Gossip-based Peer Sampling in Social Overlays

MANSOUR KHELGHATDOUST

Master’s Degree Project
Stockholm, Sweden February 2014

XR-EE-LCN 2014:001
Abstract

Performance of many P2P systems depends on the ability to construct a random overlay network among the nodes. Current state-of-the-art techniques for constructing random overlays have an implicit requirement that any two nodes in the system should always be able to communicate and establish a link between them. However, this is not the case in some of the environments where distributed systems are required to be deployed, e.g., Decentralized Online Social Networks, Wireless networks, or networks with limited connectivity because of NATs/firewalls, etc. In such restricted networks, every node is able to communicate with only a predefined set of nodes and thus, the existing solutions for constructing random overlays are not applicable. In this thesis we propose a gossip based peer sampling service capable of running on top of such restricted networks and producing an on-the-fly random overlay. The service provides every participating node with a set of uniform random nodes from the network, as well as efficient routing paths for reaching those nodes via the restricted network. We perform extensive experiments on four real-world networks and show that the resulting overlays rapidly converge to random overlays. The results also exhibit that the constructed random overlays have self healing behaviour under churn and catastrophic failures.
Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Dr. Sarunas Girdzijauskas for continuous support during my thesis research, his patience, motivation, enthusiasm, and immense knowledge. I would like to express especial thank and love to my beloved mom and dad, you are all the reasons for me. Last but not the least, I would like to thank my wife, your patience and love always encourage me in this way.
## Contents

List of Figures ........................................................................................................... 2

1 Introduction ............................................................................................................. 3

2 Related Work .......................................................................................................... 7
   2.1 Gossip approach ................................................................................................. 8
   2.2 Random Walk approach ..................................................................................... 8
      2.2.1 Simple/Classic Random Walk ................................................................... 9
      2.2.2 Metropolis-Hasting Random Walk (MHRW) ........................................... 9
      2.2.3 Re-Weighted Random Walk (RWRW) ..................................................... 10
      2.2.4 Random Walk Membership service ......................................................... 11

3 Solution ................................................................................................................... 13
   3.1 System Model .................................................................................................... 14
   3.2 Solution ............................................................................................................. 14
      3.2.1 Path Construction ....................................................................................... 15
      3.2.2 The Basic Protocol ...................................................................................... 17
      3.2.3 Enhancing Randomness ............................................................................ 18
      3.2.4 Handling Churn .......................................................................................... 19
      3.2.5 The Enhanced Protocol ............................................................................. 19
      3.2.6 API ............................................................................................................ 20

4 Experiments ............................................................................................................. 24
   4.1 Data Set ............................................................................................................. 25
   4.2 Clustering Coefficient(Effect of parameters $\alpha$ and $\beta$) ................................. 25
   4.3 Average Path Length ......................................................................................... 26
   4.4 Degree Distribution ........................................................................................... 26
   4.5 Catastrophic Failure ......................................................................................... 26
   4.6 Churn ................................................................................................................. 31

5 Conclusion ............................................................................................................... 32

References .................................................................................................................. 33
List of Figures

1.1 The main idea is building an on-the-fly random overlay on top of social overlay. Neighbors in random overlay are connected via a route on social overlay. For example, a connection from A to F in random overlay is a route that passes D,E to reach to F on social overlay. .................................................. 5


4.1 X-axis shows cycle and Y-axis demonstrates average clustering coefficient. The effect of parameters $\alpha$ and $\beta$ on global randomness. The diagrams gray ($\alpha=d, \beta=d$) and blue ($\alpha=2d, \beta=2d$) converge correctly. No delay scenarios (yellow and green) do not converge to random overlay. It shows the importance of delay mechanism. .................................................. 27

4.2 X-axis shows cycle and Y-axis demonstrates average path length. Blue diagram represents the average shortest path length of all entries on social overlay. Green diagram denotes the average path length of all entries on on-the-fly random overlay ........... 28

4.3 X-axis shows in degree and Y-axis demonstrates number of nodes. The diagrams represent In degree distribution ................................... 29

4.4 Percentage of unreachable paths after catastrophic failure ...... 30

4.5 X-axis shows cycle and Y-axis demonstrates percentage of reachable paths. The diagrams represent percentage of reachable paths under churn .................................................. 31
Chapter 1

Introduction
Gossip based protocols are popular for communication in large-scale distributed systems, for they are efficient, robust and cost-effective [7]. In many gossip-based protocols, each node continuously exchanges some information with a subset of nodes that it selects randomly from the network. In general, the quality of random node selection has a significant impact on the effectiveness of such protocols. Therefore, at the heart of such gossip based protocols, lies a Peer Sampling Service (PSS), which provides each node with a continuously changing uniform sample of the existing nodes in the network. In generic gossip based peer sampling protocols, each node knows a small, continuously changing set of other participating nodes in the network, called its neighbors or cache. Each node occasionally contacts a random one from its cache and exchange some of its neighbors. After a while, the overlay network converges to a nearly random overlay and keeps it connected by gossiping continuously [7]. Equivalently, a PSS can be described by a random overlay with direct links, where a link indicates that the source node has a pointer to the destination node in its sample.

The underlying assumption beneath the existing gossip based peer sampling services, such as [21], [6], and [3] is that a node can directly contact any other node from its sampled set for information exchange. There are, however, some applications that do not allow unrestricted communication between nodes. Consider decentralized online social networks (DOSNs) that are based on an overlay network, called social overlay, that reflect the social relationships among users. The communications in such networks are limited to immediate friends only, mainly for privacy reasons. In such restricted networks, the connectivity graph is predefined and the communication is only enabled/allowed between nodes that are directly connected. Implementing a PSS on a restricted network is not straightforward, because unlike a random overlay, we cannot add or remove links arbitrarily.

