Report on Unstructured Network in Peer-to-Peer System

Zhang Zhenjie
Yu Feng
Yang Xiaoyan
{zhangzh2,yufeng,yangxia2}@comp.nus.edu.sg

1 Introduction

In *Structured* peer-to-peer systems, there are several problems hard to tackle. First, loads on different peers are difficult to balance. Since the peers in the system can be heterogenous, it is not fair to distribute equal load to every peer. In *Chord*, the authors argued that one peer with large capacity can occupy several nodes on the ring to relieve the pressure on other peers. In fact, there is still a lower bound on the workload and it is difficult to estimate the number of nodes one large peer should take. Second, the communication cost in *Structured* system can be great. The topology of a *Structured* peer-to-peer system is decided by the global mapping only. No information about the communication condition among those peers are considered. This may lead to the connection of peers who are far away geographically and great latency during information exchange.

Unstructured system in peer-to-peer can overcome the drawbacks in *Structured* system. There is no mandatory topology in *Unstructured* system. Every peer can choose whether to accept or reject the replication by itself and only need to contact with his neighbors in reasonable latency. However, most of the operations on *Unstructured* system are not deterministic. That is why the performances of such systems are not stable. Moreover, message flooding can overflow the whole network when there are too many queries from different peers at the same time.

Generally speaking, most of the current research works on *Unstructured* system are based on some heuristics to reduce the number of messages. In [1], Clark et al. used the Depth First Search technique and proposed several encryption methods in *Unstructured* system. In [5][6], k-random walker was adopted to relieve the problem of message flooding. *Gossip Protocol* is another direction in *Unstructured* network. In [2], Kempe et al. tried to aggregate the information on all the peers by gossip

protocol. In [3], Acuna et al. presented the new information sharing system "PlanetP" which exchange the summarization in Bloom Filter[4].

The rest of the report is organized as follows: Section 2 addresses some basic ideas in Freenet, Section 3 presents GIA and the techniques used in it, Section 4 introduces Gossip Protocol and two corresponding applications. finally, conclusion is given in Section 5.

2 Freenet

Freenet is an adaptive P2P network application that allows users to store, replicate and retrieve data while keeping users' readership and authorship anonymous. It operates on a network of identical nodes which can query one another to store and retrieve data files. Each node maintains its own local data store, which it makes available to other peers for reading and writing, and one dynamic routing table, which is used for routing requests to the most likely physical location of data. There is no flooding or centralized location index in the system. Files are named by location-independent keys and are dynamically replicated in locations near requestors and deleted if they are never requested. The novelty of this system is that it is impossible to discover the true origin of the files in the network and the owner of one node can deny the ownership or knowledge for the physical contents in his node.

In the paper, the author mentioned the design goals of Freenet, which are:

- Anonymity for both producers and consumers of information
- Deniability for storers of information
- Resistance to attempts by third parties to deny access to information
- Efficient dynamic storage and routing of information
- Decentralization of all network functions

In this system, files are named by location-independent keys. The basic model is that requests for keys are passed along from one node to another and each node along the way makes a local decision about where the request should be sent. The routing algorithm will adaptively adjust the routes so as to provide efficient efficient routing while using local, rather than global knowledge. This ensures each node in the system has more privacy. The detailed routing algorithm for data storing and retrieval will be discussed in later section.

2.1 Files keys

Files in Freenet are identified by file keys which are obtained using hash functions. Depending on the purpose and construction methods, the file keys can be categorized to three types: KSK, SSK and CHK.

KSK, which is short for keyword-signed key, is derived from a short descriptive string assigned to the file by the user. This string is firstly used as input to generate a public/private key pair. The public half is hashed to yield the file key. The private half is used to sign the file, providing a means for the user to check the retrieved file genuinely matches its file key. The file is also encrypted using the descriptive string. The reason of this encryption is not for security reasons, as anyone retrieved the file can decrypt it using its description. However, for the owner of the node physically storing this file, all he knows about this file is its file key since its content is encrypted, therefore, he can plausibly deny any knowledge of the contents of his data store for political or legal reasons. KSK is easy to remember and communicate to others since the author of the file only need to publish the descriptive string of the file to allow others to retrieve it. However, one problem of this key is that it forms a flat global namespace, which is undesirable and problematic. For example, two users from different locations can choose the same description for two different files and malicious users can insert junk files under popular descriptions.

To solve the problem of KSK, SSK, short for signed-subspace key, is introduced. The main difference between these two keys is SSK enables personal namespaces. It is achieved by randomly generating a public/private key pair to identify the namespace of one user. The SSK key of one file is generated by hashing the public namespace key and the descriptive string chosen by user for the file independently and XORing them and hashing result. Similar to KSK, the private half of the asymmetric key pair is used to sign the file and the file is also encrypted by the descriptive string.

