all incoming pairs at time t-1
Output: estimate: estimate of aggregate at time t
1 $(\hat{s}_r, \hat{w}_r)$=prevIncomingPairs;
2 $s_{t,i}= \sum_r \hat{s}_r$, $w_{t,i}= \sum_r \hat{w}_r$
3 target=chooseUniformlyAtRandom()
4 send($\frac{1}{2}s_{t,i}$, $\frac{1}{2}w_{t,i}$) /*to yourself and target*/
5 estimate=$\frac{s_{t,i}}{w_{t,i}}$
6 return estimate
```

item, sets its local value to zero. This is not possible in FID problem because FID problem has multiple items and each peer only knows its local state initially. Peers are not aware of items that exist in the system at initial state. Hence Push-Sum algorithm converges to true result, if and only if all peers initially have a knowledge about all items in the system, which might be an inconvenient requirement for large networks without centralized agents. Moreover, efficiency of the algorithm uses the assumption of uniform gossiping. They also present a scheme to compute rank and select samples using O($\log^2 N$) rounds and O($N \log^2 N$) messages.

Push-Pull Based Aggregation

Jelasity et al. [22] present a fully distributed way of calculating aggregates such as counting, averages, sums, products, and extremal values. Instead of push based communication, they use a push-pull based communication. Moreover, they provide a theoretical analysis of the proposed algorithm. They restart the gossip protocol periodically in order to prevent accumulation in estimation error. They also give both theoretical analysis and empirical results of convergence of the proposed algorithm. Moreover, they analyze the effect of overlay topology on convergence of the algorithm, and show that it effects the convergence significantly. Hence, they use topology information while determining the termination time. This is not practical since it may not be available at all peers.

Other Gossip-based Aggregation Protocols

A gossip-based protocol for computing aggregates such as min, max, sum, average, and rank has been proposed in [10]. However, only theoretical analysis is performed without any simulation results. At the end of the proposed algorithm, results are computed at all peers in $O(logNloglogN)$ rounds with message complexity $O(NloglogN)$. Main contribution in that study is the decrease in message cost by a small increase in round complexity. According to their system model, each peer holds a single value and gossip rounds are synchronized. Moreover, a node can send a message to only one node (in push-based model) or receive from only one node (in pull-based model). Furthermore, each peer can communicate with every other node in the network. Upper bound for a gossip message size is given as $O(logN+logQ)$, where Q is the range of values at the nodes. Atomicity is maintained by queuing connections in case of multiple connection request. If queue fills up, connection requests are rejected. Node failures are not considered, however, links and initial node crashes are assumed. In order to compute aggregate values, clusters are formed using a coloring mechanism. Cluster roots compute aggregate value of their own cluster. Then, cluster roots gossip among themselves to compute global aggregate. Finally, cluster roots propagate the result back to their cluster members.

In [25], a non-uniform gossip-based aggregate computation technique is proposed. According to this technique, node i has a probability of P_{ij} to communicate with its neighbor node j. They relate the convergence time of the gossip-based averaging with the second largest eigenvalue of a doubly stochastic matrix. Since minimizing this quantity is a semi-definite program (SDP), and SDPs can not be solved in a distributed environment. They propose a subgradient method that solves the minimization problem.

Another study [26] suggests a gossip-based technique, namely Distributed Random Grouping (DRG), for aggregate computation in wireless sensor networks with better performance. It is local and randomized and it uses probabilistic grouping to efficiently converge to the actual value. They give an upper-bound for the convergence time of the proposed algorithm, as well as comparison results with other distributed algorithms. Yet, they use the broadcasting property of wireless sensor network which reduces its applicability.

Another recent study [29] proposed asynchronous distributed protocols for average computation and proved the convergence of the proposed protocol both by analytical studies and simulation results deployed on PlanetLab. Proposed algorithms do not require global coordination and converge under asynchronous timing assumptions. They use a push-pull based pairwise communication. In pairwise updates, peers do not simply half the values, but use a *stepsize* parameter γ to update local state as shown in Fig. 2.3. They also use atomic pairwise state exchange assumption in order to calculate the average correctly. Their algorithm sends NACK messages when a peer is waiting a reply from another peer, but receives a push (they call *state*) message from a peer. This may cause redundant message exchanges and these increase message overhead.

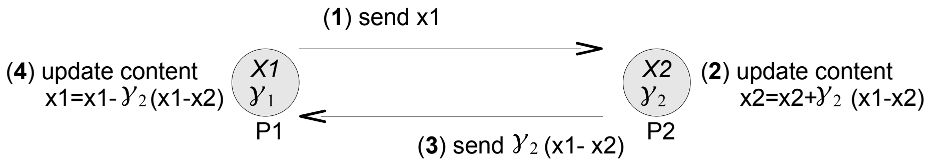


Figure 2.3: Pairwise update using *stepwise* parameter

2.1.2 Hierarchical aggregate computation

In hierarchical aggregate computation, a virtual hierarchy is constructed by either giving some tasks to some specific peers or forming an infrastructure and passing local aggregates to upper levels till the whole aggregate values are merged at the root peer of the infrastructure. Since the state is aggregated over a hierarchy, if a failure occurs in an upper level peer, then all the aggregate information coming to that peer will not be able to reach the root node. Hence, a lot of aggregation information will be lost even in the case of a single peer failure.

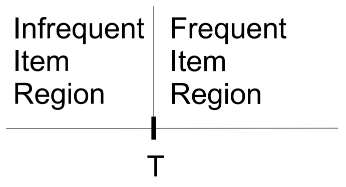
An approach in this category [1] forms an infrastructure from the most stable peers to calculate the aggregate information. Another study [30] constructs a virtual infrastructure which has some monitor nodes monitoring frequency counts of a specific stream, and a unique root node which monitors the frequent items. Even though the aggregate information is not calculated directly, popular items are identified by using an infrastructure to control communication of peers. Study of [3] computes the aggregate values by giving some specific tasks to some chosen peers in the network. A hybrid solution has been introduced in [31] to compute aggregates in sensor networks, which uses strengths of both gossip and tree aggregation protocols together.

Using a hierarchy is risky as failure in the root peers, or upper level peers in infrastructures or responsible peers in task-based hierarchy may cause undesirable results such as loosing all pre-computed aggregates, and constructing the hierarchy again and again. Hence hierarchical aggregation needs a recovery mechanism as opposed to the gossip-based aggregation.

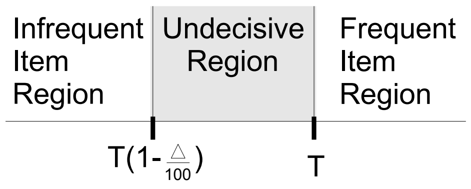
2.2 Threshold Mechanism

Threshold is the parameter that differentiates the aggregate computation from FID problem. The term *frequent* is not very clear by itself; however, by adding a user defined parameter called *threshold* , frequent items can be discriminated from infrequent items. If an item occurs more than a threshold, then it is called frequent item, otherwise we call it as an infrequent item. In order to discover frequent items in a set, a reasonable threshold must be chosen. However, the term *reasonable threshold* has also unclear meaning and it is user/application dependent. In most applications, determination of a reasonable threshold value is non-trivial. Consider the problem of discovery of frequently downloaded files in a P2P network. The term *frequently downloaded* is not obvious and it can be interpreted differently by each person. Note that, discovery of mostly downloaded top-k items is much easier, since number of files that needs to be output is known a priori. Consequently, we are left with a term *reasonable threshold* in the problem of discovery of frequent items whose value needs to be determined by considering the application.

There are several studies that use threshold mechanism in distributed systems for different applications. In one study [3], threshold is used in monitoring applications. If an item's frequency exceeds the threshold, then it is monitored on the screen to inform the user. The problem is to decide when and how frequently to inform monitor nodes in order to output frequent items continuously with a minimum message overhead in the network. Other studies [1, 30] use the threshold to discriminate frequent items from infrequent items, as well as to prune infrequent items during algorithm to reduce communication cost. Another study [32] uses threshold just to discriminate frequent items from infrequent items, which is similar to our usage of threshold.



(a)



(b)

Figure 2.4: (a) Regular threshold (b) Lahiri's threshold

If it is not possible to calculate the exact frequencies of items, which is generally the case, one can determine a region named *indecisive region* which includes frequencies around the threshold and helps to reduce erroneous decisions. Study [32] uses such an approach to prevent taking wrong decisions on boundaries as shown in Fig 2.4b.

In general, determination of the threshold can be categorized into two depending on whether it has a predetermined or a data dependent value. If the threshold is determined independent of data, it is called *absolute threshold*. If the threshold is determined based on the distribution of data, then it is called *relative threshold*.

2.2.1 Absolute Threshold

In absolute threshold mechanism, threshold is determined before algorithm is started and it is commonly used in monitoring applications. The following examples are appropriate for using absolute threshold mechanism.

- Raise an alert when the average temperature is above 60 in a region (might be an indicator of a fire)

- Raise an alert when the total number of cars on a specific highway exceeds a certain value. This value can be obtained after some observation and it might be different for each highway. (might be indicator of a traffic congestion in that highway) [3]

2.2.2 Relative Threshold

In relative threshold mechanism, peers do not know the item distribution; hence, predetermining the threshold is not possible. Peers determine the threshold after the calculation of frequencies of items. For instance, peer may set the threshold to the average or median of frequencies of all items it knows. Relative frequency mechanism is more appropriate in applications in which estimation of item distributions is troublesome.

2.3 Frequent Item Set Discovery (FID)

Frequent item set discovery (FID) problem has been studied for data streams, P2P services and network monitoring. First deterministic algorithm for data streams is proposed in [33]. Algorithms named *Sticky sampling* and *Lossy counting* for identifying items whose frequency exceeds a user-defined threshold value are proposed in [34]. They find approximate frequencies, and the error rate is bounded by a user-specified parameter. They also propose an algorithm to optimize finding frequent item set in a single pass. However, if the dataset is highly skewed, algorithm may result in high error rates. Even though their algorithms are efficient and applicable for the discovery of frequent items in datastreams, it is a centralized algorithm. More related to our work, [30] proposes an algorithm to find approximate frequent items in distributed data streams. Datastreams are composed of item occurrences with time stamps and recent items are given more importance while discovering frequent items. The aim is to output approximate frequencies of items whose true frequencies exceed a user-specified threshold. In order to accomplish that, a hierarchical communication structure is constructed. Moreover, the concept of precision gradient is introduced in order to minimize communication overhead . Another recent work [35] describes significant algorithms in FID in data streams and provides baseline implementations of them. Experimental evaluation on different data sets is also performed in order to figure out how practical their implementations are. An algorithm for the problem of distributed monitoring of thresholded counts is proposed in [3, 36]. The goal is to monitor all items whose frequency is above a threshold value within specified accuracy bounds. They use hierarchical approach. However, since the frequent items are only available on a single node, this work is exposed to single point of failure.

Obtaining exact frequent items with a technique named in-network filtering is investigated in [1]. It consists of two phases called candidate filtering and candidate verification. In order to calculate the aggregates, a tree-based hierarchy composed of only stable peers is formed, and the most stable peer is chosen as the root of the hierarchy. This is done to make algorithm more stable, since constructing tree-like structures is not robust against failures. The basic idea is to form item groups and calculate the aggregates of item groups. If item group frequency is below the threshold value, than all items in that group fails to be a candidate, otherwise items are chosen as candidates (for being frequent item). They theoretically show how to optimize parameters of the proposed approach such as group size, filter size, and give the effects of parameters on the performance of the algorithm. However, this approach is not practical and easy to deploy for large scale distributed settings.

A uniform gossip based technique for disseminating the local information of each peer is proposed in [32]. Since they use gossiping for data dissemination, result is probabilistic as opposed to the study of [1]. However, thanks to gossiping, there is no underlying network structure or central control which makes algorithm simpler. In order to identify frequent elements, a threshold mechanism is used, and by using sampling, the communication load is decreased. Moreover, a space efficient data representation named *sketch* is used to store aggregate values efficiently.

Chapter 3

PROFID: PROTOCOL FOR FREQUENT ITEM SET DISCOVERY

We propose a gossip-based fully distributed approach with pairwise averaging function for discovering frequent items in unstructured P2P networks. Utilizing pairwise averaging function with gossip-based aggregation and a practical convergence rule are novel features of our approach. Due to the unstructured form of communication in gossiping, it is not easy to prevent a local frequency of an item at a peer from being accounted for multiple times. For this reason, either peers need to handle the duplicates in incoming gossip messages or use an aggregation function which is insensitive to duplicates. Pairwise averaging function is the latter choice. Even though the local frequency of an item at a peer might be accounted for multiple times, it still efficiently converges to the approximate global average frequency of the item at each peer due to *mass conservation* [23]. The only assumption we do is to perform pairwise averaging operation atomically. We used buffering and timeout mechanisms to provide atomicity.

3.1 System Model

We consider a network consisting of N peers with unique id which have only local state information initially. We also consider that peers form an unstructured network and they know only a subset of other peers, called *neighbors*, in the network. In order to have knowledge about system wide information, peers have to collaborate and obtain information about all peers' states in the network. In order to collaborate and communicate, peers need to know the identifiers of their neighbors. This neighborhood relation determines the topology of an *overlay network*. We also assume that peers may leave (intentionally or due to failures) or join the network at any time, which is inevitable in P2P networks.