In this thesis, we propose gossip based peer sampling service on restricted overlays. In our solution, an on-the-fly random overlay is continuously being constructed on top of restricted overlay to ensure that every node is provided with a set of uniform random samples. The neighboring nodes in on-the-fly overlay are not able to communicate together directly. Instead, they communicate through the existing restricted network links. Therefore, each node not only maintains a set of sampled nodes, but also a path via available links indicating the route towards them. For example, in Figure 1.1 node A has two entries in its sampled set, which are C and F. A is directly connected to C in the restricted network, since the route from A to C only includes C. However, A is not directly connected to F, and it can only reach F if it traverses through other nodes, e.g., D, E and then F. Suppose in Figure 1.1, node C selects node A as one of its neighbors in on-the-fly overlay to exchange node F with it that is reachable through path [E, F]. Clearly, when node A has received path [E, F], it is not able to reach to F via the received path. The first contribution of this work is a novel algorithm for solving this problem by reconstructing paths to make them usable for the recipient node. Clearly, exchanging paths among nodes only enable them to reach each other via restricted network links and does not converge to a random overlay. The second contribution is a complete framework for constructing an on-the-fly random overlay on restricted networks which is also efficient under churn. To this end, the protocol is annotated with some control variables and mechanisms. Since each gossiping round is proportional
The main idea is building an on-the-fly random overlay on top of social overlay. Neighbors in random overlay are connected via a route on social overlay. For example, a connection from A to F in random overlay is a route that passes D, E to reach F on social overlay.

Figure 1.1: The main idea is building an on-the-fly random overlay on top of social overlay. Neighbors in random overlay are connected via a route on social overlay. For example, a connection from A to F in random overlay is a route that passes D, E to reach F on social overlay.

to path length, it causes a set of nearer nodes perform gossiping more among themselves than with other nodes which leads to creating clusters in some parts of the network. Accordingly, the resulting overlay doesn’t converge to random overlay. We define a delay mechanism to synchronize the length of gossiping rounds so that it eliminates negative impact of having various path lengths. In order to improve the randomness of the on-the-fly overlay and balance In-degree distribution of nodes that results in removing sampling bias, a variable called Swapped is used. Also, we address handling churn in this protocol. First, we handle dead target nodes by defining a variable for each entry that is called age to collect cache entries with dead target node from the network. Then, we propose a solution for handling unreachable paths and isolated nodes.

Although this protocol can be used as sampling service for any gossip protocol, the main target is particularly for push based gossip protocols such as [16]. In push based method, nodes periodically send their state to random selected nodes and after a while they converge to the desired goal. Clearly, unlike push-pull and pull mechanisms, in push based, nodes update their state only if they have been selected as sample by any participating node to receive information. Currently, to execute such gossip protocols on restricted networks, every node periodically sends its state either to one of its neighbors randomly or through an unbiased random walker to a farther node. In the first case, there is sampling bias towards higher degree nodes. In the second case, unbiased random walkers such as Metropolis-Hasting provide single random sample. It means that there is no correlation between random walks initiated by different nodes (global randomness). As a result, a fraction of nodes may never be selected as sample (In particular, low degree nodes with high degree neighbors) and thus cannot update their state at all and some nodes will be selected many times. Thus, both of them are impractical.

To evaluate the algorithm, we use multiple data sets of real world social networks [12],[8],[19] and one collaboration network[9]. We illustrate that the resulting overlay converges to uniform random overlay. The experiments demonstrate that the average length of the paths kept by each node is very close to
the average shortest path length of social overlay. We show that the algorithm provides an unbiased sampling service by investigating in-degree distribution of nodes in random overlay. Thorough experiments are operated for evaluating the algorithm under churn and catastrophic failures.

The remaining parts of this report are organized as follows: related work is described in chapter 2, chapter 3 explains system model and the core protocol including gossip protocol in detail and path construction method. In chapter 4, evaluation of executing algorithm for the four real world graphs is illustrated. The conclusion of the work is represented in chapter 5.
Chapter 2

Related Work
2.1 Gossip approach

To the best of my knowledge, the proposed algorithm is the first gossip based peer sampling service capable of running on restricted overlays. Gossip based peer sampling has been widely studied in the area of unrestricted overlay networks.

In [7], a generic framework for gossip based peer sampling usable for unrestricted networks is proposed. In short, every node (1) maintains a relatively small local membership table that provides a partial view on the complete set of nodes and (2) periodically refreshes the table using a gossiping procedure. The framework is generic and can be used to instantiate known and novel gossip-based membership implementations. In fact, this framework captures many possible variants of gossip-based membership dissemination. These variants mainly differ in the way the membership table is updated at a given node after the exchange of tables in a gossip round.

In [21], a gossip based peer sampling protocol for building and maintaining overlays with properties suitable for diverse applications is proposed. More specifically, it introduces inexpensively building and maintaining very large, connected overlay networks that exhibit important properties of random graphs. These properties should be maintained even in highly dynamic environments. In essence, it is an inexpensive membership management, in the sense that any disruptions of the overlay’s properties resulting from joining or leaving nodes are quickly and efficiently corrected. Like previous protocol, it assumes that nodes maintain a small, partial view of the entire network. The main difference with the previous protocol is the view exchange mechanism that is called shuffling. It ensures that connectivity of the overlay is maintained as long as membership does not change. It avoids loss of data during the execution of the protocol. Once the overlay has converged to random overlay, then every node is provided with a set of random nodes addresses in the network. After that, it starts serving any gossip protocol by providing random sample. In both mentioned algorithms, any pair of nodes should always be able to connect together in the network and thus they are executable only on unrestricted networks.

2.2 Random Walk approach

Random walks on graphs are a well-studied topic; see [10] for an excellent survey. They have been used for sampling the World Wide Web (WWW), peer-to-peer networks, and other large graphs. In restricted networks, such as ad hoc networks and social networks, majority of studies are on the basis of random walk approach [2],[23],[22].