The third type of key is CHK, short for content-hash key, which is useful for file updating and splitting. A content-hash key is derived simply by hashing the content of the corresponding file. The file is encrpted by a randomly-generated encryption key. The author needs to publish the decryption key along with the content-hash key of the file to let others retrieve it. As before, the decryption key (previously the description sting) is never stored with the file itself, for reasons explained above.

2.2 Retrieving data

To retrieve a file, a user must first obtained or calculated the file key for that file. Then he submits the request to his own node with the key and hops-to-live value. When a node receives a request, it checks its local store first. If a file with the specified file key is found, the node will return the result to the user, declaring itself as the data source. If the file is nt found, the node will find the nearest key in its routing table and forward the request to the corresponding node. If the request is ultimately successful, the node with the file passes the file back using the reverse route. Each node along the route caches the file, adds an new entry in routing table, associating the data source with the requested key. Hence future requests with the same key can be satisfied by the node from local cache directly. Also requests with similar keys will be forwarded to the previous successful data source. For security concern, any node along the way can unilaterally change the reply message to claim itself as the data source. When hops-to-live value reaches 1, the routing can continue with finite probability at each node, to disguise the information about the distance from requester to the actual data source. To conclude, data retrieval in Freenet operates as a depth-first search with backtracking.
There are several advantages for this mechanism:

- Nodes will be specialized in locating sets of similar keys since requests with similar keys will forward to previous successful node.
- Nodes will similarly be specialized in storing clusters of files having similar keys. This is due to local cache once a request is successful.
- Popular data will be transparently replicated and mirrored closer to requestors, which enhances the efficiency of routing.
- With more and more requests processed, connectivity of the whole system increases. For each successful data retrieval, a direct link between the data source and the requester is established.
- Since file keys are obtained using hash functions, lexicographic closeness of the keys does not imply any closeness of original description. In this way, files for the same subject will be scattered across the system, thus avoid single point of failure.

2.3 Storing data

The data storing process is similar to data retrieval. First the author calculates the file key for his file and submits the key to his own node with a hops-to-live value.

Afterwards, the same process as data retrieval is carried out. If a file with the same key is found ultimately, the file insertion causes a collision. The node possessing the file passes the pre-existing file back along the reverse route, behaving as if a request is made. The nodes receiving the file along the way cache it in its local data store and create a routing table entry for the data source. When he inserter receives this file, he knows a key collision has occurred and he then tries using another key for this file. If hops-to-live limit is reached without key collision, a message will be sent to inserter. The user then sends the file to insert, which will be propagated along the path established by the previous query process and stored in each node along the way. Again, any node along the way can unilaterally decide to claim itself as the data source. A new entry associating the data source with the new key will be added in the routing table of each node.
The Advantages of this mechanism are:

- New files will be placed on the nodes with similar keys since the routing algorithm will always route the request to the node with similar keys. This reinforces the clustering effect set up by the data retrieval mechanism.
- New nodes can use inserts as a supplementary means to announce their existence and build up its routing table.
- This mechanism strengthens the resistance of the system to attackers. Attempts by attackers to supplant existing files by inserting junk files under existing keys are likely to spread the real file further.

2.4 Managing data

In Freenet, the owners of a node can configure the amount of storage he is willing to dedicate to his data store. Finite storage capacity poses the problem of data management. In Freenet, node storage is managed as an LRU (least recently used) cache. When a new file causes the data store to exceed to its designated size, the least recently used files will be evicted to make room for the new file. However, the impact of data deletion on availability is alleviated by the fact that routing table entries for those evicted files will remain, enabling the node to get a new copy from the data source in future if it is requested again. Strictly speaking, once all nodes dropped a particular file, it will no longer be available to the network, though the probability of this case is very small. One advantage of this expiration mechanism is that outdated files will fade away naturally if not requested for long therefore no explicit deletion operations are necessary to maintain the system.

The experiments in the paper prove that Freenet is scalable and fault-tolerant with an efficient adaptive routing algorithm. To conclude, Freenet provides an effective means of anonymous information storage and retrieval. It can keep information anonymous and available while remaining highly scalable.