We use a pairwise communication model meaning that a peer can communicate with only a single peer at any time. In distributed computing, communication is determined by a time model and it can be categorized as synchronous and asynchronous. In synchronous time model, local clocks of peers are synchronized and each peer performs operations within the same time interval. This is actually not practical in large distributed systems due to the variable and unpredictable message delays. In asynchronous time model, each peer uses its local clock to perform a computation. In our case, peers communicate in rounds with a fixed duration. However, it is not necessary to synchronize the peers' local clocks because peers use clocks just to perform periodic operations. Round duration just determines how often a peer sends local state to its neighbor(s). It should be long enough (more than max round trip time (RTT) between any two peers) to complete a push-pull communication, but must not be too long in order to compute the result quickly. We also make the assumption that no malicious information is given by peers that will cause algorithm to work improperly such as modifying local state.

3.2 Problem Statement

Consider a network consisting of N peers denoted as $P=\{P_1, P_2, \ldots, P_N\}$ and M item types denoted as $D=\{D_1, D_2, \ldots, D_j, \ldots, D_M\}$, where D$_j$ has a global frequency g_{D_j}. Parameters *N, M,* and g are system-wide information, hence they are unknown to all peers a priori.

Let each peer (P_i) has a local set of items $S_i \subseteq D$ and each local item (D_j) has a local frequency f_{i,D_j} such that

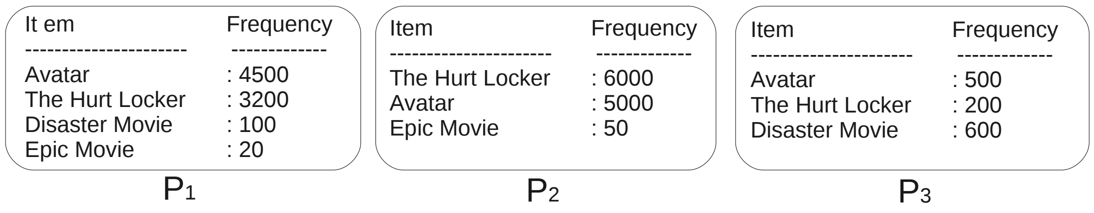
$$g_{D_j} = \sum_{i=1}^{N} f_{i,D_j}, \qquad D_j \notin S_i \Longrightarrow f_{i,D_j} = 0 \tag{3.1}$$

The aim is to find (at all peers) all items with frequency above threshold T(see Eq. 3.2) as fast as possible with low communication overhead and high accuracy.

$$F(T) = \{D_j | g_{D_j} > T, \forall j \in 1, 2, \ldots, M\} \tag{3.2}$$

3.2.1 Illustrative Example

To illuminate the parameters, let's examine how we can use FID protocol in a cache management problem. Consider three peers in Fig.3.1 representing local movie databases distributed in different countries. Items at each peer correspond to the name of movies queried/searched by clients and frequency of each item corresponds to the number of queries that includes the item (movie name in this case). Our system parameters for these peers can be written as given in Table 3.1. After running a FID protocol with threshold 8000, all the movies that are queried more than the threshold are computed. In this example, protocol needs to output *Avatar* and *The Hurt Locker*. Peers may actually utilize this information to cache frequently queried/searched movies in order to access them in a shorter time because clients will probably query these movies more than others in future, at least for a period of time.



P1

It em	Frequency
Avatar	: 4500
The Hurt Locker	: 3200
Disaster Movie	: 100
Epic Movie	: 20

P2

Item	Frequency
The Hurt Locker	: 6000
Avatar	: 5000
Epic Movie	: 50

P3

Item	Frequency
Avatar	: 500
The Hurt Locker	: 200
Disaster Movie	: 600

Figure 3.1: Sample three peers with local frequencies of movie items queried/searched

3.3 *Algorithm*

Being a frequent item is directly related with the global frequency of that item and a straightforward way of calculating the global frequency of an item is to use a sum aggregate function. However, computing a global frequency using a sum aggregate function is problematic since local values of peers may contribute to the sum more than once during the computation due to the random nature of communication in gossip-based algorithms. For this reason, we propose an approach that uses atomic pairwise average aggregate function along with a network size estimation to compute global frequencies of items.

Atomic pairwise averaging function computes the global average of an item D_j, ga_{D_j}, which is defined as follows:

$$ga_{D_j} = \frac{1}{N} \sum_{i=1}^{N} f_{i,D_j} \tag{3.3}$$

Then, network size estimation is used to obtain the global values of items:

$$g_{D_j} = N \cdot ga_{D_j} = N\left(\frac{1}{N} \sum_{i=1}^{N} f_{i,D_j}\right) \tag{3.4}$$

Global parameters (unknown a priori):
$P = \{P_1, P_2, P_3\}$, N=3
$D = \{Avatar, DisasterMovie, TheHurtLocker, EpicMovie\}$, M=4
$g(Avatar) = 10000, g(DisasterMovie) = 700, g(TheHurtLocker) = 9400, g(EpicMovie) = 70$
P$_1$'s local parameters:
$S_1 = \{Avatar, TheHurtLocker, DisasterMovie, EpicMovie\}$
$f_{1,Avatar} = 4500, f_{1,TheHurtLocker} = 3200, f_{1,DisasterMovie} = 100, f_{1,EpicMovie} = 20$
P$_2$'s local parameters:
$S_2 = \{TheHurtLocker, Avatar, EpicMovie\}$
$f_{2,TheHurtLocker} = 6000, f_{2,Avatar} = 5000, f_{2,EpicMovie} = 50$
P$_3$'s local parameters:
$S_3 = \{Avatar, TheHurtLocker, DisasterMovie\}$
$f_{3,Avatar} = 500, f_{3,TheHurtLocker} = 200, f_{3,DisasterMovie} = 600$

Table 3.1: Parameters of distributed movie database example

In general, gossip algorithms can be divided into 3 parts in terms of decisions regarding:

1. To whom gossip messages to send
2. What to perform when a message comes in
3. When to stop (convergence rule)

In ProFID, all of those decisions are taken locally, and peers do not know any system wide information such as topology and network size, initially. Algorithm 2 shows initialization, periodic send operation and handling of incoming messages. First, peers setup their local states, and a single peer, namely *initiator*, adds an item to its local state with unique id (*ui*) and frequency 1. Then, gossip rounds start and each peer sends its state to one of its neighbors periodically. The algorithm continues until every peer decides that it has converged. Descriptions of algorithm parameters are given in Table 3.2 to better understand the algorithm. Moreover, we provide a state diagram of a peer in Fig. 3.2 to clarify how a peer reacts to certain events during a gossip round.

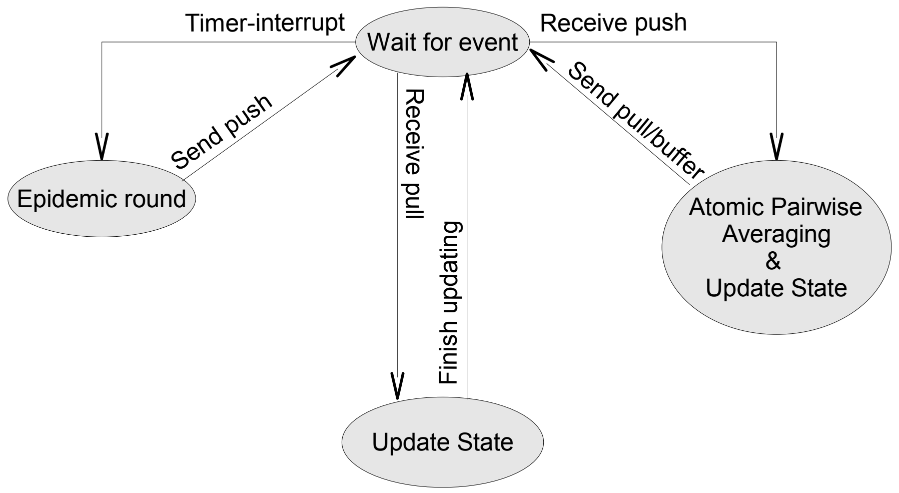


Figure 3.2: Peer State Diagram

In the rest of this chapter, the details of ProFID will be explained. First, system size estimation technique used in ProFID is explained in Sect. 3.4. Then, atomic pairwise averaging operation is described in Sect 3.5. Then, convergence rule is explained in Sect. 3.6. Finally, threshold mechanism used to distinguish frequent items from infrequent items is given in Sect. 3.8.

3.4 System Size Estimation

System size N is a global parameter, so it is not known by any peer a priori and it also needs to be calculated. There are different approaches for network size estimation. Three candidate ones are Sample&Collide [37], Hops Sampling [38], and Gossip-based aggregation [2]. A comparative study of those approaches are also done in study [39], using different performance metrics such as scalability, accuracy, message overhead,reactivity to changes. The results show that even though the message overhead is high in gossip-based aggregation, it is the only algorithm in which result is computed at each node as opposed to others in which result is computed only at the initiator peer. Hence, there is no need for a broadcast of

Algorithm 2: *ProFID: Protocol for Frequent Item Set Discovery*

```
Input: $fan-out$, $mms$, $ui$, $convLimit$, $\varepsilon$, $T$, $S$
Output: $F$: set of frequent items
Initialize
if $Initiator$ then
    $S$.add($ui$,1);
$converged$=$false$; $prevSizeEstim$=0; $convCounter$=0;
1.Gossip(periodically do)
if !$converged$ then
    $targets = getNeighbors(fan-out)$;
    for $i$=$1$:$fan-out$ do
        $send(push, message(S, mms), targets(i))$;
2.Handle incoming messages
$messg$=accept();
if $messg.Type == push$ then
    $avg$ = AVERAGE($S$, $messg$); $S$.update($avg$); send($pull$, $avg$, sender)
else if $messg.Type == pull$ then
    $S$.update($messg$);
$reciprocalSizeEstim$=$messg$.getVal(ui);
if $ISCONVERGED(convLimit, \varepsilon)$ then
    $converged$=$true$;
Query
$F$ ={$item$ | $\forall$ $item \in S$, $S$.getAvgFrequency($item$)*$\frac{1}{reciprocalSizeEstim}$; $> T$ };
```

Parameter Name	Usage	Description
T	Threshold Mechanism	threshold value used to distinguish frequent items from infrequent items
fan-out	Aggregation Protocol	number of peers to whom gossip message is sent at each round
mms	Aggregation Protocol	maximum gossip message size a peer can send.
ui	Aggregation Protocol	unique item used in network size estimation
convLimit	Convergence Rule	number of successive rounds a peer needs to satisfy epsilon condition (see Sect. 3.6) in order to converge
ε*(epsilon)*	Convergence Rule	parameter used to determine epsilon condition

Table 3.2: Algorithm parameters

system size estimation information. In ProFID, we use gossip-based aggregation technique to compute the system size both due to its stated advantage and easily adaptable properties. It is easily adaptable because ProFID already uses gossip-based aggregation technique to compute the global frequencies of items. In order to calculate system size an initiator peer adds a unique item type named *ui* in its local item set. The local frequency of this item is set to 1. Since only one peer has that unique item, average frequency of that item would converge to $1/N$ at the end of the algorithm.

3.5 Atomic Pairwise Averaging

In order to calculate global frequencies of items, we use pairwise averaging function with network size estimation. Our pairwise averaging function uses push-pull scheme meaning that a peer sends its state (in a push gossip message) to a target peer and the target peer performs averaging operation using its own state and incoming state, then replies the average (in a pull message) back to the sender. Then, sender updates its state. By this way, a single push-pull based pairwise averaging operation is completed. In order to prevent misleading calculations this operation must be performed atomically.

During atomic pairwise averaging, a peer resides in different states as follows: (see Fig. 3.2).

- *Timer-interrupt*: With a timer-interrupt, either a new epidemic round of a peer starts and the peer sends push gossip message, or current round ends.

- *Receive push*: Upon receiving a push gossip message, a peer may perform one the following three operations:

  - If the peer is waiting for a pull message from any other peer, then incoming push gossip message is buffered. Details of buffering mechanism will be discussed in Sect. 3.5.2.
  - If the peer receives a push gossip message from a peer it waits a pull gossip message from, deadlock case may occur (see Fig. 3.4). In order to prevent this, the peer performs atomic pairwise averaging operation and updates its state, but it does not reply with a pull gossip message. Details of deadlock cases will be discussed in Sect. 3.5.3.
  - Otherwise, the peer performs atomic pairwise averaging, updates local state and sends the reply back to the sender (of push gossip message) in a pull gossip message.

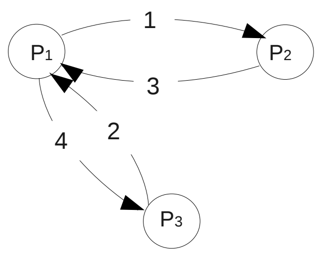
- *Receive pull*: Upon receiving a pull gossip message, a peer either performs an atomic pairwise averaging and updates its state, or buffers the message.

- *Send push*: Upon sending a push gossip message, a peer starts waiting for its corresponding pull gossip message. Until receiving the corresponding pull gossip message (or timeout event), peer cannot perform any other operation such as sending another push gossip message or handling an incoming push gossip message. It puts them into the buffer, instead. The peer either gets the corresponding pull gossip message in time and updates its state, or timeout occurs. In both cases, the peer immediately checks the buffer for the next operation. If fan-out is greater than 1, next push message is sent since sending a push message has higher priority than responding to an incoming message.