Given a graph and a starting point, we select a neighbor of it at random, and move to this neighbor, then we select a neighbor of this point at random, and move to it etc. The (random) sequence of points selected this way is a random walk on the graph. A random walk is a finite Markov chain. In fact, there is not much difference between the theory of random walks on graphs and the theory of finite Markov chains.

In the following, three well known random walk approaches for obtaining a random sample (Simple, Metropolis-Hasting and re-weighted) as well as a random walk based membership service based on maximum degree method are
explained.

### 2.2.1 Simple/Classic Random Walk

Suppose there is a graph $G = (V, E)$ in which $V$ denotes a set of nodes, $E$ is a set of edges and $k_v$ is degree of node $v$. In the classic random walk, the next-hop node $w$ is chosen uniformly at random among the neighbors of the current node $v$. i.e., the probability of moving from $v$ to $w$ is

\[
P_{v,w} = \begin{cases} \frac{1}{k_v} & \text{if } w \text{ is neighbor of } v \\ 0 & \text{otherwise} \end{cases}
\]

If the graph is connected and non-bipartite, the probability of being at the particular node $v$ converges to the stationary distribution $x_v = \frac{k_v}{2|E|}$.

Therefore, a stationary distribution of a simple random walk on a graph is uniform if and only if the graph is regular. Otherwise, classic random walk is inherently biased. This is clearly biased towards high degree nodes; e.g., a node with twice the degree will be visited twice more often. Accordingly, it does not give uniform sample.

### 2.2.2 Metropolis-Hasting Random Walk (MHRW)

As we have seen, in simple random walk there is a sampling bias towards higher degree nodes. There are some well-known random walk techniques that provide unbiased sampling. One of them is Metropolis Hastings algorithm that is a Markov chain Monte Carlo (MCMC) method for obtaining a sequence of random samples from a probability distribution $P(x)$ such as uniform distribution. The Metropolis Hastings algorithm works by generating a sequence of sample values in such a way that, as more and more sample values are produced, the distribution of values more closely approximates the desired distribution, $P(x)$. These sample values are produced iteratively, with the distribution of the next sample being dependent only on the current sample value. Specifically, at each iteration, the algorithm picks a candidate for the next sample value based on the current sample value. Then, with some probability, the candidate is either accepted (in which case the candidate value is used in the next iteration) or rejected (in which case the candidate value is discarded, and current value is reused in the next iteration). The probability of acceptance is determined by comparing the likelihoods of the current and candidate sample values with respect to the desired distribution $P(x)$. In order to obtain a uniform sample in a graph or restricted network using metropolis-hasting algorithm, we sample nodes from the uniform distribution $\frac{1}{|V|}$ in which $V$ is set of nodes in the network. This can be achieved by the following transition probability: ($k_v$ is degree of node $v$)

\[
P_{v,w} = \begin{cases} \min\left(\frac{1}{k_w}, \frac{1}{k_v}\right) & \text{if } w \text{ is neighbor of } v \\ \sum_{v! = y} P_{v,y} & \text{if } w = v \\ 0 & \text{otherwise} \end{cases}
\]
At every iteration, at the current node $v$ we randomly select a neighbor $w$ and move there based on $\min(1, \frac{k_v}{k_w})$. We always accept the move towards a node of smaller degree, and reject some of the moves towards higher degree nodes. This eliminates the bias towards high degree nodes. In other words, sampling bias is being removed in every step before moving to the next step.

Although the proposed random walk technique provides unbiased samples, it suffers from large mixing time. The mixing time captures the notion of the speed of convergence to the stationary distribution. In other words, it indicates the number of steps the random walker needs to reach a uniform sample. Our algorithm in comparison with Metropolis hasting algorithm provides a set of uniform samples that are accessible in smaller number of steps. Additionally, Metropolis hasting are impractical to be used as sampling service for push based gossip protocols because there is no correlation between various random walks initiated by different nodes (global randomness). As a result, some nodes may never be or much fewer other nodes selected as sample and some others may be selected many times and thus the push based gossip protocol cannot converge to the desired state.

2.2.3 Re-Weighted Random Walk (RWRW)

Like Metropolis hasting technique, Re-Weighted random walk also provides uniform samples. As opposed to Metropolis hasting, it corrects degree bias after moving to next step by appropriate re-weighting of measured values. Consider a random walk that has visited $V = v_1, v_2, ..., v_n$ unique nodes. Each node can belong to one of $m$ groups with respect to a property of interest $A$, which might be the degree, network size or any other discrete-valued node property. Let $(A_1, A_2, ..., A_m)$ be all possible values of $A$ and corresponding groups; accordingly $A_i$ is the sum of all $m$ number of $A$ that would be $V$ (all nodes). For example, if the property of interest is the node degree, then $A_i$ contains all nodes $u$ that have degree $k_u = i$. To estimate the probability distribution of $A$, we need to estimate the proportion of nodes with value $A_i$, $i = 1, 2, ..., m$.

$$P(A_i) = \frac{1}{\sum_{u \in A_i} k_u} \frac{1}{\sum_{v \in V} k_v}$$

Compared to our algorithm, RWRW needs larger mixing time to provide uniform samples. Also, the same as MHRW and for the same reason, this approach is impractical to be used as sampling service for push based gossip protocol.

Both MHRW and RWRW performed excellently in practice for obtaining uniform samples. When comparing the two, RWRW is slightly more efficient for the applications such as social networks. This appears to be due to faster mixing in the latter Markov chain, which (unlike the former) does not require large numbers of rejections during the initial sampling process. However, when choosing between the two methods there are additional trade-offs to consider. First,
MHRW yields an asymptotically uniform sample, which requires no additional processing for subsequent analysis. By contrast, WWRW samples are heavily biased towards high degree nodes, and require use of appropriate re-weighting procedures to generate correct results. Ultimately, the choice of WWRW versus MHRW is thus a trade-off between efficiency during the initial sampling process (which often favors WWRW) and simplicity/versatility of use for the resulting data set (which often favors MHRW).