3 Scalability Problem in Unstructured P2P systems

3.1 Introduction

Gnutella, a purely decentralized and unstructured P2P system, is a famous distributed file-sharing application. Its query method is TTL-limited flooding. However, such search mechanism as flooding is inherently unscalable. For example, a single query may generate large amount of messages, which may ultimately contribute to the blocking of the whole network traffic. Using TTL to limit the number of hops of one query message partly address this problem. Since the probability of search success and message overhead on network traffic increase with TTL, there must be a trade-off between search success and network traffic. Further, the load on each node grows with the number of queries and the system size. Nodes, especially high degree ones, are prone to get overloaded.

In order to address the scalability problem of Gnutella-like P2P systems, works have been done to explore alternatives to Gnutella's design. In the following sections, several search methods and replication strategies[5] will be discussed and a new P2P system GIA[6] is introduced.

3.2 Search Methods

In [5], two search methods were proposed and proved to be more scalable than flooding, concerning four different network topologies: Power-Law random graph, normal random graph, Gnutella graph and 2D Grid.

3.2.1 Expanding Ring

The first seach method studied is expanding ring and its basic idea is very simple. Once a node has a new query, it starts a flood with small TTL. A new flood with larger TTL will be invoked only when the last search fails. Such process repeats until the query succeeds. To summarize, expanding ring is multiple floods with increasing TTLs.

Results show that expanding ring improve the network traffic by significantly reducing the message overhead. However, since it still use floods, expanding ring did not address the message duplication problem, which is inherent in flooding. Thus a new method - random walks is proposed.

3.2.2 Random walks

This search method is based on the standard random walk, which, at each step, forwards a query message to a randomly chosen neighbor rather than to all neighbors until search succeeds. However, the disadvantage of this technique is obvious. Although such random walk reduces the message overhead largely, the resultant long delay of one query success is quite unacceptable.

In order to speed up per query processing, a modified version of standard random walk named "multiple-walker" random walks is proposed. Here, a walker is equal to one query message. Instead of just forwarding one message each time, a k multiple-walker random walks forwards k messages at each time. Thus k walkers after N hops is roughly equal to 1 walker after kN hops. "Checking" is adopted as the mechanism to determine when to terminate the walks. To reduce the overhead caused by checking, random walks check with the original query requester every c steps.

A further improvement to random walks is by keeping state. A unique ID is assigned to each query together with all its k walkers. Each node remembers the k neighbors it has forwarded query messages to with a certain ID. Such mechanism guarantees that more nodes will be visited during random walks. However, the message overhead savings achieved by this technique varies among different topologies. Therefore, different P2P network should consider the savings and implementation overhead of state keeping and decide separately whether it is worthwhile or not.

3.2.3 Comparison

Let's compare the two search methods with flooding in Gnutella network topology. As Figure[1] shows, expanding ring and random walks improve network traffic by reducing the average number of messages per node per query and the average number of nodes visited per query significantly. Random walkers achieve the most savings in the Gnutella graph.

On the other hand, the delay of finding one target file in the form of the number of hops per query caused by random walks and expanding ring is quite acceptable as Figure[2] (1) shows.

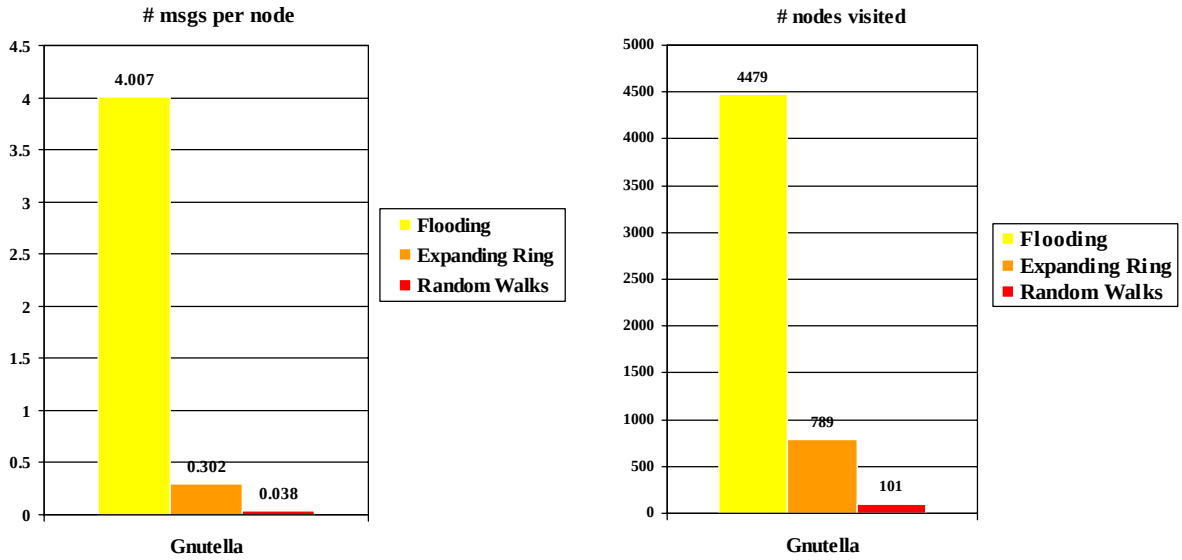

Figure 1: Network traffic comparison in Gnutella network topology.