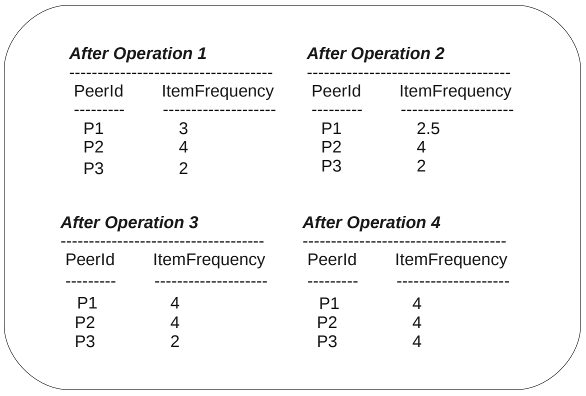
- *Send pull*: A pull gossip message is sent immediately after incoming push gossip message is read from the buffer and averaging operation is completed. (see the last operation in *Receive push*)

3.5.1 Illustrative Example: The Result of a Non-atomic Pairwise Communication

Fig. 3.3a illustrates the order of an example of push-pull based non-atomic pairwise averaging operation. Assume that system has a unique item named *item1* with global frequency 10 and initial local frequencies are $f_{1,item1}$=3, $f_{2,item1}$=5, $f_{3,item1}$=2. The updated local frequencies of *item1* after each operation shown in Fig. 3.3b. When states of all peers are considered after operation 4, Eq. 3.1 does not hold any more, which puts the system in an inconsistent state. Note that result converges to 4 instead of $\frac{10}{3}$. Thus, even though no addition/removal of item, or loss of a message occurs, global value of item changes, which contradicts with *mass conservation* [22].



(a)



After Operation 1

PeerId	ItemFrequency
P1	3
P2	4
P3	2

After Operation 2

PeerId	ItemFrequency
P1	2.5
P2	4
P3	2

After Operation 3

PeerId	ItemFrequency
P1	4
P2	4
P3	2

After Operation 4

PeerId	ItemFrequency
P1	4
P2	4
P3	4

(b)

Figure 3.3: (a)Illustration of operation order of a non-atomic pairwise averaging (b)States of peers after each operation

3.5.2 Buffer Mechanism for Atomic Pairwise Averaging

This section is devoted to the details of how the atomicity is achieved in pairwise averaging using buffering mechanism.

In a distributed environment, the main issue is to control the interaction and order of events because of independent terminals with different computing power and capabilities. Besides that, network layer is also problematic in terms of control of events due to varying latencies and message drops. Therefore, we can not control the message order or atomicity in the network layer. Instead, we need to use either a TCP (Transmission Control Protocol) like retransmission mechanism or a buffer mechanism to ensure atomicity in pairwise averaging in application layer. Retransmission causes high communication overhead, so we use a buffer mechanism in such a way that whenever a peer sends the first push gossip message to another peer, all the requests coming into that peer or other push gossip messages that will be sent in this round (in case of fan-out>1) are buffered until the reply of the first push gossip message reaches. After the arrival of the reply of the first push gossip message, second push gossip message is read from the buffer if fan-out is larger than 1. After completing to send all the push gossip messages, peer checks the buffer for incoming push gossip messages. If there are any, peer computes the average of local and incoming gossip message state and sends the reply (in a pull gossip message) back to the sender. Furthermore, to deal with message losses, we use a timeout mechanism. If the response of a message does not arrive within a predefined time period, peer stops waiting for the response in order not to wait infinitely for the response in case of message losses.

For the example in Fig. 3.3, we may perform operations atomically using buffering as follows: P_1 sends push gossip message to P_2 (*event 1*) and starts waiting for the corresponding pull message. Push gossip message came from P_3 (*event 2*) is not replied immediately, but buffered. Whenever the reply comes into P_1 from P_2 (*event 3*), P_1 updates its state, then removes and performs the next event in the buffer which is sending pull message to P_3. After sending the pull message, P_3 receives the pull message and updates its state (*event 4*). By means of buffering and timeout mechanisms, we perform push-pull based pairwise averaging operation atomically.

3.5.3 Deadlock Prevention

Maintaining atomicity may cause deadlock since peers may wait each other infinitely, which is known as *circular wait* [40]. In ProFID, there can be two cases of deadlock. First, two peers may send push gossip messages to each other simultaneously (Fig. 3.4). Second, multiple peers may send push gossip messages simultaneously in such a manner that a circle is constructed (Fig. 3.5). We handle these 2 cases differently because in the former case we don't need a circle detection mechanism, we only need the sender information of incoming message.

There are two schemes to prevent deadlock: deadlock avoidance and deadlock prevention. We use the latter scheme because former scheme requires a central administration that will decide which process should wait using their resource necessities. In order to prevent deadlock, at least one of the four conditions should not hold: mutual exclusion, hold and wait, no preemption, and circular wait. We solved the deadlock problem by preventing circular wait if only 2 peers are involved in circle. This is achieved if both peers perform only averaging operation using local state and incoming state without sending a pull message back to the sender. Peers can detect this deadlock case without any communication overhead. A peer just needs to check if it receives a push gossip message from the peer it waits for a pull message. If more than two peers are involved in a circular wait, then we use timeout mechanism to preempt waiting peers. If timeout occurs, peers cancel the current operation and continue with the next operation in their buffer.

Consider a case that a peer sent a push gossip message and waits for a pull message. However, push gossip message is lost and did not reach the destination peer. In that case, pull message will never reach to the peer that sent push gossip message, and it will wait infinitely. This kind of problems are actually very common in network layer and they are handled by a timeout mechanism. It is a mechanism such that a peer sending a message waits for at most a predefined time interval. If response comes within that period, peer continues with the next task. Otherwise, timeout occurs and peer stops waiting for the reply. Loss of a push gossip message is not the worst case when compared to late arrival or loss of pull, since in case of a push gossip message loss, no update occurs at each side of the pairwise averaging operation. This will just increase the convergence time but will not drop the accuracy. If push gossip message is not lost, but either pull message is lost or does not arrive in time, timeout will occur on the push-sender peer. This is the worst case scenario because only pull-sender peer updates its state in a pairwise averaging operation. This will cause a drop in accuracy since *mass conservation* fails. In case of a timeout, a peer checks the buffer if there are any waiting operations.

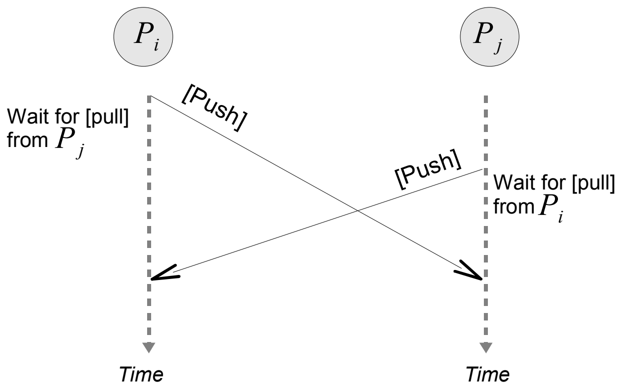


Figure 3.4: Circular Wait Scenario (for 2 Peers)

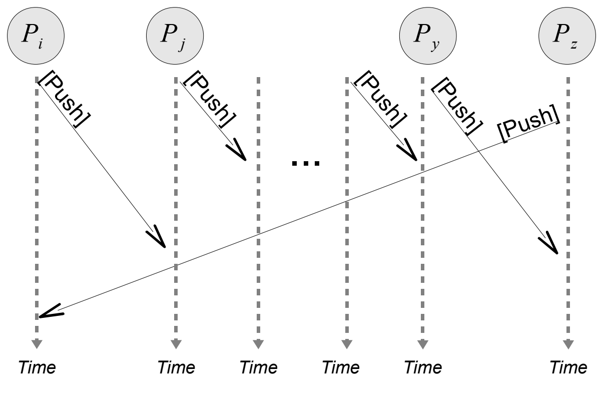


Figure 3.5: Circular Wait Scenario: Each peer is waiting for a pull message to send reply.

3.6 Convergence Rule

In ProFID, peers use parameters ε and *convLimit* to determine whether the algorithm converged and frequent item set results are available. The idea of convergence is simply to compute *similar frequency* values within at least a time length of *convLimit* gossip rounds. Here, computing *similar frequency* means getting two average frequency estimation values that change at most ε percentage in consecutive rounds of algorithm (see Alg. 3). When a *similar frequency* is calculated, then a counter ,initially zero, is incremented. Otherwise, counter is reset to zero. Whenever, the counter reaches *convLimit* value, then peer decides that algorithm converged and can use frequent item set in a required service. As noted in [22], initial distribution of an item does not affect the convergence speed, hence we use *ui* value's average frequency estimation to determine the convergence of the algorithm.

Convergence Rule. Let θ_t be the average frequency estimation of item *ui* at time *t*. *Epsilon condition* check is performed at each gossip round and *convCounter*, starting from zero, is set accordingly. Whenever the *convCounter* reaches to *convLimit*, then peer decides that algorithm has converged.

Definition 3.1 *Epsilon condition:* A peer satisfies this condition if current frequency of *ui* at that peer changes at most ε percentage after the last gossip.

$$convCounter = \begin{cases} convCounter + 1, & \text{if } 100|\frac{\theta_t - \theta_{t-1}}{\theta_{t-1}}| \le \epsilon, \\ 0, & \text{otherwise,} \end{cases}$$

Algorithm 3: *ISCONVERGED: the convergence rule for ProFID.*

```
Input: $convLimit$, $\varepsilon$, $\theta$
Output: $converged$: boolean representing convergence

if $\varepsilon \ge (\theta - prevSizeEstim)/prevSizeEstim$ && $\theta$ != 0 then
    $convCounter$++
else
    $convCounter = 0$
$prevSizeEstim = \theta$;
return $converged$ || $convCounter$==$convLimit$
```

In order to prevent misleading convergence cases at the beginning of the algorithm due to the similar initial states of neighbors, we keep *convLimit* not less than a certain value so that peers can not decide to converge in the early stages of the algorithm.

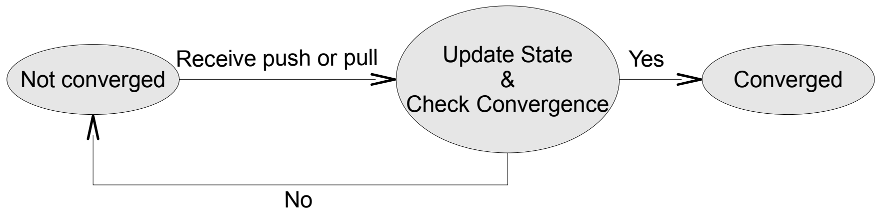


Figure 3.6: Peer Convergence State Diagram

Convergence check of a peer is performed after each receipt of push/pull gossip messages. Upon receiving a push/pull gossip message, peer uses *convergence rule* to decide whether it converged or not as shown in Fig.3.6, and takes actions accordingly. If a peer decides that it converged, it stops sending push gossip messages to its neighbors, yet it continues to reply incoming push gossip messages in order to contribute other peers' convergence. Replying push gossip messages may continue a few more rounds (∼10 rounds) since peers converge approximately at the same round. Once a peer is converged, it stays at converged state [1]. Then, information is sent to the application process that needs frequent items in the system. Otherwise, peer continues sending and receiving gossip messages.

Since we design a fully distributed protocol, each peer makes a local decision about convergence. In terms of state exchanges, our gossip operations for averaging are similar to [22]. However, termination of the algorithm in that study uses topology information which may not be available at all peers initially. On the other hand, ProFID uses no system-wide information which makes it more practical, less costly, and easy to deploy.

As another related work, [23] gives a probabilistic upper bound on the number rounds for all peers to converge. This upper bound depends on network size and accuracy. It is shown theoretically that the more the algorithm runs, the more probable to get correct

[1]a converged peer can not change its state to *not converged* state since convergence means result was computed and sent to application process

estimations. However, the problem in the proposed algorithm is that peers may not have knowledge about all items in the network, which will cause accuracy drop in the algorithm.

3.7 Discussion of Analytical Findings

Convergence of the gossip-based atomic pairwise averaging aggregation is discussed in this section. A peer periodically exchanges its state with neighbors until *ui* item in that peer does not change more than $\varepsilon\%$ in *convLimit* consecutive rounds. Then, in order to calculate the frequencies of items the average frequencies of items is multiplied by network size, as discussed in Sect. 3.5. However, can one make sure that algorithm converges to the correct result? The answer is 'no' since gossip based approaches always give probabilistic result and in order to get 100% accurate result, algorithm needs to run indefinitely which is not practical. Gossip-based algorithms are expected to give at most an upper bound on the error.

Another question is how the result converges to the global average by performing atomic pairwise averaging operations with other peers. Consider a network with a single item which is distributed to multiple peers. Let's define a vector, w, whose elements represent the local frequencies of that item on each peer [22].

$$w_0 = w_{0,1} + w_{0,2} + \cdots + w_{0,N} \tag{3.5}$$

where $w_{i,j}$ is the frequency of an item on peer j at round i. Consider also a function, *avg* which takes w as an input and replaces two random elements with the average of those elements in place N times as described in Alg. 4 (see also Eq. 3.6). We, for simplicity, analyzed the uniform gossiping meaning that every peer may communicate with every other peer.