2.2.4 Random Walk Membership service

As mentioned above, the simple random walk on a graph converges to a uniform limit distribution if and only if the graph is regular. Social and ad hoc graphs are typically non-regular, and thus we cannot use the simple random walk directly to obtain uniform sampling of network nodes. In [1], a different random walk, called the Maximum Degree (MD) random walk is used to achieve uniform sampling. Let $G = (V,E)$ be an undirected, connected, and non-bipartite graph, which is not necessarily regular. Suppose we have an upper bound $D$ on $d_{max}$, the maximum degree of $G$. We use $D$ to transform $G$ into a regular graph $G'$. To this end, we add to each node $v$ of $G$ a weighted self loop (i.e., multiple edges from $v$ to itself). The weight of the self loop of $v$ is set to be $D - d_v$. The degrees of all nodes in the resulting graph $G'$ are the same and equal $D$. The Maximum Degree random walk on $G$ is the simple random walk on $G'$. The transition matrix of this random walk is then the following:

$$
P_{v,w} = \begin{cases} 
\frac{1}{D} & \text{if } (v,w) \in E \text{ and } v \neq w \\
0 & \text{if } (v,w) \notin E \\
1 - \sum_{v' \neq w} P'_{v',w} & \text{if } w = v
\end{cases}$$

If $G$ is connected, then $G'$ is connected and non-bipartite, and hence (since $G'$ is undirected, connected and non-bipartite) the MD random walk has a limit distribution. Furthermore, since $G'$ is regular, this distribution is uniform. In [1], a random walk based lightweight membership service for wireless ad hoc networks is proposed. This service provides every node with a set of continuously changing uniform random samples. It uses Maximum degree random walk described above to achieve uniform sampling as well as utilizing reverse random walk sampling for routing. The approach for applying the MD random walk for generating uniform samples is the following. Every node $v$ starts the sampling algorithm using the MD random walk, passing its own id and the random walk’s mixing time as parameters. The last node reached in the random walk notifies $v$ of its id. This id represents a uniformly sampled node from the network. The notification can be done by using the reverse path of the RW. It introduces significant additional communication overhead. To solve this problem, it proposes a reverse sampling technique. That is, instead of informing the source node $v$ about a samples destination node $u$, the destination $u$ is informed about the source $v$. This way, there is no additional routing overhead for notifying the result of the RW to its initiating node. Every node periodically initiates a MD random walk and constructs a view for keeping uniform samples.

The problems of this protocol compared to our protocol are the following. This protocol assumes network size and maximum degree are known by every
node. Therefore, to make it practical in dynamic environments, it needs extra protocols to be executed in parallel to obtain those values. Furthermore, although many steps would be self loop, it requires large mixing time even worse than MHRW. It is suitable for small network (less than 1000 nodes) and also for networks with low in degree standard deviation. It is not practical for large scale networks such as social networks. Additionally, average path length produced by this protocol is large value that is inefficient for running push based gossip protocols. According to the experiments of the paper, for a network with size 800, the average path length is about 40. In contrast, the algorithm proposed in this master thesis, converge to random overlay faster and produces short average path length and is practical for large scale networks.
Chapter 3

Solution
3.1 System Model

Network and Communication Model. Distributed online social network is a dynamic concept meaning that users form new friendship relations or break the existing one. Here, for simplicity, we assume an immutable social network since it doesn’t affect the protocol. Furthermore, users may be online or offline or they may stop and start their nodes. The protocol can be executed only if the node of the user is up. Being offline does not prevent of executing the protocol if the node is up. Therefore, we assume a dynamic network in which nodes can join and leave the network frequently.

Knowledge of the social graph. We assume an undirected graph with un-weighted edges. Every node knows its friends and its friends-of-friends which is a common feature in today’s OSNs. It facilitates users to expand their networks and does not undermine the scalability of the system.

Trust & Recognizing the social overlay. We assume that nodes are able to discover the IP address of their online immediate friends to be able to forward incoming messages and also are aware of their two hop friends. We assume that friends are trusted, non malicious and are willing to act as relays for forwarding messages.

3.2 Solution

Each node is assigned a unique identifier (such as an IP address) and maintains two groups of caches. (Figure 1.1)

- **Social Cache.** The paths towards immediate neighbors (One hop), called friends and towards two hop neighbors, called friends-of-friends extracted from social overlay.

- **Random Cache.** A small, fixed-sized and continuously changing cache of C entries (typically with the size of 10, 20, 50) of paths towards peers that excerpted from random overlay.

Every time a node joins to the network, it fills in random cache with entries of social cache. Although the way of selecting entries of social cache doesn’t affect the convergence property of random overlay, it is recommended to start filling with lowest degree neighbors of node to prevent initial bias toward higher degree nodes. In each gossiping round, nodes exchange S entries (1 ≤ S ≤ C/2), called Swapping cache, between themselves.

Therefore, each node periodically selects an entry (node,path) from its own random cache to exchange a set of entries , called swapping cache, with it. Note, unlike generic gossip based sampling services, the length a gossiping round depends on path length of the selected node. However, paths that are sent by sender node are not usable by recipient node and do not show a path from the receiver node to the received node identifier. For example, In Fig 3.1, node A selects a path towards node N ([D,E,I,N]) from its own cache and carrying [B,C,G,K] as swapping cache. Clearly, node N is not able to reach to node K through path ([B,C,G,K]). Thus, we introduce a path construction algorithm to resolve this problem.
3.2.1 Path Construction

The aim of path construction algorithm is to make paths that are dispatched by sender node operational for recipient node. In order to reduce communication cost, speed up convergence to random overlay, constructing paths with as short length as possible is crucial. Recall, each node knows its friends and two-hop friends. We exploit this common feature of distributed online social networks to construct paths and make them stunted.