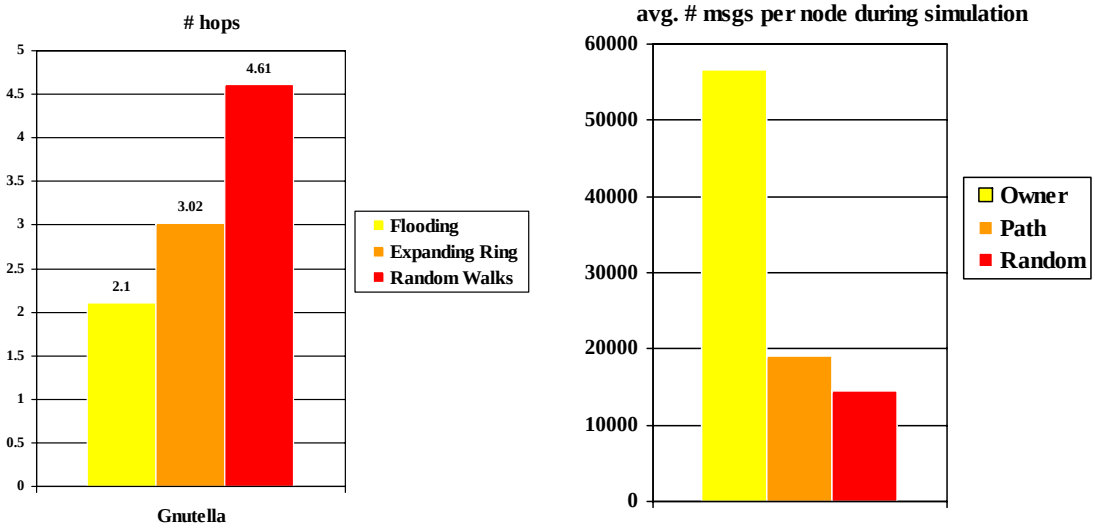

Figure 2: (1).Delay in Gnutella network topology. (2).Message traffic of different replication strategies.

3.3 Replication Strategies

Before discussing different replication strategies, we first introduce three replication distributions studied in [5]: uniform, proportional and square-root.

3.3.1 Replication distribution

Assume that there are m different objects in the P2P system. Each object i is stored in n_i nodes and we have $\sum_{i=1}^{m} n_i = N$. Let q_i denote the relative popularity of object i, in terms of the normalized number of queries issued for object i. Thus we have $\sum_{i=1}^{m} q_i = 1$. The three replication distributions are as follow:

1. Uniform: $n_i = N/m$, which means the number of nodes each object is replicated at is the same.

2. Proportional: $n_i \propto q_i$, which indicate that the replication of i's object is proportional to its popularity.

3. Square-root: $n_i \propto \sqrt{q_i}$, which means that the replication of i's object is proportional to the square root of its popularity.

Both uniform and square-root replication distributions tend to allocate less replicas to popular objects than their "fair-share" and vise versa. But compared with uniform, square-root does so to a less extent. Uniform achieves the highest utilization rate while proportional provides optimal load balancing. Square-root proves to be optimal as it minimizes the average search size thus the overall network traffic overhead.

3.3.2 Replication Strategies

The performance of three different replication strategies in random graph network topology is discussed in [5]. They are: owner replication, path replication and random replication, among which the first two are easy to implement while the last one is harder.

1. Owner replication: the requested object is stored at the requester node only after a successful search.

2. Path replication: the requested object is stored at each node along the path from the requester node to the provider node.

3. Random replication: actually it is not totally 'random'. Let p denote the length of the path (the number of nodes along the path) from the requester to

the provider. Then from the nodes that the k walkers have visited in a search, randomly choose p nodes to replicate this object.

Since square-root replication distribution achieve the minimal search traffic compared with the other tow distribution, we want to find out which replication strategy generate replication ratio most similar to square-root. Both path replication and random replication can. From Figure[2] (2), we can see that path replication and random replication reduces the whole message traffic significantly compared with owner replication due to the benefit of square-root replication distribution that minimize the average search size.