```
Algorithm 4: avg operation
  Input: $w$: input vector
  for $k = 1$ to $N$ do
      $i,j$ ← choose two random elements from [1,N]
      $w_i \leftarrow w_j \leftarrow \frac{w_i+w_j}{2}$
  return w
```

$$w_{i+1} = avg(w_i) \tag{3.6}$$

Note that *avg* operation is performed atomically in ProFID , and hence performing multiple *avg* operations in a single round does not create a problem; on the contrary, it speeds up the convergence. *avg* function does not change the mean of the vector elements.

$$\overline{w_0} = \overline{w_i} \tag{3.7}$$

where $\overline{w_i}$, mean of the vector elements at round i, is

$$\overline{w_i} = \frac{1}{N} \sum_{k=1}^{N} w_{i,k} \tag{3.8}$$

and the variance of values at round i is

$$\sigma_i^2 = \frac{1}{N-1} \sum_{k=1}^{N} (w_{i,k} - \overline{w_i})^2 \tag{3.9}$$

Eq. 3.9 implies that if σ_w^2 is zero, each peer keeps the correct average value. Hence, the aim is to minimize σ_w^2. For the purpose of analysis, centralize the initial vector. In other words, remove the mean of the vector elements from each element.

$$w_{0,k} = w_{0,k} - \overline{w_0}, \qquad k = \{1, 2, \cdots, N\} \tag{3.10}$$

This is just to simplify the following equality:

$$E(\sigma_w^2) = \frac{1}{N} \sum_{k=1}^{N} E((w_k - \overline{w_0})^2) \tag{3.11}$$

If the elements are independent random variables with zero mean, then we can rewrite Eq. 3.11 as:

$$E(\sigma_w^2) = \frac{1}{N} \sum_{k=1}^{N} E(w_k^2) \tag{3.12}$$

Now, using Eq. 3.10 and 3.12, it can be concluded that if expected value of σ_w^2 goes to zero that means variance of vector elements tend to zero and all elements will converge to the actual average $\overline{w_0}$.

LEMMA. Let *w'* be the vector obtained after replacing w_i and w_j by $\frac{w_i+w_j}{2}$ in vector *w*. If *w* is composed of uncorrelated random variables with mean 0, then expected value of reduction in σ_w^2 is given by

$$\begin{aligned}
E(\sigma_w^2 - \sigma_{w'}^2) &= E(\frac{1}{N-1}\sum_{k=1}^{N} w_k^2 - \frac{1}{N-1}\sum_{k=1}^{N} {w'_k}^2) && (3.13)\\
&= E(\frac{1}{N-1}\sum_{k=1}^{N}(w_k^2 - {w'_k}^2)) && (3.14)\\
&= E(\frac{1}{N-1}(w_i^2 - 2(\frac{w_i+w_j}{2})^2 + w_j^2)) && (3.15)\\
&= E(\frac{1}{N-1}(\frac{w_i^2+w_j^2}{2})) && (3.16)\\
&= \frac{1}{2(N-1)}E(w_i^2) + \frac{1}{2(N-1)}E(w_j^2) && (3.17)
\end{aligned}$$

Note that Eq. 3.16 is obtained from Eq. 3.15 using the fact that w contains uncorrelated random variables, and hence satisfies

$$E(w_i w_j) = E(w_i)E(w_j) = 0 \tag{3.18}$$

Moreover, note that obtained result is always positive so each *avg* operation will reduce the variance; therefore, variance will be approxiamtely 0 if *avg* operation is applied 'enough' number of times. It is why this operation is seen as elementary variance reduction step in [22]. Remember that *avg* also preserves the sum; hence, considering those two properties, it can be concluded that each value in the vector will approximately converge to the actual average.

3.8 Threshold Mechanism

Threshold is the parameter that is used to differentiate the frequent items from infrequent items. To the best of our knowledge, two types of thresholds have been used in literature for FID problem. First is regular threshold in which there is a single value that separates frequent item region from infrequent item region (see Fig. 2.4a). Second is Lahiri's threshold in which an *indecisive region* is used to prevent taking wrong decisions on boundaries. The idea behind this approach is that making a wrong decision is worse than not making any decision. If the frequency of an item is larger than the maximum value in that region than that item is considered as frequent. On the other hand, if item frequency is less than the minimum value in that region it is considered as an infrequent item. Finally, if it is in *indecisive region*, algorithm does not give any decision. Note that, if Δ is 0, then this technique becomes a *regular threshold.* By using an *indecisive region*, we relax the threshold and prevent a strict decision rule. Consider an example with a threshold 10000 and assume that there are two different items with frequencies 9995 and 10005 respectively. With a strict decision rule, we would identify the former as infrequent, while the latter as frequent, which does not seem to be reasonable because such a small difference should not cause a distinct decision.

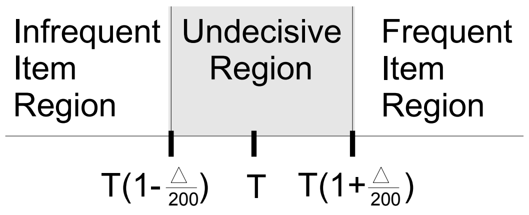


Figure 3.7: Centered threshold

We propose another threshold technique, namely *centered threshold*, which is similar to Lahiri's threshold technique but indecisive region is shifted to the right so that threshold is centered as shown in 3.7. We compared the accuracy of all three techniques. As expected, *centered threshold* performs better, since in ProFID estimated frequencies might be more or less than the actual frequency of item due to our convergence rule (see details in Sect. 4.4.6).

Chapter 4

PERFORMANCE ANALYSIS

We evaluated the efficiency of atomic pairwise averaging and ProFID protocol through extensive simulations. We used various performance metrics in our analysis. Furthermore, we compared ProFID with a well known Push-Sum protocol which we modified to adapt it to FID problem. We used PeerSim simulation environment to model ProFID and adaptive Push-Sum protocols since it is more scalable than most of the simulators. It also provides transport layer configuration so that we can test the robustness of algorithms in case of peer or message losses. Moreover, it supports both structured and unstructured networks. The most attractive property of PeerSim is its extensible and pluggable components which makes it easy to come up with a new protocol.

We evaluated the behavior and performance of ProFID through extensive large-scale distributed scenarios (up to 30,000 peers) on PeerSim. We tested different topologies such as scale-free Barabasi-Albert topology, random topology, and transit stub topology with average degree around 10 in the experiments. During experiments, network is static unless otherwise stated. All the simulation data points presented in graphs are the average of 50 experiments unless stated otherwise.

4.1 Experimental Setup

In this chapter, P2P simulator used to model ProFID, namely PeerSim [9], is introduced and explanation of why this simulator is found to be favorable among others is given. Then, in Sect. 4.1.2, we explain the simulation environment. Next, in Sect. 4.1.3, we discuss how we implemented ProFID and configured the overlay network in PeerSim.

4.1.1 P2P Network Simulators Review

There are several network simulators used in P2P simulations. In order to choose the proper simulation environment, we used the following set of criteria [41]:

Name	Url	Language
ns-2	http://www.isi.edu/nsnam/ns/	C++
PeerSim	http://peersim.sourceforge.net/	Java
P2PSim	http://pdos.csail.mit.edu/p2psim/	C++
Query-Cycle Sim.	http://p2p.stanford.edu/	Java
Narses	http://sourceforge.net/projects/narses	Java
Neurogrid	http://www.neurogrid.net/	Java
GnutellaSim	http://www-static.cc.gatech.edu/computing/compass/gnutella/	C++
GPS	http://www.cs.binghamton.edu/wyang/gps/	Java
myNS	http://www.cs.umd.edu/suman/research/myns/index.html	C++
Overlay Weaver	http://overlayweaver.sourceforge.net/	Java
DHTSim	http://www.informatics.sussex.ac.uk/users/ianw/teach/dist-sys/dht-sim-0.3.tgz	Java
VPDNS	http://p2p.cs.mu.oz.au/software/vpdns/	C
PlanetSim	http://planet.urv.es/planetsim/	Java

Table 4.1: Commonly used network simulators

- *Simulation Architecture* : the design and functioning of the simulator, how node behaviors are simulated.
- *Underlying Network Simulation* : which properties of the network layer can be simulated.
- *Scalability* : how the protocol scales to high number of nodes (thousands of nodes).
- *Statistics* : how valuable and informative the output is.
- *Usability* : how easy to get used to the simulator, if the simulator is well documented.

We took commonly used P2P network simulators into consideration as stated in study [41]. Two of the well known simulators were ns-2[42] and PeerSim[9]. A list of simulators can be found in Table 4.1. Especially, ns-2 is the best known and most frequently used network simulator; however it is designed to perform network layer simulations and not scalable to thousands of peers.

On the other hand, PeerSim is specifically designed for epidemic algorithms but it can also be used for other protocols as well. Moreover, it can scale up to 10^6 peers and it is modular so that new components can be plugged without modifying existing infrastructure.

4.1.2 PeerSim: A Peer-to-Peer Simulator

We used PeerSim simulator to build the model for ProFID. It is a java-based modular and very scalable simulator and gives us a way to access and configure transport layer properties such as message loss and delays. It is also highly pluggable and extendable so new protocols can be added easily 4.1. User can also create components to gather statistics about the network at any time, which eases the analysis part of our algorithm. Some of the well known network topologies such as ScaleFreeBA (Barabasi-Albert Model), Watts-Strogatz Model, star, and random topologies are ready to use, as well as a tree structure. Configuration of parameters such as network size, gossip round length, and link failure probability are easily done using a configuration file.

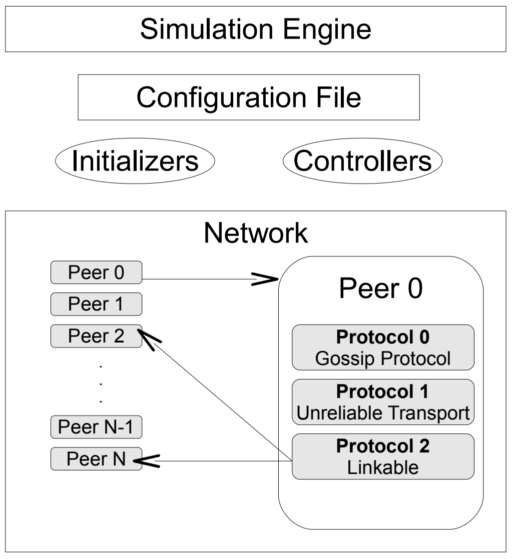


Figure 4.1: PeerSim Architecture

There are two types of simulation engines in PeerSim: cycle driven and event driven. In *Cycle driven* engine, up to 10^6 peers can be simulated. However, there is no transport layer simulation and messaging. It is specifically designed to schedule periodic behaviors of epidemic algorithms. *Event driven* engine is more complex and realistic. It supports transport layer simulations and it can scale up to $\sim$ 3x10^5 peers. It can also be used together with cycle driven engine to model more complex epidemic algorithms with ease.

4.1.3 ProFID in PeerSim

PeerSim is highly configurable and new protocols can be plugged easily without modifying the infrastructure. ProFID has 3 main mechanisms we plugged into PeerSim as shown in Fig. 4.2: Atomic Pairwise Averaging, Convergence Rule, and Threshold Mechanism. Application process requests frequent items which triggers the ProFID, and atomic pairwise averaging operations start via push-pull communication with other peers. Then, whenever the peer converges, it stops sending push message and continues to communicate just by answering requests of other peers. After convergence, computed average aggregate values are compared with threshold and application process is informed about the frequent item set.

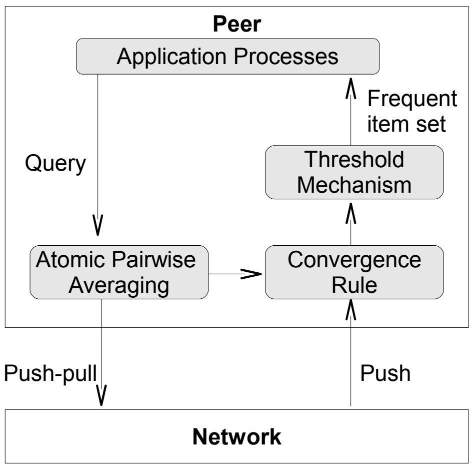


Figure 4.2: ProFID Architecture

4.2 Performance Metrics

We use various performance metrics to measure the efficiency and accuracy of atomic pairwise averaging, threshold mechanism, and ProFID.

We consider the following performance metrics in our analysis:

- **Accuracy:** This metric measures how approximate the estimated result to the actual result. We analyzed the types of accuracy: accuracy of average item frequencies, accuracy of items being frequent or not.

- **Number of rounds (to converge):** This performance metric measures how fast the algorithm converges. The effects of convergence parameters, average degree of nodes, and number of nodes on convergence speed have been analyzed.

- **Messages sent per peer:** This performance metric measures the energy efficiency of the algorithm, the effects of convergence parameters and average degree of nodes on energy efficiency have been analyzed.

- **Percentage of converged peers:** This metric is another measure of how fast algorithm converges. It shows the percentage of peers that converged during each gossip round. The effects of this metric has been analyzed in terms of convergence parameters and network size.

We also analyzed the effects of following parameters:

- **epsilon and convLimit:** The effects of convergence parameters have been analyzed in terms of number of rounds to converge and average number of messages sent per peer. Relative error has also been analyzed for different combination of convergence parameters.