The algorithm is straightforward. Source node, destination node and relay nodes (nodes within path) that are involved in a gossiping round execute this algorithm once they have received swapping cache. In this algorithm, if recipient is source node, the current identifier is only added to the first of the resulting path. If destination node or any relay nodes have received a path, reverse it and start traversing node identifiers to build new one. If the visiting identifier belongs to friends or friends-of-friends list of the current node, cuts the path and discards the remaining part and adds friend(s) information to the first of the resulting path otherwise, the visited identifier is simply added to the first of new path. In addition, in relay node, after traversing the path, the current node identifier is also added. (Algorithm 1)

There is a system parameter called $\alpha$, denotes maximum threshold of path length. It should be set to an arbitrary large value preferably to a value close to the diameter of the network so that it ensures that there is at least one acceptable path among any pair of nodes. Although it happens rarely, once destination node has received a path with the length larger than it, simply reject and does not save it in the cache to ensure having short length paths.

For example, In Fig 3.1, node A selects a path towards node N ([D,E,I,N]). For the sake of simplicity, we assume the size of swapping cache is 1, meaning that each gossiping message only piggyback 1 path as swapping cache([B,C,G,K]). Beside the forwarding path of swapping cache, every node cooperates in building reverse path used as route for sending swapping cache of receiver node. The algorithm is straightforward and when every node has received the message, only adds its identifier to the first of the reverse path. A as source node, only add itself as the first element of path and forward message to D(I). Since B is in the friends list of D, it cuts the path and removes A from the path (II). When E has received the path, since C is its friend-of-friend, it cuts the path and remove the remaining parts and build the path [E,F,C,G,K] (III). The resulting path after execution of the algorithm by I is [I,J,K](IV). N as target node executes the algorithm and builds the path [L,J,K]. Note that since N is target node, it doesn’t add itself to the end of path(V). Node N saves the path in its cache and selects a swapping cache to send back towards node A using the constructed reverse path. In Fig 3.1 we only have shown the process from A to N and return scenario follows the same instruction.
Algorithm 1 PATH CONSTRUCTION

Require: any node p has the following methods:
  • isNeighbor(q) : return true if q is friend of p.
  • isTwoHopNeighbor(q) : return true if q is friend-of-friend of p.
  • getNeighbor(q) : return common neighbor between current node and node q.

1: function RECONSTRUCT PATH(path)
2:   resultPath = empty
3:   if self.Id is sourceNode then
4:     resultPath.addFirst(self.Id)
5:     return resultPath
6:   end if
7:   for all id ∈ path.reverse() do
8:     if isNeighbor(id) then
9:       resultPath.addFirst(id)
10:      break
11:     else if isTwoHopNeighbor(id) then
12:       resultPath.addFirst(id)
13:       resultPath.addFirst(getNeighbor(id))
14:      break
15:     else
16:       resultPath.addFirst(id)
17:     end if
18:   end for
19:   if self.Id is relayNode then
20:     resultPath.addFirst(self.Id)
21:   end if
22:   return resultPath
23: end function
3.2.2 The Basic Protocol

The basic algorithm, is a simple peer-to-peer communication model incorporated with path construction algorithm introduced in the previous section. Each node $P$ repeatedly initiates a swapping cache operation, by executing the following seven steps:

1. Select a copy of random subset of $S$ neighbors ($1 \leq S \leq C/2$) from $P$'s own random cache, and a random node $Q$, within this subset to swap cache with.

2. Execute the path construction algorithm for entire entries of the swapping cache.

3. Send the updated cache to the next node of the path towards $Q$.

4. Receive from one of its social neighbors within the reverse path a subset of no more than $S$ of $Q$’s entries.

5. Execute the path construction algorithm for all the received entries.

6. Discard the entries with a path longer than $\alpha$ (Maximum path length).

7. Update $P$’s cache, by firstly replacing the same entries already existed with longer path, (if any), and secondly replacing entries with the ones selected randomly.

On reception of a swapping cache request, node $Q$ randomly selects a copy of subset of its own entries, of size $S$, executes step 2 and sends it to one of its social neighbors in the constructed reverse path, set age field of the received reverse path towards node $P$ to zero and insert it to the received swapping cache, and executes steps 5, 6 and 7 to update its own cache.

Any relay node, involved in gossiping, execute path construction algorithm and cooperate in building reverse path.
3.2.3 Enhancing Randomness

Recall, the aim of this protocol is continuously constructing an on-the-fly random overlay in order to ensure that every node is provided with a set of random samples. Accordingly, it can act as sampling service for any gossip protocol particularly push-based to execute on restricted networks. Clearly, exchanging paths among nodes only enable them to reach each other via restricted network links and does not give a randomness property as good as random overlays. To this end, the protocol is annotated with some control variables and mechanisms.

The first key difference between basic and enhanced versions is that nodes don’t randomly choose which neighbor in random cache to exchange with. Instead, they select the entry that is waiting more than others in cache. To do that, a variable called Waiting Time is added to each entry. It represents time that the entry is waiting to be selected for gossiping in the current cache. Once the entry has moved to another node or has been selected for gossiping, the value is reset. The motivation behind this is improving the quality of on-the-fly overlay in term of randomness.

Note, as opposed to generic gossip protocols, in this protocol, every gossiping round is a proportional to path length. For example, gossiping from node A to F through path [B, C, D, E, F] takes 10 hops while to node G through path [D, G] takes 4 hops. Thus, nodes may have several rounds executing at the same time. In other words, nodes start new gossiping round without receiving any response from the previous round. Additionally, it causes a set of nearer nodes perform gossiping more frequently among themselves than other nodes. As a result, the overlay doesn’t converge to random overlay perfectly.