3.4 Discussion One

In [5], among the four network topologies studied, uniformly random graph proves to be better for both searching and data replication, as such topology doesn't generate high degree nodes; while in Gnutella graph, there are some high degree nodes, which may get overloaded easily or cause great message traffic to the whole network. However, such conclusion is generated without considering the situation that different node may have different capacity. In the following sections, we will introduce a P2P system GIA, which takes node heterogeneity into account.

3.5 GIA

GIA is a P2P system that includes several modifications to Gnutella's design. It's design aim is to achieve a higher system capacity ie., higher scalability than Gnutella. Four key components are proposed, which contribute to the improvement of system capacity.

3.5.1 Key Components

- Dynamic Topology Adaptation: the topology adaptation algorithm proposed is to enable higher-capacity node with higher degree ie., more neighbors and high-capacity node are easy to access. Each node has a variable named *level of satisfaction (S)*, whose value is between 0 and 1. S is a function of the capacity and degree of this node and its neighbors. $S = 0$ means that the node is dissatisfied and has no neighbors; $S = 1$ means that the node is fully satisfied and has enough neighbors. The topology adaptation algorithm will continue to seek suitable neighbors to improve S unless $S = 1$.

- Active Flow Control: flow control is used to avoid overloaded nodes. Token is introduced to indicate kind of query a node is willing to process. The

total amount of tokens a node allocates to its neighbors is proportional to its capacity of processing queries. Further, tokens of one node are not evenly distributed among all neighbors but according to neighbors' capacity. This implies that a high capacity node will probably handle more queries and at the same time can forward more queries to other nodes.

- One-hop Replication: as high-capacity nodes are high degree nodes after topology adaptation, if we want them to handle high query rate, they should be guaranteed to have more answers to different queries. One-hop replication means that each node stores information about files at its neighbors. Since high-capacity implies high degree thus more information, those high-capacity nodes are able to handle probably more queries. Moreover, active replication, letting higher capacity nodes to store popular files at their overloaded low capacity neighbors, improves the total capacity of file sharing.

- Search Protocol: the search protocol of GIA is based on random walks [5]. However, instead of forwarding queries randomly, queries are guided to high capacity nodes which probably have more answers. GIA uses biased random walk and each time a node forward its query to the neighbor with highest capacity.

3.5.2 Performance Analysis

Simulations show that GIA achieves 3 to 5 magnitude improvement in total system capacity. Such improvement cannot be obtained by any of the four components but a result of the combination of the four Table 1.

Algorithm	Collapse Point	Hop-Count
GIA	7	15.0
GIA - One-hop Replication	0.004	8570
GIA - Biased Random Walk	6	24.0
GIA - Topology Adaptation	0.2	133.7
GIA - Flow Control	2	15.1

Table 1: Factor analysis for GIA. Collapse point and hop-count of GIA are measured with each of the four key components removed. Here, collapse point means that the query rate per node that the system can sustain.

3.6 Discussion Two

The main contribution of GIA is that it took node capacity into account. By topology adaptation, it dynamically adjust the whole network topology to as to make high-capacity nodes have high degree and easy to access;while at the same time it guarantee high-capacity nodes have more answers to queries. These contribute to the large improvement in capacity of GIA.

4 Gossip Protocol on Peer-to-Peer

In the previous sections, all of the methods are based on the dynamic topology of the network in the system. To find the information in the peer-to-peer system, the message should be spread from the source peer to other peers according to the some searching conditions. *Gossip Protocol* is a quite different method in peer-to-peer environment. First, any operation in the system needs the involvement of all the peers. Second, no peer will send the information to the source peer directly. All the information are spread in the network by "rumor".

Generally speaking, a system supporting *Gossip Protocol* should satisfies the following assumptions:

1. In Gossip Protocol, it is assumed that every peer knows the route to any other peer in the network;
2. the communication cost between any pair of peers is reasonable.
3. All the queries are continuous queries, or the information needed by the queries is continuous;

The first assumption guarantees that one peer can contact any other peer with the same probability. So, there will never be any bottleneck in the whole system which block the spread of information from one part to another. The second assumption assures that the communication between any peers can be done quickly. There does not exist unendurable delay of communication. The third assumption is very important. Since in *gossip protocol* any operation involves all the peer, if all the queries are independent and instantaneous, the peers will have to spread the information needed by different queries at the same time. This will increase the burden of peers greatly and block the whole network with communications.

The framework of *Gossip Protocol* is very simple. Every time a peer can choose one peer in his route table to spread the information it takes. At the same time, it

will summarize the information it gets from other peers. Just like "rumor" in human society, the information from any corner of the system can cover all the peers at a very short time.