- **mms:** This parameter determines the size of a single gossip message. Its effects on convergence speed and network overhead have been analyzed.

- **C:** This parameter is the convergence time constant.The effects have been analyzed in terms of network size and percentage of peer that converged.

- **fan-out:** This parameter defines the number of peers to whom gossip message will be sent at each round, its effects on convergence speed and C value have been analyzed.

- **T:** Threshold parameter is used to differentiate frequent items from infrequent items and the effects of different threshold mechanisms on accuracy have been analyzed.

4.3 Efficiency of Atomic Pairwise Averaging

We evaluate the performance of pairwise averaging function by excluding the convergence rule of ProFID. In this analysis, a peer compares the actual averages of items with its estimated averages while deciding whether it converged or not. This provides better evaluation of the performance of pairwise averaging. During the analysis of efficiency of pairwise averaging, we use a random topology with average degree around 10. The default values of algorithm parameters are shown in Table 4.2, unless stated otherwise. Number of items in the network is uniformly distributed between 1 and 1000, and they are distributed to peers uniformly. Moreover, we use a regular threshold mechanism. We assume that network is reliable and timeout does not occur. This assumption is made to see the pure performance of pairwise averaging without network problems.

Parameter	Value	Parameter	Value
N	1000	convLimit	10
M	N/10	ε(epsilon)	10
mms	M	fan-out	1
T	500		

Table 4.2: Default parameter values

4.3.1 Scalability

Fig. 4.3 depicts the scalability of ProFID in terms of time complexity. Number of gossip rounds needed for the convergence of all peers in the system is measured to examine the time complexity of ProFID as the network size scales up to 30000 peers. This result confirms the scalability of our system in terms of time needed for convergence. In fact, our result (for number of rounds to converge) agree with the O(logN) time complexity of epidemic dissemination [43].

Fig. 4.4a illustrates the effect of *C*, convergence time constant, on the network size. It decreases logarithmically with the increasing network size from which we can conclude that for large networks, *logN* value will be much more dominant than *C*. Fig. 4.4b shows the effect of *C* on the number of peers that converges. It is another way of showing that C value decreases with increasing network size.

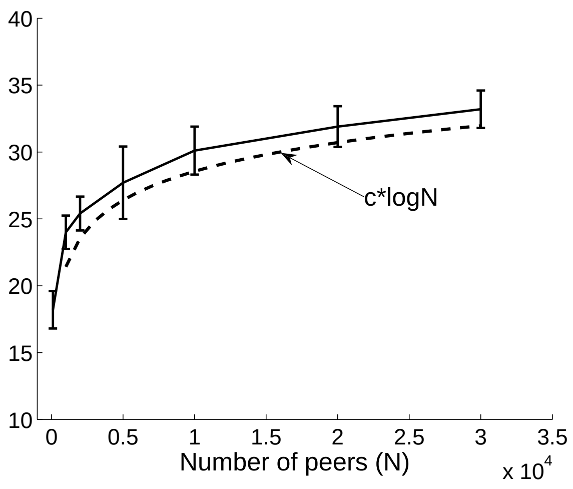


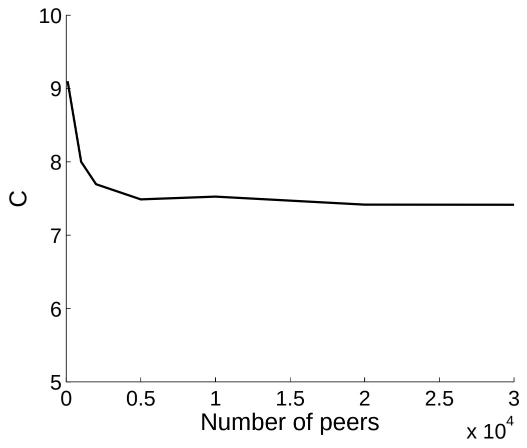
Figure 4.3: Number of gossip rounds needed for all peers to converge.

4.3.2 Effect of fan-out

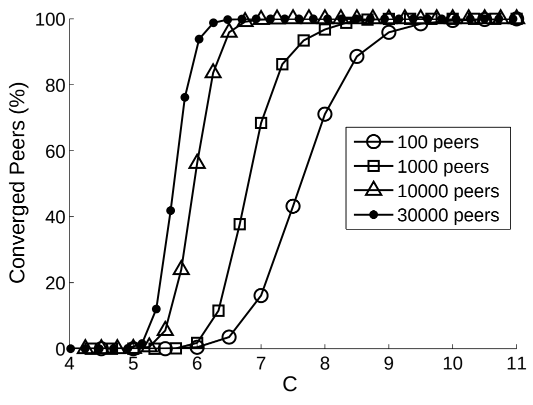
The effect of fan-out on convergence speed is illustrated in Fig. 4.5a. For larger values of fan-out, algorithm converges faster since a peer exchanges its state with more neighbors and its state is disseminated faster to the network. However, as depicted in Fig. 4.5b, total number of messages sent per peer until the convergence of the algorithm does not change since a peer sends more messages in a single gossip round. Note that while setting the value of fan-out, one needs to consider the gossip round length so that peers can complete all operations with in a single gossip round.

4.3.3 Link failures

Fig. 4.6 illustrates the convergence error of the atomic pairwise averaging in case of link failures and resulting message losses. In this simulation, the message loss probability of each link is independent and identically distributed. Relative error is even less than 1%, in case of 5% message drop probability, which is high enough when compared to real networks. Hence, we can conclude that atomic pairwise averaging is robust against independent link failures. However, in real-world networks, heterogeneity is inevitable; hence, simulations may not give totally correct insight about how robust a gossip protocol in a particular environment. It is still an open research area that to which kind of failures gossip based



(a)



(b)

Figure 4.4: (a)The minimum value of C such that after $ClogN$ rounds all peers converge. (b)Percentage of peers converged after $ClogN$ rounds

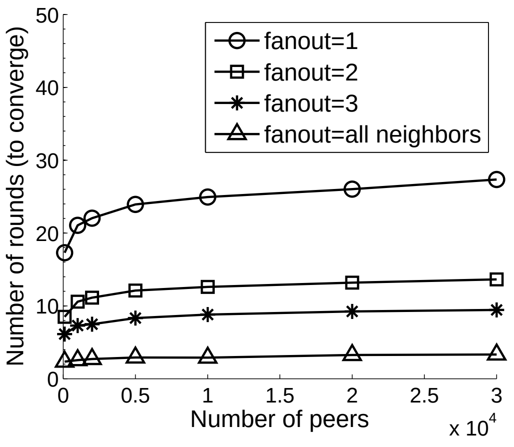
protocol is robust against.

4.4 *Efficiency of ProFID*

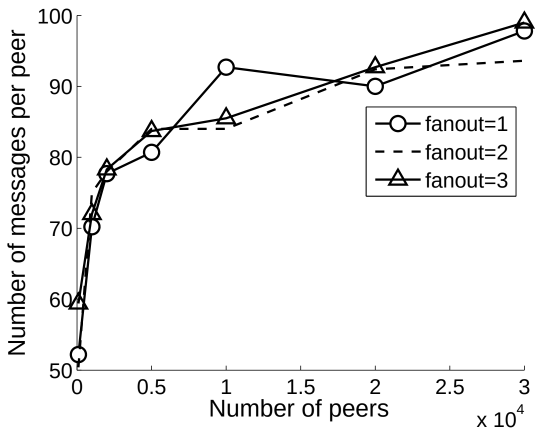
In this section, we will measure the efficiency of ProFID in terms of message complexity, convergence speed, and accuracy.

4.4.1 Simulation methodology

Popular P2P networks such as freenet, napster and gnutella were analyzed a lot in order to observe the characteristics of P2P networks. The open architecture and self-organizing structure of the Gnutella file-sharing network make it an remarkable P2P architecture to study. For this reason, in our simulations, we use the characteristics of gnutella network in terms of degree and item distribution and popularity (see Table 4.3). The pioneering study of scale-free networks [44] shows that the Barabasi-Albert (BA) model produces a power-law distribution, which is also known as internet-like topology, with exponent 3 independent of BA model parameter, hence while constructing the network we set the BA model parameter to 5 so that the average node degree is around 10 which is the average of some P2P studies [45, 46] (see Fig. 4.7). Besides a more realistic topology, we set the global frequencies of



(a)



(b)

Figure 4.5: (a)The effect of fan-out on the number of rounds needed for all peers to converge. (b)The effect of fan-out on number of gossip messages sent per peer.

Topology	BarabasiAlbert(k=5)
Number of items	N/10
Frequencies of each item	[1,N] (Zipf Distribution ρ =0.271)
Distributions of items	PowerLaw(p=4)
Threshold	N/2

Table 4.3: Simulation parameters used in the analysis of ProFID

items using zipf-like distribution with skew factor 0.271 [47, 48]. Moreover, we distribute those items to the network using a power-law topology with exponent 4 [48]. Each link delay is independent and uniformly distributed between 2ms and 100ms and timeout is 300ms. We set the gossip round length to 1sec, which is long enough for a gossip operation to be completed.

4.4.2 Effect of convergence parameters

We evaluate the effects of convergence parameters, namely ε and *convLimit*, and average degree on the efficiency of ProFID. Fig. 4.8a shows that increasing ε, decreases both av-

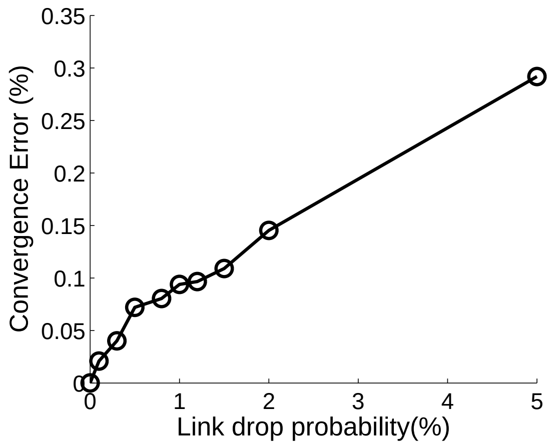


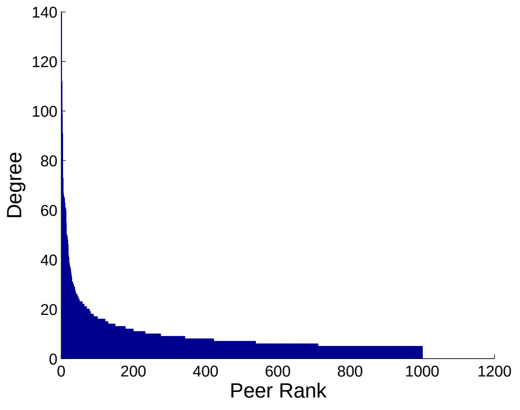
Figure 4.6: The effect of link drop probability on the accuracy of pairwise averaging.

erage number of messages sent per peer and number of rounds to converge because a peer increments *convCounter* value with more probability in each state update, which results in faster convergence. Since algorithm converges faster and number of gossip messages sent by each peer is constant, peers communicate less and average number of messages sent per peer decreases. In contrast to ε parameter, increasing *convLimit* increases both the average number of messages sent per peer and number of rounds to converge because *convCounter* needs to be incremented more to reach *convLimit* (see Fig. 4.8b).

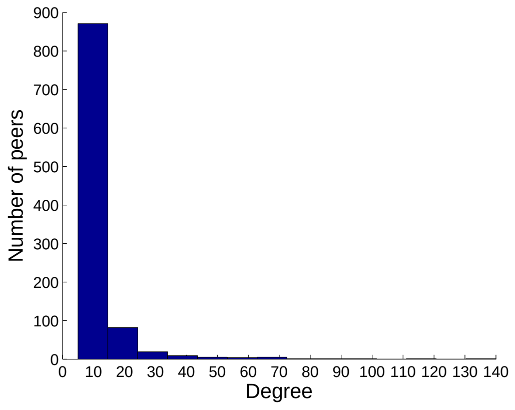
Fig. 4.9 illustrates the effects of convergence parameters, ε and *convLimit* on the number of rounds to converge. The fastest convergence occurs whenever ε parameter takes its largest value and *convLimit* takes its smallest value, which agrees with the convergence rule. However, there is a tradeoff between convergence speed and accuracy. Hence, user should select convergence parameters depending on the application requirement. If application requires a fast computation than one can choose larger ε and smaller *convLimit* values. If application requires high accuracy, then ε might be set to a smaller value, while *convLimit* might be set to a larger value.

4.4.3 Effect of average degree

Fig. 4.10 illustrates the effect of average degrees of peers on the number of rounds to converge. Increased connectivity of the network, decreases the number of rounds to converge.

