Ensuring Uniformity in Random Peer Sampling Services

Anne-Marie Kermarrec, Vincent Leroy, Christopher Thraves-Caro

Ensuring Uniformity in Random Peer Sampling Services

Anne-Marie Kermarrec
INRIA Rennes – Bretagne Atlantique
France

Vincent Leroy
INSA Rennes, UEB
France

Christopher Thraves
INRIA Rennes – Bretagne Atlantique
France

***Abstract*—The peer sampling service is a core building block for gossip protocols in peer-to-peer networks. Ideally, a peer sampling service continuously provides each peer with a sample of peers picked uniformly at random in the network. While empirical studies have shown that uniformity was achieved, analysis proposed so far assume strong restrictions on the topology of the overlay network it continuously generates. In this work, we analyze a *Generic Random Peer Sampling Service* (GRPS) that satisfies the desirable properties for any peer sampling service –*small views*, *uniform sample*, *load balancing*, and *independence*– and relieve strong degree connections in the nodes assumed in previous works. The main result we prove is: starting from any simple (without loops and parallel edges) directed graph with out-degree equal to c for all nodes, and recursively applying GRPS, eventually results in a *random* simple directed graph with out-degree equal to c for all nodes. We test empirically convergence time and independence time for GRPS. We use this empirical evaluation to show that GRPS performs better than previously presented peer sampling services. We also present a variant of GRPS that ensures that the in and out-degrees of nodes in the initial network are maintained in the resulting graph. Finally, we discuss on how to deal with new nodes in both settings.**

***Keywords*-Peer sampling service; random networks; P2P networks; random process**

I. Introduction

In the context of peer-to-peer (P2P) networks, gossip-based protocols have emerged as a useful tool for information dissemination [1], aggregation [2], load balancing [3] and networking management [4], among others. In a gossip-based protocol, it is assumed that each peer maintains partial and bounded knowledge of the peers in the network. This is called the *partial view* (each node knows only their neighbors, a bounded number of peers). Periodically, each peer exchanges messages with a peer from its partial view, the exchange can be done by pushing or pulling information, or both, from one peer to the other. This simple communication pattern provides a robust way to disseminate information in large-scale distributed systems. Nevertheless, to comply with the bounds on dissemination information, the partial view of any peer should be a uniform sample of all peers currently in the network. Therefore, ideally, any node exchanges information with peers that are selected following a uniform random sample of all nodes in the system [1], [5], [6], [7]. A Random Peer Sampling (RPS) is the *service that continuously provides each peer with a random sample of the peers in the network*, i.e., an RPS continuously updates the partial view of each node in the network so that it represents a uniform random sample of the peers in the network. Consequently, it is of the utmost importance to prove the uniformity of the sample a peer sampling service provides.

Due to the distributed nature of P2P networks, any RPS has to be a distributed algorithm running in parallel on each peer of the network, relying on a bounded amount of information. An RPS running on each node takes benefit of partial views of nodes in the network to randomly generate a new partial view for the host node. Gurevich and Keidar in [8] described the desirable properties of an RPS: *Small views:* the partial views size must be considerably smaller than the size of the whole network; *Load balancing:* eventually the variance of the node in-degrees is bounded; *Uniform samples:* each node in the system is eventually provided with a sample chosen *uniformly* among all the possible samples; *Independence:* fast independence between the present partial view and the partial views in the past.

In an RPS, each node collects information from other partial views to produce a new partial view. The collection of partial views can be local or deep. A local collection is only provided by the neighbors, i.e., the nodes in the present partial view [7], [9], [8]. On the other hand, a deep collection uses random walks to collect farther partial views [10]. Nevertheless, the topology plays an important role in random walks and the sample generated could be far from a uniform one [11]. Furthermore, dynamism in networks make random walks less effective [12]. In this work, we analyze a *Generic Random Peer Sampling Service* (GRPS) that uses local collection of partial views, and that satisfies the desirable properties for any RPS.

A. Our Contributions

In this paper we study GRPS, a Generic RPS service that, due to its design, keeps the original size of the partial views of each node, and also preserves simplicity of the network (i.e., if there are no repetitions neither self citations in the partial views of the initial network, GRPS preserves that property). GRPS could be seen as the *swapper* particular case presented in [7]. It captures the *swapper* idea of the gossip exchange by minimizing the loss of information between the two nodes involved in the exchange.

First, we prove its atomicity, i.e., even if GRPS is running in parallel among the nodes, and therefore some operations are performed in parallel, its design allows a sequential analysis. Thereafter, we prove that when GRPS is applied recursively in a network, its topology converges asymptotically to a *random* topology.

One of our contributions with respect to previous analysis is the model used in this work, representing P2P networks better. On one hand, Cooper et al. in [13] presented a similar protocol that builds a random regular network. However, its design is for undirected networks, while a P2P network uses directed communications. On the other hand, the RPS introduced in [8] by Gurevich and Keidar connects the in-degree and out-degree of each node. For a node v, its out-degree is the size of its partial view, and its in-degree is the number of nodes containing v in their partial views. In [8], the out-degree plus two times the in-degree of each node is bounded by three times the size of the partial views, a strong restriction for a P2P network. With GRPS, we extend the reachable topologies for the RPS and leave out dependencies between in- and out-degrees, also a desirable property considering the nature of P2P networks. Moreover, GRPS is designed such that self-loops and parallel links are avoided. That is a crucial point, because usually there is an application running on top of the RPS and using it. If that application requires c nodes to communicate with, it should be able to have them. A node communicating with itself has little interest in a distributed system, and communicating twice with the same node can be very different from communicating with 2 different nodes.