In this report, two application scenarios will be introduced in this section: *aggregation and information searching*. As mentioned above, these two kinds of operations must satisfy the third assumption. For aggregation, it is the overall statistics of information from all peers which is continuous in nature. For information searching, we can present the information of one peer by some structure and spread the relatively stable structure across the system.

The philosophy of *Gossip Protocol* is that "You never try to find what you want, just wait for their arrival and check whether it is enough for you."

4.1 Aggregation on Gossip-Based System

There are three types of aggregations on numerical data: *Linear, Linear-Computable, Non-Linear*. Linear aggregation can be calculated by some linear numerical operation on partial aggregations. For example, $Sum(X) = \sum Sum(X_i)$, if $X = \bigcup X_i$ and $\forall i, j,\ X_i \bigcap X_j = \phi$. Linear-Computable aggregation can be represented by some simple numerical operation on several Linear operations. For example, $Avg(X) = Sum(X)/Count(X)$, where $Sum(X)$ and $Count(X)$ are all Linear aggregation. Non-Linear aggregation can not be represented by any Linear aggregation, like $Median(X)$ or $Quantile(X)$.

In [2], Kemp et al. adopt the gossip protocol to the aggregation problem in peer-to-peer environment. Their method can deal with all Linear, Linear-Computable aggregations and some of the Non-Linear aggregations.

4.1.1 Problem Definition

In Peer-to-Peer system S, every peer i maintains a part of the data X, namely X_i. Assume $Op(X)$ is an aggregation on the whole data set X. Then, the computation of $Op(X)$ in S should be conducted efficiently and effectively. The result is available for every peer i in S.

4.1.2 Gossip Protocol for Linear and Linear-Computable

Obviously, if the Linear aggregations can be computed, Linear-Computable aggregation can be done just by simple operation on those Linear aggregations. So, the Gossip Protocol will be given on the example of $Avg(X)$. In the following part, x_i

Algorithm 1 Protocol Push-Sum

1: Let $(s_r, w_r$ be all pairs sent to peer i in round $t-1$
2: Let $s_{t,i} := \sum_r s_r$,$w_{t,i} := \sum_r w_r$
3: Choose a target $f_t(i)$ uniformly at random
4: Send the pair $(\frac{s_{t,i}}{2}, \frac{w_{t,i}}{2})$ to $f_t(i)$ and itself
5: $\frac{s_{t,i}}{w_{t,i}}$ is the estimate of the average in step t

will denote the sum on one peer i.

At all time t, each peer i maintains a *sum* $s_{t,i}$, which is initialized to $s_{0,i} := x_i$, and a weight $w_{t,i}$, which is initialized to $w_{0,i} := 1$.

At time 0, every peer sends the pair $(s_{0,i}, w_{0,i})$ to itself. After time 0, all the communication will follow the **Protocol Push-Sum** in *Algorithm 1.*

During the process of spreading, every peer will distribute what it get to other peers at every round. There is no loss to the sum s and weight w, so $\sum_i s_{t,i} = \sum_i x_i$ and $\sum_i w_{t,i} = n$ are invariant. When the round number increases, the sum of s and w will be distributed evenly to all peers. This gives an intuition to the correctness of the algorithm.

4.1.3 Diffusion Speed

Diffusion speed is the number of round after which the information in the network can be distributed to all the peers evenly. Some stochastic process techniques are used int the proof of the following conclusions.

If the choice of the peer to spread $s_{i,t}$ and $w_{i,t}$ is uniform distribution, we call this protocol is *Uniform Gossip.* In stochastic process, it can be represented by a evenly distributed transition matrix T. In the square matrix T, every entry is $\frac{1}{n}$, where n is the number of peers.

Theorem 1: The diffusion speed of Unifrom Gossip is $T_U(\delta, n, \epsilon) = O(\log n + \log \frac{1}{\epsilon} + \log \frac{1}{\delta})$. Thus, with probability at least $1-\delta$, there is a time $t = O(\log n + \log \frac{1}{\epsilon} + \log \frac{1}{\delta}$ such that $\max |\frac{s_{t,i}}{w_{t,i}} - A(X)| < \epsilon \cdot A(X)$, where $A(X)$ is the true average of the dataset X.
Proof: please refer to [2].

The theorem above shows that the averages on the all peers can converge to the true value in a very fast speed. Moreover, to improve the probability of accuracy

δ, ϵ, more time of diffusion is necessary.

Sometimes, there may be some peers failed or some messages blocked in the network. These factors can affect the performance of the system. The following theorem proves that the effect is very limited.