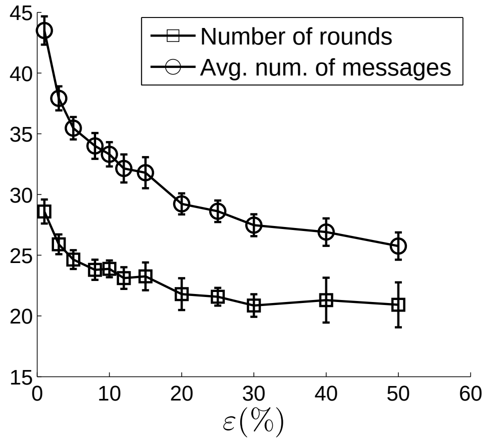


(a)

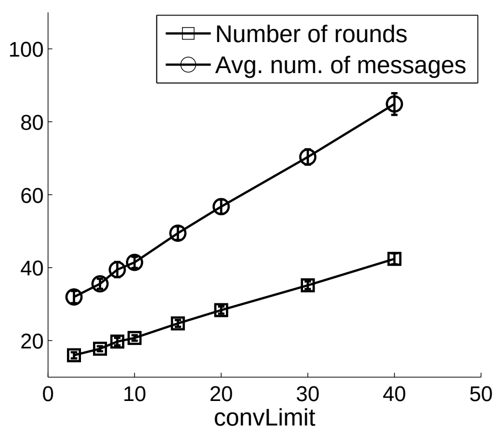


(b)

Figure 4.7: (a)Peer degree distribution. (b) Histogram of peer degrees.



(a)



(b)

Figure 4.8: The effects of ε(a) and *convLimit*(b) on the number of rounds to converge and average number of messages sent per peer

This actually shows that, optimum results would be taken in complete graphs, as Kempe et al. [23] showed.

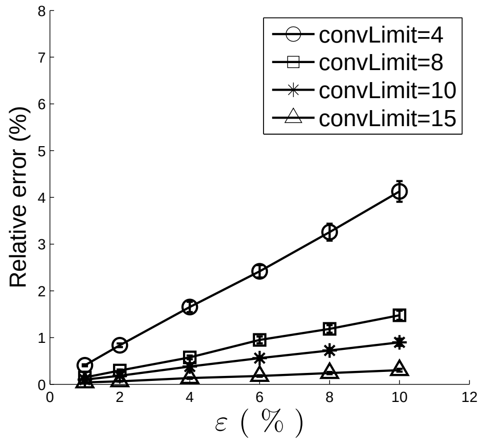


Figure 4.9: The effects of the convergence parameters on number of rounds to converge.

In order to compute frequent item set, peers share local states with each other. Consider a network consisting of millions of items. If peers iterate by sharing whole local states, it will be very costly to send state after a while since the size of the local states of peers increase by time. There are two solutions to this problem. First solution is that each peer chooses a subset of its local state to send in a single gossip message. This actually solves the problem of high bandwidth requirement but local state of a peer still gets larger and larger while receiving first-time- encountered items in gossip messages. Second solution is to filter out items before adding to local state in order to hinder local state from getting larger. We assumed that peer has enough capacity to store frequencies of all items, and used the former solution. However, using second approach with an intelligent filtering would be also a reasonable choice. Those two solutions might be even combined to make use of powers of both approaches.

In our simulations, we limit the gossip message size using a parameter, namely *mms*, which represents maximum number of <itemId,freq> tuples (a.k.a item in this context) sent in each gossip message. A peer chooses that much item from its state and puts into a gossip message to be sent. It is possible that a peer may not send the item with id *ui*

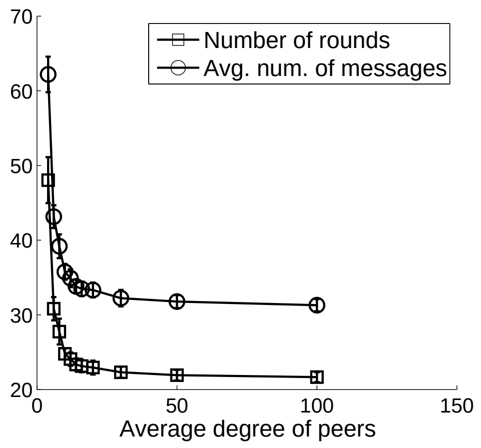


Figure 4.10: The effects of average degree on number of rounds to converge and average number of messages sent per peer.

in some gossip messages. If a peer receives such a gossip message, it bypasses convergence check, which means *convCounter* is neither reset nor incremented.

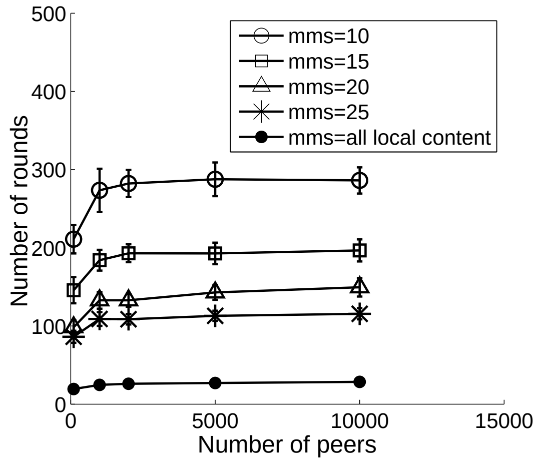
4.4.4 Effect of mms

Fig. 4.11a shows how many rounds it takes to converge if items are drawn uniformly random from local state. Convergence time is conversely proportional to *mms* because it takes more time to distribute items to the system if each peer distributes less item in each gossip message. Fig. 4.11b depicts the effect of *mms* on message overhead. Since peers converge faster for larger *mms*,on the one hand, average number of messages sent per peer decreases, but on the other hand, a gossip message size increases.

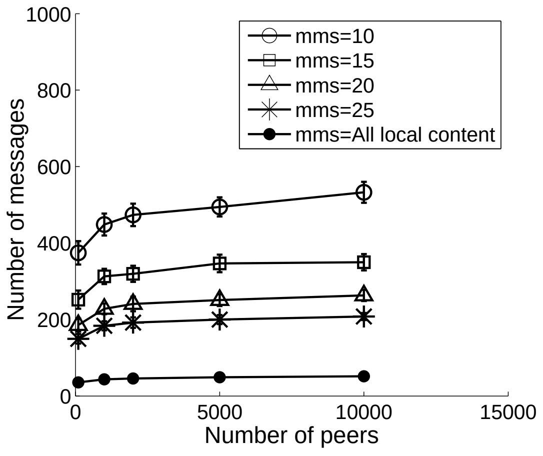
4.4.5 Gossip Target Selection

Gossip protocols always have some form of randomness in peer selection mechanisms. We changed some random behaviors of ProFID to some deterministic decisions to analyze how the algorithm's performance is affected.

First, we start with equal probability of selecting each neighbor as a gossip target. After selecting a peer as a gossip target, we multiply its probability with a constant value, namely



(a)

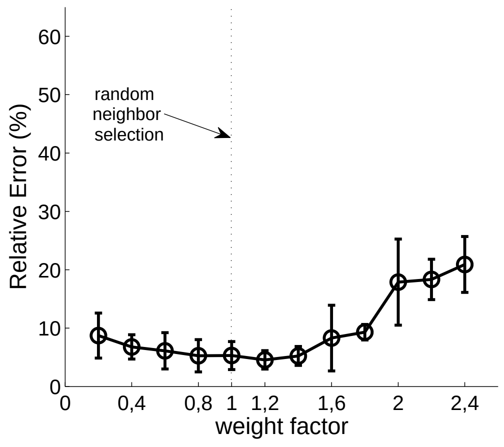


(b)

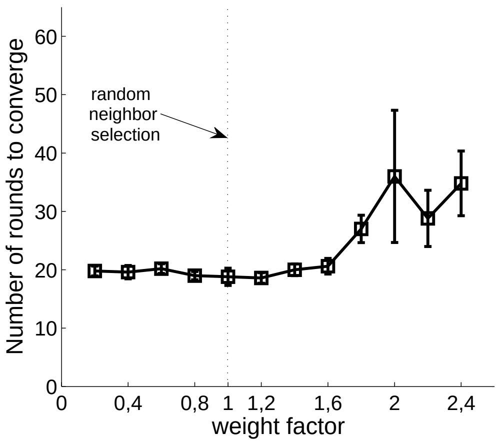
Figure 4.11: (a) The effect of *mms* on the convergence speed. (b) The effect of *mms* on average number of messages sent per peer.

weight factor. Note that if *weight factor* is larger than one means, then the probability of selecting a peer that has already been selected is higher than a peer that has never been selected before, and vice-versa. We see that optimum result in terms of both relative error and convergence time is obtained when *weight factor* is around one, which shows that selecting peers always with equal probability decreases both the relative error and convergence time as seen in Fig 4.12a and 4.12b respectively.

In another approach, we again start with equal probability of selecting each neighbor as a gossip target. Then during gossiping, peer gets information about the neighbors' degrees and sets the selection probabilities of neighbors inversely proportional to their degrees. This means that neighbor with a higher degree has a lower probability to be chosen as a gossip target than a neighbor with lower degree. The aim of this setup was to prevent the starvation of low degree peers and make all peers' contribution equal during gossiping. An example setup is given in Fig. 4.13. Fig. 4.14 depicts that random neighbor selection is better than neighbor selection method using neighbors' degree information in terms of accuracy. The reason might be that considering only the neighbor degree may not be enough, we may also need to consider the local clustering coefficient or other characteristics of the peer to get a



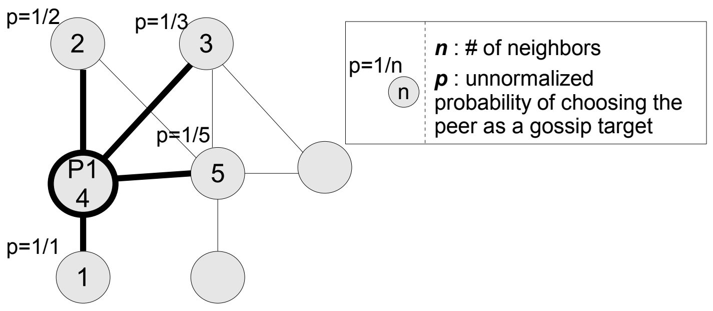
(a)



(b)

Figure 4.12: (a)The effect of weight factor on relative error. (b)The effect of weight factor on convergence time.

better accuracy than random neighbor selection.



p=1/2
p=1/3
p=1/5
p=1/1
2
3
5
1
P1
4
p=1/n
n
n : # of neighbors
p : unnormalized probability of choosing the peer as a gossip target

Figure 4.13: Neighbor selection using degree information of neighbors.

4.4.6 Threshold Mechanism

In this section, we analyze the effect of threshold on ProFID in terms of accuracy. We compare regular, centralized and Lahiri's threshold techniques using statistical measurements of sensitivity, specificity, accuracy, error, positive predictive value(PPV), and negative predic-

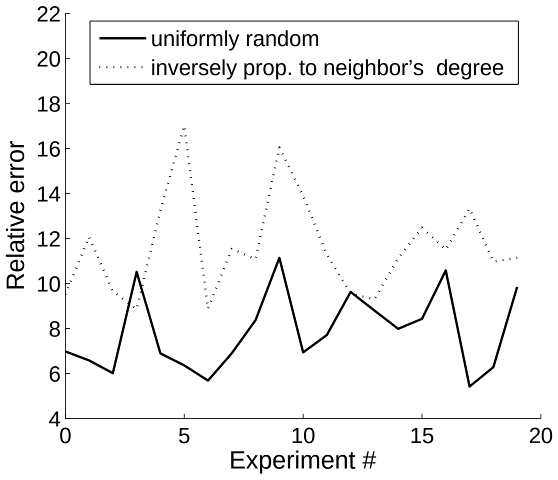


Figure 4.14: Neighbor selection based on degree information.

tive value (NPV). In regular threshold technique, items with frequency above threshold are called frequent, while items with frequency below threshold are called infrequent. In centralized and Lahiri's threshold techniques [32], there is an undecisive region and the algorithm does not make any decision on items that falls into this region. The difference in those two threshold techniques is the placement of undecisive region. In centralized threshold technique, the threshold divides undecisive region into half, whereas in Lahiri's threshold technique, the threshold lies on the right border of undecisive region (see Fig. 2.4 and 3.7). Explanations of these measures in frequent item set discovery context are given in Table 4.4.

Experimental results of each threshold technique are given in Table 4.5. The simulated network consists of 1000 peers and 100 items, and there are 24 actual frequent items whose frequency is above T (which is chosen to be 500). Actual item frequencies are depicted in Fig. 4.15. In all three approaches, the results are satisfactory, meaning that percentage of wrong decisions is very low when compared to correct decisions. For instance, the right-uppermost value, 23.815, is the value representing the average number of actual frequent items that were computed as frequent in 50 experiments. Note that this value needs to be 24 if all frequent items were computed as frequent in all experiments. Statistical measurement results such as sensitivity, specificity, accuracy, etc. (see Table 4.4) shows that centered threshold outperforms other techniques in all cases (Fig. 4.16). As expected, *centered*

Measurement Type	**Meaning**
Sensitivity ($\frac{TP}{TP+FN}$)	the proportion of actual frequent items which are correctly identified as such
Specificity ($\frac{TN}{TN+FP}$)	the proportion of infrequent items which are correctly identified as such
Accuracy ($\frac{TP+TN}{TP+TN+TP+FN}$)	probability of correctly predicting whether an item is frequent or not
Error ($\frac{FP+FN}{TP+TN+TP+FN}$)	probability of incorrectly predicting whether an item is frequent or not
Positive Predictive Value ($\frac{TP}{TP+FP}$)	the proportion of items computed as frequent which are actually frequent
Negative Predictive Value ($\frac{TN}{TN+FN}$)	the proportion of items computed as infrequent which are actually infrequent.