The second key difference is injecting a delay mechanism. The motivation behind this is to prevent creating clusters in some parts of the overlay that is the result of having gossiping rounds with various lengths. This mechanism resolves the problem by equalizing the length of gossiping rounds. It leverages two system parameters called $\alpha$ (path length maximum threshold) and $\beta$ (maximum delay) that is equivalent to $\alpha$. The solution is postponing a gossiping round with a time $\gamma$ that is obtained from the difference between $\beta$ and the length of the selected path. For example, if $\beta = 10$ hops and the length of selected path is equal to 3 hops, then, round is simply started 7 hops later ($\gamma = 7$).

The third enhancement in comparison with the basic version is about replacement policy of the cache entries. Each node periodically picks a copy of $S$ entries randomly and propagates them in the network. Since each node starts new round without getting any response from previous rounds and also swapping cache selection is done randomly, as a result, some entries might be propagated much more than the others. Furthermore, the replacement policy is also done randomly. Thus, less propagated entries might be replaced while more propagated ones remain in the cache. The first motivation behind this enhancement is improving the randomness of the on-the-fly overlay. The second motivation is balancing In-degree distribution of nodes in on-the-fly overlay that results in removing sampling bias. The third - and far less obvious - motivation is ensuring global randomness. To this end, a variable called Swapped is added to entries. It denotes the number of times that the entry was selected as one of the swapping cache entries in the current node. Whenever an entry is selected as one of the swapping cache members, the swapped variable of that entry is increased by one. Once each node has received a swapping cache, it replaces
them with the existing entries with the highest *Swapped* variable. It means that the entries that sent out (propagated) as swapping cache more, are given higher priority for replacement.

### 3.2.4 Handling Churn

Almost every distributed system has to deal with churn in case of joins, graceful leaves, and failures. Social overlays usually (although not random) have quite good expansion properties [11], thus in practical churn scenarios it is very hard to disconnect them. However, if the failures are correlated, isolated components may appear. In such a case, although the peer sampling service cannot provide a globally uniform sample any more, it will keep working for each of the partitions, independently. The motivation for handling churn is detecting and removing unreachable paths that reduce vain walking and increase performance of the protocol. There are three challenges that the protocol should cover.

- **Handling dead target nodes.** To collect cache entries with dead target node from the network, we define a variable for each entry that is called *age* that represents the freshness of the given cache entry in the network. Each entry keeps this value while moving among nodes in the network. It gathers gradually old paths and replace with newly introduced ones. Therefore, replacement policy is changed such that M entries with highest swapped variable and H entries with highest age variable are replaced with newly received entries (H+M=S). It is a way to fresh the information that enter to the caches.

- **Handling unreachable paths.** In some cases, although the target node is alive, relay nodes might be dead. For example, node A is going to communicate with node D through path [B,C,D]. Suppose node B is dead, since social networks are highly clustered networks, most likely B and C would have a common neighbor such E that act as a bypass node to proceed path. Thus, node A sends the path to node E to continue proceeding.

- **Handling isolated nodes.** There might be nodes that all their entries are unreachable and also no reachable path towards the node in the network exists. Although the node is alive, it becomes isolated. To detect such a scenario, age variable is leveraged. Since entries of an isolated node remain forever in the cache, accordingly, ages of entries increase. If the average age of entries exceeded from a maximum threshold (λ), the node is suspected to be kept apart. To resolve it, simply join again by cleaning the cache and filling it with alive social cache entries.

### 3.2.5 The Enhanced Protocol

In enhanced version of the protocol, nodes initiate swapping cache request periodically, yet not synchronized, at a fixed period δ. In addition to the network address and path, cache entries contain three extra fields called age, waiting time and swapped that were introduced in the previous section. Note, waiting time and swapped variables of each entry are local variables. It means that once each entry has moved to another node, the values of them are set to zero. Thus,
only network address, path and age of entries are moved across the network to reduce network bandwidth consumption.

The enhanced version of the protocol is performed by letting the initiating peer P execute the following ten steps:

1. Increase by one the age and the waiting time of all entries.
2. Select copy of entry Q with the highest waiting time from the cache, and copy of S - 1 other random entries as swapping cache.
3. Increase by one the swapped field of all selected S - 1 entries within cache by one.
4. Set waiting time entry Q within cache to zero.
5. Execute the path construction algorithm for entire entries of the swapping cache.
6. Send the updated cache to the next node of the path towards Q.
7. Receive from one of its social neighbors within the reverse path a subset of no more than S of Q's entries.
8. Execute the path construction algorithm for all the received entries.
9. Discard the entries with a path longer than $\alpha$ (Maximum path length).
10. Update P's cache, by firstly replacing the same entries already existed with longer path, (if any), and secondly replacing M entries with the highest swapped and H with the highest age such that M + H = S. Notice, the replacement is one by one process meaning that a newly added entry might be replaced by the next one. (if apply)

On reception of a swapping cache request, node Q randomly selects a copy of subset of its own entries, of size S, execute step 3 for S entries, executes step 5 and sends it to one of its social neighbors in the constructed reverse path, execute step 8, set age field of the received reverse path towards node P to zero and insert it to the received swapping cache and executes steps 9, 10.

Like in basic version, any relay node involved in gossiping, execute path construction algorithm and cooperate in building reverse path.

3.2.6 API

The peer-sampling service provides a participating node with a random subset of other participating nodes to execute one or more protocols that require random samples. The API of the peer-sampling service simply consists of two methods: init and getPeer. The init() method initializes the service on a given node if this has not been done before. The getPeer() method returns a sample peer along with a path so that it reaches to the sample. Note, the characteristics of this sample including its randomness and correlation in time and with other peers are influenced by the implementation. To increase the local randomness, an implementation of getPeer() method is as follows. Take a queue equal to the cache size and fill in it with the paths exist in the cache. Return the head path
in the queue as sample and move it to the tail of the queue. Once the cache has been updated, synchronize queue entries with cache entries. It prevents returning one entry more often than the others and results in increasing local randomness. In addition, it considerably increases the performance of push based gossip protocols.