Theorem 2: If $\mu < 1$ si an upper bound on the probability of message loss in each round resp. the fraction of failed peers, then the diffusion speed T' in the presence of failures satisfies $T'(\delta, n, \epsilon) \leq \frac{2}{(1-\mu^2)} T(\delta, n, \epsilon)$.
Proof: please refer to [2].

If the failure probability μ can be controlled in a small range, the increase of diffusion speed will not be very large.

4.1.4 Practical Consideration

To exploit the full potential of the gossip protocol, several practical problems need to be considered carefully: *Synchronization, Termination Condition and Update in aggregation.*

Algorithm 1 is presented in terms of synchronous rounds, and with a synchronized point. The latter synchronization condition is certainly unnecessary. One peer can start a new aggregation query with an unique identifier Q, and use the protocol to spread the start signal to other peers. The synchronous rounds assumption is also not truly necessary for the definition of the protocols. Every peer in the system can follow its own clocks in deciding when to forward a share of its s and w.

If the number of peers in the system is known to every peer, the bound of the round number is easy to compute. However, in real application, with the frequent logon and logoff of these peers, it is nearly impossible to know the stable number of the peers. In [2], the authors didn't give a definite solution to this problem.

There may be some updates to the data during the aggregation process. If a node i increment the value x_i by Δ_i, it can add the Δ_i to $s_{t,i}$ at any time t. The sum of the $s_{t,i}$ is updated and the increment will soon be distributed to other peers.

4.1.5 Protocol for Quantile Aggregation

Assume that each peer i holds a multiset M_i of m_i elements, and let $M = \bigcup_i M_i$ be the union of all these multisets, writing $m = |M| = \sum_i M_i$. In order to draw a

Algorithm 2 Protocol Push-Random

1: Let (q_r, w_r) be all pairs sent to i in round $t-1$
2: Let $w_{t,i} := \sum_r w_r$
3: Choose $q_{t,i}$ at random from q_r with probability $\frac{w_r}{w_t,i}$
4: Choose shares $\alpha_{t,i,j}$ for each j
5: Send $(q_{t,i}, \alpha_{t,i,j} \cdot w_{t,i})$ to each j
6: $q_{t,i}$ is the random element at time t

random sample from M, each node i first samples an element $q_{0,i}$ from M_i uniformly at random, and then sends the pair $(q_{0,i}, m_i)$ to itself. Subsequently, each node executes the protocol presented in *Algorithm 2*.

After several rounds of exchange, every peer will take some elements $q_{t,i}$. The following theorem characterized the convergence behavior of the protocol.

Theorem 3: Let T be the diffusion speed of the underlying communication mechansim. Then, with probability at least $1-\delta$, after $T(\delta, n, \frac{\epsilon}{(2+\epsilon)n})$rounds, the element at each peer i will be ϵ-close to uniform, i.e. each element is selected with probability between $\frac{1-\epsilon}{m}$ and $\frac{1+\epsilon}{m}$.
Proof: Please refer to [2].

With the theorem above, we can get a sketch of all the elements on all the pairs. Then, the quantile can be drawn from the sampled elements on all pairs respectively.

4.2 PlanetP: Gossip-Based Information Sharing System

Information sharing on unstructured peer-to-peer system often has to face several severe problems. First, in unstructured network, no peer can guarantee it can find what exists in the system deterministically. Second, the performance of the system can be very poor if flooding policy is adopted. Third, these systems never support semantic searching service but only name matching searching.

PlanetP is a peer-to-peer (P2P), semantically indexed storage layer [3]. It has the following characteristics:

1. PlanetP's unit of storage are snippets of eXtensible Markup Language (XML);
2. PlanetP provides personal search service. It can do content searching rather than title searching;

3. PlanetP is designed from the ground up as a P2P system;

4. Gossip Protocol is used in the communication

4.2.1 System Framework

In contract with other information sharing P2P system, the most important difference in PlanetP is that peers in PlanetP never send their queries to other peers. Every peer keeps receiving information representation from other peers and try to maintain the newest ones. Once a query is issued, the peer only needs to search in local representations and send request to those peers who potentially have the information it wants.

Bloom Filter is used to represent all the information shared by one peer. Assuming BF_i is the bloom filter of peer i. Next subsection will give the detail of the structure.

Once a peer logons to the system or has some new shared information, it will publish its new bloom filter BF_i to all its neighbor, and try to get all the bloom filters stored by its neighbors.

At a definite interval, neighbor peers will contact each other to check whether the other one takes any new bloom filter and exchange them if necessary. According to the theorem proved in the last subsection, any new bloom filter can be spread across the network quickly.