Table 4.4: Statistical measurements and their meanings

threshold technique performs better since, in ProFID, estimated frequencies might be more or less than the actual frequencies of items due to our convergence rule. In other words, we can not make sure that an item is frequent if its frequency is computed a little bit more than threshold. We can also not make sure that an item is infrequent if its frequency is computed a little bit less than threshold. Therefore, being indecisive around threshold reduces the number of wrong decisions. This is why *centered threshold* technique gives the minimum number of false decisions (see Fig.4.17).

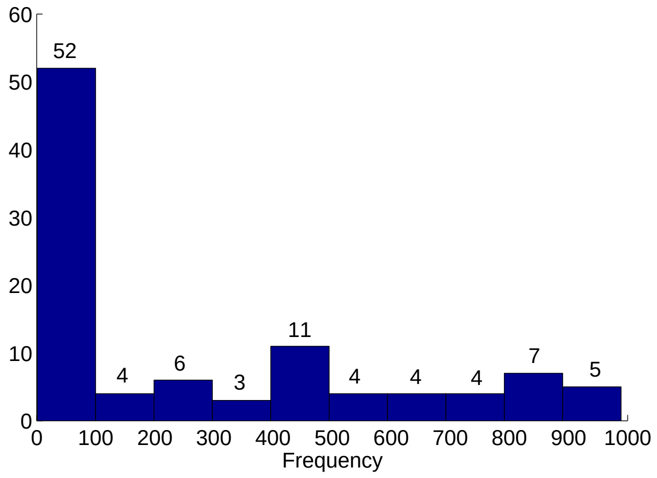


Figure 4.15: Histogram of item frequencies

			ACTUAL					
			Lahiri Threshold		Centered Threshold		Regular Threshold	
			Freq	non-Freq	Freq	non-Freq	Freq	non-Freq
PREDICTED	Δ=2	Freq	23,815	0,593	23,671	0,284	23,815	0,593
		non-Freq	0,116	74,753	0,116	74,753	0,185	75,407
		Undecisive	0,069	0,654	0,213	0,964	-	-
	Δ=5	Freq	23,867	0,075	22,881	0,064	23,867	0,433
		non-Freq	0,049	73,426	0,049	73,663	0,134	75,568
		Undecisive	0,084	1,906	1,070	2,275	-	-
	Δ=10	Freq	23,840	0,017	21,900	0,013	23,840	0,385
		non-Freq	0,028	71,321	0,028	71,564	0,161	75,617
		Undecisive	0,132	4,053	2,073	4,425	-	-

Table 4.5: Statistical measurement results of different threshold techniques

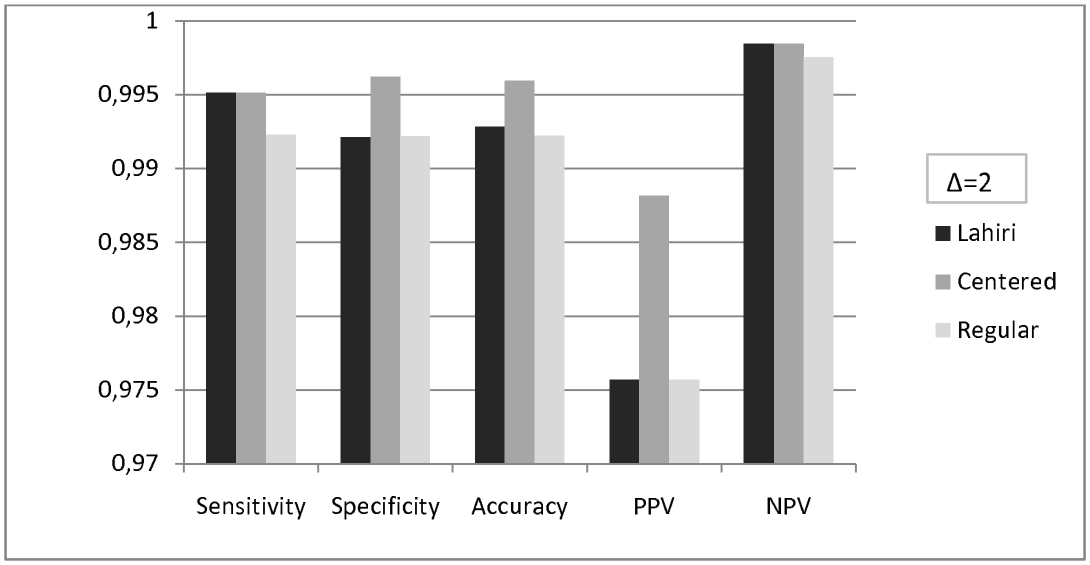


Figure 4.16: Statistical measure results of different threshold techniques

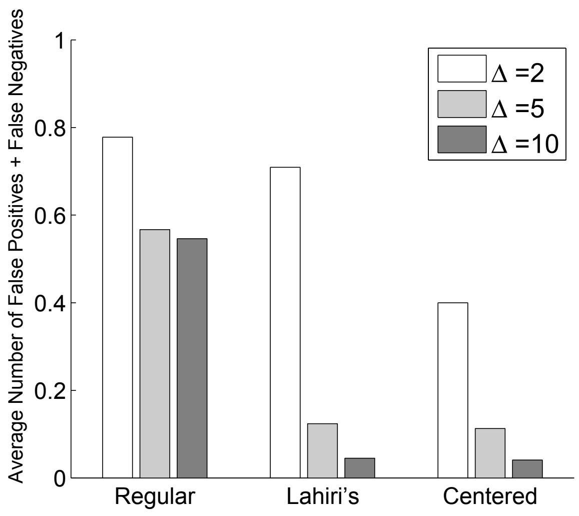


Figure 4.17: Average number of wrong decisions (False Positives + False Negatives).

Chapter 5

COMPARISON WITH PUSH-SUM PROTOCOL

In ProFID, peers use push-pull based aggregation scheme, which means they exchange their states mutually. On the other hand, push-based protocol is proposed in [23]. In this case, peers choose random peers only to send their state. The reason to use push-based protocol is that the correctness of the algorithm requires *mass conservation* and this is possible with only Push-Sum protocols. Push-Sum protocol can not be directly used in FID problem since peers are not aware of all items that exist in the network.

5.1 Adaptive Push-Sum Protocol

Consider the problem of discovering most downloaded files in a network, since almost all of the peers may not know about all items at the beginning, they have no chance to initialize their local value at time t_0. However, Push-Sum protocol (described in Sect. 2.1.1) assumes that if a peer has not downloaded a specific file, it initializes its sum to 0 without considering whether the peer is aware of that file or not. This is impractical in large networks since peers may have only a partial view of the network and can not store all file information in their local storages due to scalability issues.

For this reason, we modified Push-Sum protocol in order to adapt it to FID problem (Adaptive Push-Sum protocol) (see Alg. 5). It takes items arrived in previous round as input and adds up all values ($\widehat{s}_r$) and weights ($\widehat{w}_r$) separately (line 1-2). Whenever a peer p_i receives a message($\frac{1}{2}s_{t,j}$,$\frac{1}{2}w_{t,j}$) about an item it has not been aware before, it adds one to its total weight (line 3-4). This means that if a peer is informed about the item first time at time t, that peer pretends to send the item (with $s_{t,i}$=0, $w_{t,i}$=1) only to itself at time t in order to satisfy the *mass conservation* for both s and w. Then, peer sends half of its total value and weight to itself and randomly chosen neighbors (line 5-6). Finally, peer calculates the average estimate of this round as $\frac{s_{t,i}}{w_{t,i}}$ (line 7-8).

In general (fan-out>=1), peers choose multiple targets and send pairs as described in

Algorithm 5: *Adaptive Push-Sum Protocol*

```
Input: prevIncomingPairs: all incoming pairs at time t-1
Output: estimate: estimate of aggregate at time t
1 $(\hat{s}_r, \hat{w}_r)$=prevIncomingPairs;
2 $s_{t,i}= \sum_r \hat{s}_r$, $w_{t,i}= \sum_r \hat{w}_r$
3 if item is encountered first time then
4     $w_{t,i}= w_{t,i}+1$
5 target=chooseUniformlyAtRandom()
6 send($\frac{1}{2}s_{t,i}$, $\frac{1}{2}w_{t,i}$) /*to yourself and target*/
7 estimate=$\frac{s_{t,i}}{w_{t,i}}$
8 return estimate
```

Eq. 5.2 to their each target peer (note that for fan-out=1, it corresponds to the case in Algorithm 5):

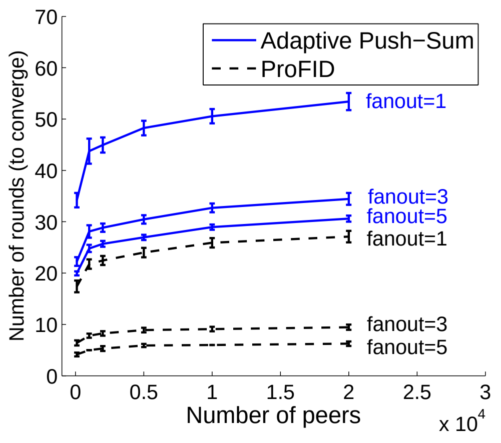
$$(\frac{1}{\textit{fan-out}+1}s_{t,i}, \frac{1}{\textit{fan-out}+1}w_{t,i}) \tag{5.1}$$

5.2 Comparison Results

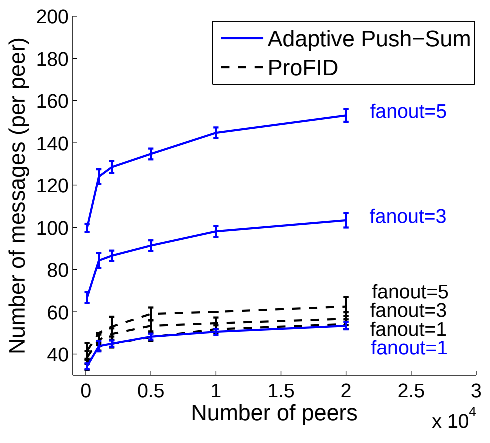
In this section, we compare ProFID and Adaptive Push-Sum algorithms in terms of message complexity and convergence time, and accuracy. We use Barabasi-Albert model in topology construction. This model constructs the topology by starting a small number of peers and adding new peers to the networks. Newly added peer is connected to the peer i with probability

$$\Pi(k_i) = \frac{k_i}{\sum_i k_i} \tag{5.2}$$

where k_i is the connectivity of peer i. This model produces scale-free power-law distribution with exponent 3. The average peer degree is set to 10. Item frequencies follows a zipf distribution and items are distributed to peers using a power-law distribution with degree 4. As depicted in Fig. 5.1a, ProFID converges faster than Adaptive Push-Sum algorithm for all fan-out values. The reason for faster convergence is that ProFID increments/resets



(a)



(b)

Figure 5.1: (a) Comparison of ProFID and Adaptive Push-Sum protocol in terms of convergence time.(b) Comparison of ProFID and Push-Sum protocol in terms of message complexity.

convLimit after each incoming message, whereas Adaptive Push-Sum algorithm does this operation after each round since aggregate is updated once in each round. Fig. 5.1b shows that ProFID is at least as much efficient as Adaptive Push-Sum in terms of message complexity for all cases. The reason of this difference might be faster item dissemination in push-pull schemes than push-based or pull-based schemes [49]. It can also be concluded that Adaptive Push-Sum is more sensitive to fan-out in terms of message complexity.

ProFID gives more accurate results when compared to Push-Sum protocol as depicted in Fig. 5.2. This difference in accuracy might be because of better performance of push-pull scheme aggregation than push-based aggregation scheme. Moreover, the convergence rule of ProFID may not fit the Push-Sum protocol and decrease its accuracy. It might have better accuracies for different convergence parameters, such as smaller ε or larger *convLimit*.

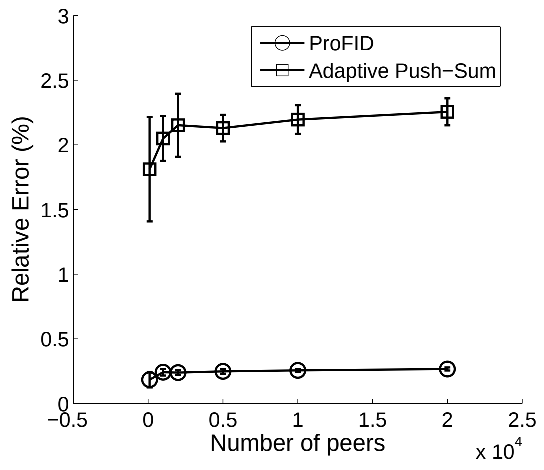


Figure 5.2: ProFID vs. Adaptive Push-Sum protocol (accuracy).

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis, we proposed a practical protocol named ProFID for discovering frequent items in unstructured P2P networks, developed an implementation on PeerSim simulator, and evaluated/compared its behavior extensively. In contrast to prior works, ProFID uses atomic pairwise averaging function with gossip-based aggregation. It is fully distributed, uses local peer information and neighborhood knowledge only. It also offers a convergence rule for efficiently approximating system wide averages of items without using any global knowledge such as overlay topology information.

In this thesis (Chap. 1), we have discussed the importance of the problem and the variety of application areas utilizing frequent item set discovery protocol on different fields such as P2P networks, database, and network security. Lastly, we have stated our contributions.

Then (in Chap. 2), we have described the previous works done on aggregate computation and frequent item set discovery, as well as threshold mechanisms. Even though various approaches have been proposed, they generally lack either analytical analysis or make unrealistic assumptions for analytical analysis, which is the general problem in this research area.