**Algorithm 2** ACTIVE THREAD

1. **procedure** JOIN
2. \[\text{InitCache()} \uparrow \text{get neighbors from the bootstrapping process}\]
3. \[\text{do every } \delta\]
4. \[\text{ExchangeCache()}\]
5. **end procedure**

6. **procedure** EXCHANGECACHE
7. \[\text{HandleDelayMessage()} \uparrow \text{decrease Time for messages and move if is zero}\]
8. \[\text{cache.IncreaseAgeAndWaitingTime()} \uparrow \text{Age and Waiting Time of all entries}\]
9. \[q \leftarrow \text{cache.SelectEntry()} \uparrow \text{return copy of oldest local age entry}\]
10. \[q.\text{Age} \leftarrow 0\]
11. \[e \leftarrow \text{cache.SelectRandom(S - 1)} \uparrow \text{return copy of S - 1 entries to swap}\]
12. \[\text{cache.increaseSwapped(e)} \uparrow \text{increase swapped of the chosen entries}\]
13. \[\text{rp.addFirst(self.Id)} \uparrow \text{Build reverse path}\]
14. \[e \leftarrow \text{reconstructPath(e)} \uparrow \text{Algorithm 4}\]
15. \[\text{if } \beta \geq q.\text{path.length} \text{ then}\]
16. \[\gamma \leftarrow \beta - q.\text{path.length} \uparrow \beta \text{ is maximum delay equal to path length threshold } (\alpha)\]
17. \[\text{saveDelayMessage(q,e,rp,}\gamma)\]
18. **else**
19. \[\text{neighbor} \leftarrow q.\text{path.removeFirst()}\]
20. \[\text{Send } \{q,e,rp,\text{request}\} \text{ to neighbor}\]
21. **end if**
22. **end procedure**
Algorithm 3 PASSIVE THREAD

1: on event ≪ q,e,rp,request ≫ \(\triangleright\) if q.path.length == 0 Target
2: re ← cache.SelectRandom(S) \(\triangleright\) return copy of S entries to swap
3: cache.increaseSwapped(re)
4: re ← reconstructPath(re)
5: neighbor ← rp.removeFirst()
6: Send {rp,re,response} to neighbor
7: updateCache(e)

8: on event ≪ q,e,rp,request ≫ \(\triangleright\) if q.path.length >0 Relay
9: e ← reconstructPath(e)
10: rp.addFirst(self.Id)
11: neighbor ← q.removeFirst()
12: Send {q,e,rp,request} to neighbor

13: on event ≪ rp,re,response ≫ \(\triangleright\) if rp.path.length == 0 Target
14: updateCache(re)

15: on event ≪ rp,re,response ≫ \(\triangleright\) if rp.path.length >0 Relay
16: re ← reconstructPath(re)
17: neighbor ← rp.removeFirst()
18: Send {rp,re,response} to neighbor
Algorithm 4 UPDATE CACHE

1: procedure UPDATECACHE(entries)
2:   cache.sortBySwapped()
3:   for all entry ∈ entries do
4:     if entry.path.length > α then
5:       return
6:     end if
7:     if cache.contains(entry) then
8:       if entry.path.size() < cache.get(entry).path.size() then
9:         cache.replace(entry, cache.get(entry))
10:     end if
11:   else
12:     if cache.size() == CACHESIZE then
13:       if m > 0 then > initialized by \( m = M \)
14:         cache.removeFirst()
15:         cache.addLast(entry)
16:         m ← m - 1
17:       else
18:         cache.sortByAge()
19:         cache.removeFirst()
20:         cache.addLast(entry)
21:       end if
22:     break
23:   end if
24:   end if
25: end for
26: end procedure
Chapter 4

Experiments
4.1 Data Set

We implemented the algorithm on PEERSIM [15], a discrete event simulator for building P2P protocols. Since social network graphs are the main target of this algorithm, the protocol is investigated on three data sets of real world social networks including Facebook [19] [12], and Wiki-Vote [8]. The experiments are also executed on collaboration network of Arxiv Astro Physics [9]. The largest connected component of the graphs are extracted for running the experiments. The detail information and properties of the graphs are given in Table 4.1.

Table 4.1: DATASET

| Data Set    | |E| | Type | Reference |
|-------------|-------------|-------------|-------------|
| Facebook I  | 4039        | 88234       | Social      | [12]       |
| Wiki-Vote   | 7066        | 103663      | Social      | [8]        |
| AstroPh     | 17903       | 197031      | Collaboration | [9]      |
| Facebook II | 63391       | 817090      | Social      | [19]       |

4.2 Clustering Coefficient (Effect of parameters \(\alpha\) and \(\beta\))

The clustering coefficient of a node is defined as the number of edges between neighbors of a node divided by the number of all possible edges between those neighbors. The average clustering coefficient of a node is the clustering coefficient averaged across all the nodes in the network. The clustering coefficient of a random network is formulated by division of \(C\) over \(N - 1\) that \(C\) is cache size and \(N\) is network size \(\left(\frac{C}{N-1}\right)\). We are interested in measuring this property so that we make sure that the clustering coefficient of resulting overlay is equivalent to clustering coefficient of random overlay which results in every node is provided with a set of uniform random samples. Note, neighbors of each node in random overlay are target nodes of paths within the cache of that node.

Furthermore, we executed the experiments with setting various combinations of parameters maximum threshold of path length \((\alpha)\) and delay \((\beta)\) to see the impact of them on global randomness of the resulting overlay. Four scenarios are executed for each graph \((\alpha = d, \beta = d ; \alpha = 2d, \beta = 2d ; \alpha = d, \beta = 0 ; \alpha = 2d, \beta = 0)\) where \(d\) denotes the diameter of the network. Considering diameter of the network as value of \(\alpha\) ensures that there is at least one acceptable path by the protocol between any pair of nodes in the network. In other words, it is the worst scenario that might take place for global randomness of the resulting overlay. In dynamic environments, however, it can be set to a value that is large enough. The specification of other parameters for all scenarios are \((C = 20, S = 5, M = 3 \text{ and } H = 2)\).