If the update of bloom filters are not very frequent, there is no need to check the coherence of neighbor's bloom filters. So, if there is no update from one neighbor for several times, it does not matter to prolong the interval between these two peers.

PlanetP support continuous queries. The queries are kept by the peer for a long time. Once updated bloom filters are available, it will search those again for previous queries. New results may be found from them.

The advantage of PlanetP is that it places very little load on the community for searches against the bulk of slowly changing data. However, if there are many new or updated information in the system, they may be spread relatively slowly.

4.2.2 Bloom Filter

Bloom Filter is an efficient structure for information compression [4].

Assume there are domain D and binary hash space S, where $|S| \ll |D|$. The purpose of bloom filter is compress the subset $d \subseteq D$ into S. Define a cluster of independent hash functions H, every of which is a hash function $f_i : D- > S$. For every $n \in d$, $S_n = \bigcup_i f_i(n)$, set all elements in S_n as 1.

Given an element $n' \in D$, if $\forall s$,$\forall i$,$s \in f_i(n')$, s is definitely 1, then we assert $n' \in d$. Obviously, the result is false negative free which means if we assert $n' \notin d$, it must be true. However, there may be some false positive cases in which $n' \notin d$, but we divide it into d.

In PlanetP, every peer publishes the bloom filter which contains the compression of the information it shares. For some other peer, checking the bloom filter is enough to obtain the sketch of the information. As mentioned above, bloom filter may have some false positive error. So, even if one bloom filter reflect the existence of some information, it may not exist truly.

On the other hand, in [3], the authors did not give any instruction about how to define the size of the hashing space. In fact, it is a crucial part in the success of a real system. Large bloom filters may overflow the capacity of the network. Small bloom filters may have more false positive errors.

4.2.3 Information Brokage

To promote the performance of the whole system, PlanetP also support the DHT (Distributed Hashing Table) services. Each peer joining a PlanetP system can choose to support the *Information Brokage* service or not. This allows other peers without sufficient computing, storage, or communication resources to avoid hosting the service.

This service works as follows. Information is published by one peer will be mapped to an identifier in a specific domain. Some architecture like Chord is used to organized all the peers supporting the service.

To search information in the network, one peer can try to map what it wants to the domain and contact the information storage peer in Chord Protocol.

4.3 Limitation of Gossip Protocol

The advantage of Gossip Protocol is that it does need to maintain the topology of the whole system. However, there are several limitations of Gossip Protocol.

First, Gossip Protocol assumes that one peer can know all the other peers by itself. Under this assumption, the route table for one peer in a large system may be too large to store.

Second, in uniform gossip, one peer can contact any peer in the system with equal probability. In real application, many peer may be very far geographically. The cost of communication between them can be too large to afford.

Third, most gossip-based peer-to-peer system prefer slow updated peers. Since all peers need to exchange asymmetrical information pairwise, if there are many updates, the performance can not be guaranteed.

5 Conclusion

In this report, we introduce several problems and corresponding solution in *Unstructured network* in peer-to-peer network.

The main drawback of *Unstructured* system is message flooding and non-deterministic results. These problems limit the scalability of *Unstructured network*. Freenet is a system using depth first searching which can partly reduce total number of messages in search operation. k-random walker is a good message-saving technique but may lead to slow response. GIA is a high-performance peer-to-peer dynamic adaptation system with k-random walker, dynamic adaptation and congestion control. Gossip Protocol shows good bound in theoretical analysis but can not be adopted in general-purpose unstructured system.

The key problem in peer-to-peer system is how to balance the scalability of unstructured network and stable performance of structured network. This may be the direction of future research work in this field.

References

[1] I. Clarke, O. Sandberg, B. Wiley and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Workshop on Design*

Issues in Anonymity and Unobservability 2000: 46-66

[2] D. Kempe, A. Dobra and J. Gehrke. Gossip-based Computation of Aggregation Information. *Symposium on Foundations of Computer Science* FOCS'03.

[3] F. M. Cuenca-Acuna, C. Peery, R. P. Martin and T. D. Nguyen. PlanetP: Infrastructure Support for P2P Information Sharing. *12th International Symposium on High-Performance Distributed Computing* HPDC03

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422-426, 1970

[5] Q. Lv, P. Cao, E. Cohen, K. Li and S. Shenker. Search and replication in unstructured peer-to-peer networks. *Proceedings of the 2002 International Conference on Supercomputing* ICS 2002: 84-95

[6] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham and S. Shenker: Making gnutella-like P2P systems scalable. *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*: 407-418