We have both given analytical discussion of how average aggregation is computed and why it should converge to the actual average, as well as simulation results that support the analytical discussion (Chap. 3). Moreover, we have proposed a practical convergence rule which uses no global information. Therefore, peers may initiate the algorithm immediately without pre-computing any global information such as network size and topology information. Furthermore, we have compared different threshold mechanisms and their effects on different statistical measures of sensitivity, specificity, accuracy, positive predictive value, and negative predictive value. We have also proposed two different neighbor selection mechanisms as an alternative to the random neighbor selection mechanism. In one mechanism, neighbor selection probabilities depend on the history of selections. Each time a neighbor is chosen, its probability to be chosen again is increased/decreased. In the other mechanism, neighbors are chosen based on their degrees. The more degree they have, the less probability they are chosen as gossip targets. However, random neighbor selection performed best, which showed the power of randomness in gossiping.

We have developed the simulation model, evaluated the behavior and performance of ProFID through extensive large-scale distributed scenarios (up to 30,000 peers) in PeerSim (Chap. 4). First, we have studied the atomic pairwise averaging function in terms of its efficiency in approximating averages of items. Then, we have evaluated the ProFID protocol by considering several metrics such as accuracy, convergence speed, and message complexity. We have also analyzed the effects of algorithm parameters such as epsilon, convergence limit, fan-out, threshold (T), maximum message size (mms). The results confirm the practical nature, ease of deployment and efficiency our approach.

We have compared the scalability and accuracy of ProFID with the well-known Push-Sum protocol [23] by adapting it to the FID problem and practical P2P network settings (Chap. 5). ProFID performed better in terms of convergence time and message complexity, as well as accuracy. This might be due to the adaptation problem of Push-Sum since it uses the assumption that all peers are aware of all items in the network. We eliminated this assumption and modified it such that a peer becomes aware of an item whenever it gets a gossip message including that item. This modification has probably reduced the efficiency of the algorithm. However, it is a necessary modification to make it applicable to FID problem in large distributed systems.

We mainly focused on an applicable and practical algorithm to compute frequent item set in unstructured P2P networks by eliminating some unrealistic assumptions. In this way, distributed system applications would make profit from using ProFID.

Future Directions:

Peer churn: Today's P2P networks are very dynamic and churn is inevitable. Hence, we plan to integrate churn into our network environment and analyze the effect of it on accuracy and convergence. In order to make ProFID more realistic and practical, we aim to evaluate it in peer churn scenarios and dynamic environment.

Furthermore, in addition to simulation model and analysis, we aim to conduct network tests of ProFID on the PlanetLab to evaluate its performance in a real network testbed.

Topology effect: The effects overlay topology properties such as vertex degrees, degree distributions and correlations, clustering coefficient and centrality [49] will be analyzed. To do that, simulations of different topologies with different properties will be performed and the results will be compared and contrasted to the previous results obtained in this thesis.

Improved robustness: ProFID's convergence depends on the estimation of ui item which is the network size estimation item. An initiator peer adds an ui item to its state and this item is eventually distributed to the whole system with frequency around *1/N*. Then, value of N is extracted by taking reciprocal of that value and multiplied with the averages of each item to compute their frequencies. Miscalculation of N causes undesirable frequency computations; therefore robustness of ProFID depends on the robust computation of N. In this implementation, early failure of initiator peer (or its neighbors), will result in a misleading computation and we plan to make ProFID more robust against such scenarios. We have two possible approaches to achieve that. First approach is to start ProFID after making sure that ui item is distributed to some stable peers in the system. By this way, removal of a single peer holding the ui item initially is not going to affect the network size estimation result, and hence frequency estimations of items as much as the original approach. Second approach is to run multiple instances of ProFID. In this approach, each instance starts with a different initiator and the result is computed by interpreting the results of all instances such as choosing the instance that gives the median of list composed of ui estimations of each instance.

Improved energy efficiency: Today's P2P networks consist of millions of peers forming an unstructured network, so computation of even a single global value such as network size consumes a considerable energy. Hence, we plan to have studies focusing on the minimization of energy consumption in FID problem. As an initiative, we developed a model for energy cost of each peer in the network. Based on our energy cost model, each peer consumes energy during the following operations: sending and receiving gossip messages and local computations including averaging operation and local state update. We plan to compute the energy consumption of some well-known algorithms in FID problem and try to improve their energy consumption based on our energy cost model. Moreover, we plan to analyze and reduce the energy cost of ProFID. More specifically, we will focus on how to compress gossip messages, use low bandwidth, less local computation and storage.

BIBLIOGRAPHY

[1] Mei Li and Wang Chien Lee, "Identifying frequent items in p2p systems," in *Distributed Computing Systems, 2008. ICDCS '08. The 28th International Conference on*, June 2008, pp. 36–44.

[2] Márk Jelasity and Alberto Montresor, "Epidemic-style proactive aggregation in large overlay networks," in *ICDCS*, 2004, pp. 102–109.

[3] Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *SIGMOD Conference*, June 2006, pp. 289–300.

[4] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry, "Epidemic algorithms for replicated database maintenance," in *PODC*, 1987, pp. 1–12.

[5] Kamel Aouiche, Jérôme Darmont, and Le Gruenwald, "Frequent itemsets mining for database auto-administration," *CoRR*, vol. abs/0809.2687, 2008.

[6] Kevin S. Beyer and Raghu Ramakrishnan, "Bottom-up computation of sparse and iceberg cubes," in *SIGMOD Conference*, 1999, pp. 359–370.

[7] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman, "Computing iceberg queries efficiently," in *VLDB'98, Proc. of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Ashish Gupta, Oded Shmueli, and Jennifer Widom, Eds. 1998, pp. 299–310, Morgan Kaufmann.

[8] Phillip B. Gibbons and Yossi Matias, "Synopsis data structures for massive data sets," in *SODA*, 1999, pp. 909–910.

[9] "The Peersim simulator," `http://peersim.sf.net`.

[10] Srinivas R. Kashyap, Supratim Deb, K. V. M. Naidu, Rajeev Rastogi, and Anand Srinivasan, "Efficient gossip-based aggregate computation," in *Proc. of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, June 2006, pp. 308–317.

[11] Kenneth P. Birman, Mark Hayden, Öznur Özkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky, "Bimodal multicast," *ACM Trans. Comput. Syst.*, vol. 17, no. 2, pp. 41–88, 1999.

[12] Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Petr Kouznetsov, and Anne-Marie Kermarrec, "Lightweight probabilistic broadcast," *ACM Trans. Comput. Syst.*, vol. 21, no. 4, pp. 341–374, 2003.

[13] André Allavena, Alan J. Demers, and John E. Hopcroft, "Correctness of a gossip based membership protocol," in *PODC*, 2005, pp. 292–301.

[14] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, 2007.

[15] François Bonnet, Anne-Marie Kermarrec, and Michel Raynal, "Small-world networks: From theoretical bounds to practical systems," in *OPODIS*, 2007, pp. 372–385.

[16] Márk Jelasity and Özalp Babaoglu, "T-man: Gossip-based overlay topology management," in *Engineering Self-Organising Systems*, 2005, pp. 1–15.

[17] Alberto Montresor, Márk Jelasity, and Özalp Babaoglu, "Chord on demand," in *Peer-to-Peer Computing*, 2005, pp. 87–94.

[18] Robbert Van Renesse, Yaron Minsky, and Mark Hayden, "A gossip-style failure detection service," Tech. Rep., Ithaca, NY, USA, 1998.

[19] Robbert van Renesse, Kenneth P. Birman, and Werner Vogels, “Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining,” *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003.

[20] Meng Zhang, Qian Zhang, Lifeng Sun, and Shiqiang Yang, “Understanding the power of pull-based streaming protocol: Can we do better?,” *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1678–1694, 2007.

[21] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum, “Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming,” in *INFOCOM*, 2005, pp. 2102–2111.

[22] Márk Jelasity, Alberto Montresor, and Özalp Babaoglu, “Gossip-based aggregation in large dynamic networks,” *ACM Trans. Comput. Syst.*, vol. 23, no. 3, pp. 219–252, 2005.

[23] David Kempe, Alin Dobra, and Johannes Gehrke, “Gossip-based computation of aggregate information,” in *Proc. of Symposium on Foundation of Computer Scienece (FOCS)*, Oct. 2003, pp. 482–491.

[24] Indranil Gupta, Robbert van Renesse, and Kenneth P. Birman, “Scalable fault-tolerant aggregation in large process groups,” in *DSN*, 2001, pp. 433–442.

[25] Stephen P. Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah, “Gossip algorithms: design, analysis and applications,” in *INFOCOM*, 2005, pp. 1653–1664.

[26] Jen-Yeu Chen, Gopal Pandurangan, and Dongyan Xu, “Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 987–1000, 2006.

[27] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Boris Koldehofe, Martin Mogensen, Maxime Monod, and Vivien Quma, “Heterogeneous Gossip,” in *Proceedings of the 10th ACM/IFIP/USENIX International Middleware Conference (Middleware)*, 2009.

[28] Ken Birman, "The promise, and limitations, of gossip protocols," *Operating Systems Review*, vol. 41, no. 5, pp. 8–13, 2007.

[29] Mortada Mehyar, Demetri Spanos, John Pongsajapan, Steven H. Low, and Richard M. Murray, "Asynchronous distributed averaging on communication networks," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 512–520, 2007.

[30] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston, "Finding (recently) frequent items in distributed data streams," in *Proc. of International Conference on Data Engineering (ICDE)*, Apr. 2005, pp. 767–778.

[31] Laukik Chitnis, Alin Dobra, and Sanjay Ranka, "Aggregation methods for large-scale sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, no. 2, pp. 1–36, 2008.

[32] Bibudh Lahiri and Srikanta Tirthapura, "Computing frequent elements using gossip," in *SIROCCO*, 2008, pp. 119–130.

[33] Jayadev Misra and David Gries, "Finding repeated elements," *Sci. Comput. Program.*, vol. 2, no. 2, pp. 143–152, 1982.

[34] Gurmeet Singh Manku and Rajeev Motwani, "Approximate frequency counts over data streams," in *VLDB*, 2002, pp. 346–357.

[35] Graham Cormode and Marios Hadjieleftheriou, "Finding the frequent items in streams of data," *Commun. ACM*, vol. 52, no. 10, pp. 97–105, 2009.

[36] Chris Olston, Jing Jiang, and Jennifer Widom, "Adaptive filters for continuous queries over distributed data streams," in *SIGMOD Conference*, 2003, pp. 563–574.

[37] Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh, "Peer counting and sampling in overlay networks: random walk methods," in *PODC*, 2006, pp. 123–132.

[38] Dionysios Kostoulas, Dimitrios Psaltoulis, Indranil Gupta, Ken Birman, and Alan J. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *NCA*, 2005, pp. 41–48.

[39] Erwan Le Merrer, Anne-Marie Kermarrec, and Laurent Massoulié, "Peer to peer size estimation in large and dynamic networks: A comparative study," in *HPDC*, 2006, pp. 7–17.

[40] Abraham Silberschatz, *Operating System Concepts*, John Wiley & Sons, Inc., New York, NY, USA, 2007.

[41] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, Ian Wakeman, and Dan Chalmers, "The state of peer-to-peer simulators and simulations," *Computer Communication Review*, vol. 37, no. 2, pp. 95–98, 2007.

[42] "Ns-2 network simulator," `http://www.isi.edu/nsnam/ns/`.

[43] Boris Pittel, "On spreading a rumor," *SIAM J. Appl. Math.*, vol. 47, no. 1, pp. 213–223, 1987.

[44] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science (New York, N.Y.)*, vol. 286, no. 5439, pp. 509–512, 1999.

[45] Matei Ripeanu, Adriana Iamnitchi, and Ian T. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, vol. 6, no. 1, pp. 50–57, 2002.

[46] Qin Lv, Sylvia Ratnasamy, and Scott Shenker, "Can heterogeneity make gnutella scalable?," in *IPTPS*, 2002, pp. 94–103.

[47] Zhe Xiang, Qian Zhang, Wenwu Zhu, Zhensheng Zhang, and Ya-Qin Zhang, "Peer-to-peer based multimedia distribution service," *IEEE Transactions on Multimedia*, vol. 6, no. 2, pp. 343–355, 2004.

[48] Daniel Stutzbach, Shanyu Zhao, and Reza Rejaie, "Characterizing files in the modern gnutella network," *Multimedia Syst.*, vol. 13, no. 1, pp. 35–50, 2007.

[49] Maarten van Steen, *Graph Theory and Complex Networks: An Introduction*, Maarten van Steen, VU University, Amsterdam, The Netherlands, 2010.

VITA

EMRAH ÇEM was born in İstanbul, Turkey on May 21, 1985. He received his B.S degree in Computer Engineering from Koç University, İstanbul in 2008. In September 2008, he joined M.Sc. Program in Electrical and Computer Engineering at Koç University as a research and teaching assistant. During his study he worked on P2P networks and gossip protocols. He has co-authored a journal paper in Computer Networks (Elsevier Science), and published papers in conferences ISCIS'10 (25th International Symposium on Computer and Information Sciences, London), and AB'10 (Akademik Bilisim, Mugla). He also has a journal paper submission under review.