As Fig 4.1 exhibits, for no delay scenarios \((\beta = 0)\) the resulting overlay never converges to random overlay. It shows the importance of delay mechanism. For the other scenarios, the clustering coefficient of all the graphs converge to random graph which ensures global randomness. Larger value for \(\alpha\) decreases the
speed of convergence and results in having longer paths that increase communication cost. However, it causes better local randomness for nodes.

### 4.3 Average Path Length

The shortest path length between nodes P and Q is the minimum number of links needed to traverse from P to Q. The average path length is the average of the shortest path lengths between any pair of nodes. It is a metric for measuring communication cost. One prominent property of random networks is having low diameter. However, in our case, communication between any two nodes in random overlay is a route over social overlay. Although the path construction algorithm makes paths shorter by utilizing the information of two hop friends and prevents constructing long paths by rejecting them with respect to $\alpha$, it does not find the shortest path between any pair of nodes. Here the motivation is to see how close paths lengths in random graph are to the shortest one in social graph. We calculated the average path length of cache entries of nodes, and then we compared them with the shortest path of those entries over social graph. In figure 4.2, the results for 1000 cycles are depicted. As it exhibits, the average path length of random overlay is very close to the average shortest path length of social overlay. It is observed, although there might be a big difference at the beginning, once the overlay has converged to random overlay, it drops sharply and remains steady during execution.

### 4.4 Degree Distribution

In undirected graph, degree of a node is a number of links it has to other nodes. But in directed graph, like in our case, in-degree and out-degree of graph are distinguished. out-degree is fixed and equal to cache size for every node. Therefore, we focus on in-degree of graph that is the number of links ending at the node. Nodes with higher degrees selected as sample more frequently than lower degree nodes in social overlay. The motivation for evaluating degree distribution of nodes is to see whether this bias is removed in random graph. In other words, how uniformly degrees are distributed. We investigate it by calculating degree distribution of target nodes of paths over random graph. The ideal case is to have a degree distribution with low standard deviation. It ensures an unbiased sampling independent of node degrees in social graph. In Fig 4.3, in-degree distribution for all graphs are depicted. X-axis represents degree of nodes and Y-axis reveals count of nodes with each degree. The experiment ran with $C = 20$, $S = 5$ and the results show 70% of nodes have in degree 20 ± 20%, a value between 16,24, for all graphs. It is the effect of swapped and age parameters.

### 4.5 Catastrophic Failure

In this section, we present simulation results on Catastrophic failure, where a significant portion of the system fails at the same time. We want to investigate the self-healing behavior of the proposed solution (age, bypass nodes and isolated nodes) under massive failure. We are interested to see how fast all unreachable
Figure 4.1: X-axis shows cycle and Y-axis demonstrates average clustering coefficient. The effect of parameters $\alpha$ and $\beta$ on global randomness. The diagrams gray ($\alpha=d, \beta=d$) and blue ($\alpha=2d, \beta=2d$) converge correctly. No delay scenarios (yellow and green) do not converge to random overlay. It shows the importance of delay mechanism.
Figure 4.2: X-axis shows cycle and Y-axis demonstrates average path length. Blue diagram represents the average shortest path length of all entries on social overlay. Green diagram denotes the average path length of all entries on on-the-fly random overlay.
Figure 4.3: X-axis shows in degree and Y-axis demonstrates number of nodes. The diagrams represent In degree distribution.
paths (due to either dead target node or crashed relay node) are gathered from the system. We executed experiments several times for each scenario and calculated confidence interval 95% to obtain more accurate result. For each graph, three scenarios are implemented by randomly failing 30%, 40%, and 50% of nodes at cycle 100. The parameter $C = 20$, $S = 5$, $\alpha$ is equal to diameter of each network and $\beta = \alpha$. Once the average age of a node has exceeded from 50, it is detected as isolated node and join again to the network. The results show that, independent of network size and percentage of failure, almost all the existing paths in the network after 100 cycles are reachable. It represents self-healing behavior of the protocol. We have observed that if the overlay is not partitioned, after a while, the overlay converges to random overlay and in case of splitting to several partitions, each partition with respect to nodes within it, separately converges to random overlay.

Figure 4.4: Percentage of unreachable paths after catastrophic failure
4.6 Churn

In this experiment, we use a scenario with churn, i.e., a scenario in which nodes can join or leave at any time. We use a real-world trace [5], which monitors a set of 4000 nodes participating in the Skype super-peer network for one month beginning September 12, 2005. To make it operational for our graphs, some changes have been made. First, we changed the crash-stop model used by Skype with a crash-recovery model. It means that each user has a unique identifier forever and once a user has left the network, it rejoin to the network with the same identifier. Second, we adapted the churn rate in each time on the basis of the size of the network that is under experiment. The churn scenario is started, and the unreachable paths are calculated after 100 cycles. As Fig. 4.5 demonstrates, for all 4 graphs 80% to 90% of paths of nodes are reachable during the execution of the churn scenario.
Chapter 5

Conclusion
In this thesis, I presented a gossip-based peer sampling service that can be executed on restricted networks such as distributed online social networks that are deployed based on social relationships among users. The solution was building regularly on-the-fly random overlay on top of social or any other restricted overlays. Nodes exchange a set of paths among themselves using a gossip protocol. The path indicates how a node can reach the sample. I proposed a path construction algorithm that makes short length paths by utilizing two hop neighbors information. This algorithm is a fully decentralized algorithm and can be applied in large scale networks. No prior knowledge about the structure of the network graph is being used in this algorithm. Even the diameter of the graph need not to be known and maximum path length variable can be set to a value close to the diameter of the network. Overall, the experiments show that the resulting overlay converges to random overlay. I demonstrated that this protocol has self-healing behavior under churn and catastrophic failures by evaluating reachable paths.
Bibliography