Secondly, we empirically test the convergence speed of GRPS, and its independence speed (roughly speaking, the time it takes to achieved a network independent from the current one). We show that GRPS converges faster and that has faster independence time than previously presented RPSs, particularly those presented by Mahlmann and Schindelhauer [14], and Gurevich and Keidar [8].

Finally, we present GRPSd, a variant of GRPS that preserves the original in and out-degrees for every node in the initial network. To this end, GRPS is modified such that if $d^t_{in}(u)$ (resp. $d^t_{out}(u)$) denotes the in-degree (resp. out-degree) of node u after t executions of GRPSd, then $d^0_{in}(u) = d^t_{in}(u)$ and $d^0_{out}(u) = d^t_{out}(u)$ for all t. This extension is motivated by the fact that some nodes could be more powerful than others and then admit more load, hence a higher in-degree. Therefore, even if the nodes pointing to some node u are periodically changing, the number of nodes pointing to u is maintained constant at any time during the execution. Finally, we discuss how new nodes are included into the network.

B. Related Work

Bonnet et al. in [15] study Cyclon, an RPS protocol where each node maintains a partial view with constant size c and exchanges l (parameter of Cyclon) of their elements with another node at each operation. The authors proved that the stationary distribution of the in-degrees follows a normal distribution centered at c and with variance equal to $c + O(1/n)$, where n is the number of nodes in the network. This work is fundamental for our purpose because this result provides GRPS with load balancing property, since the RPS analyzed in [15] is equivalent to GRPS.

There exists different works with empirical studies of RPSs. In [7], Jelasity et al. presented the RPS as a building block for any gossip-based protocol. They proposed a framework to implement an RPS in a decentralized manner. Using its framework, the authors empirically compare the behavior of several RPSs, demonstrating the uniformity of the partial views experimentally. Yet, this work is not backed up by a theoretical analysis. Bortnikov et al. [9] presented Brahms, a byzantine resilient RPS that provides a uniform sample. However, the resulting sample is not dynamically updated. Brahms ensures an eventual random sample without churn. In case of churn, the whole process has to be restarted. PuppetCast, introduced in [16], is an RPS that supports malicious nodes. Again, the study of PuppetCast remains empirical. Cyclon is an RPS presented in [17] by Voulgaris et al. Cyclon constructs graphs with low diameter, low clustering and highly symmetric node degrees. Also, it is shown that Cyclon is resilient to node failures. Nevertheless, the study of Cylcon also remains empirical.

On the theory side, Mahlmann and Shindelhauer in [18] presented k-Flipper, a graph transformation algorithm that transform regular undirected graphs, preserving regularity and connectivity. The authors use a random version of k-Flipper in order to create random regular connected undirected graphs. Later, Cooper et al. in [13] introduce a random protocol working on regular graphs that sample from all such graphs almost uniformly at random. They proved polynomial convergence time in the size of the network and $\log \epsilon^{-1}$, where ϵ is the error of the uniform sample. But those analysis are done for undirected networks, whereas P2P networks are better represented with directed networks due to the directed nature of their communications. A node may be present in the partial view of another node without this condition to be satisfied in the opposite direction.

In the framework of directed networks, Mahlmann and Shindelhauer in [14] presented Pointer-Push&Pull, a local random graph transformation for multi-digraphs with regular out-degree which produces every such graph with equal probability. Nevertheless, Pointer-Push&Pull may produce parallel links and self-loops, while these situations are avoided in GRPS. Furthermore, we empirically show that GRPS produces a random network faster than Pointer-Push&Pull, also we show a better independence time. Finally, Gurevich and Keidar [8] described the desirable properties for any RPS. Moreover, they also proposed an RPS satisfying these properties. Nevertheless, the RPS proposed

in [8] is restricted by the assumption that all the nodes must satisfy a strong relation between their in and out-degree, the sum of the out-degree plus two times the in-degree has to be bounded. This assumption restricts considerably the range of overlay networks that the RPS can build. Contrary to this assumption, in our work we only assume that the out-degree of each node is constant, and there is no relation between in-degrees and out-degrees. Thus GRPS opens widely the range of possible topologies.

C. Road Map

The rest of the paper is organized as follows. In Section II, we describe the model used in the analysis. In Subsection III-A, we present the gossip operation between two peers, the building block for GRPS presented in Subsection III-B. In Section IV, the analysis is described showing small views, load balancing, and uniform samples properties. Subsection IV-B provides an empirical analysis showing fast convergence and independence for GRPS. In Section V, we extend GRPS in order to preserve the initial in and out-degrees of each node. We also discuss how to include nodes arriving to the network. Finally, we give some concluding remarks in Section VI.

II. Model

The main goal of this work is the construction of an unstructured network (an overlay topology for a P2P system). We represent a network as a directed graph denoted by $\mathcal{G} = (V, E)$. Let us denote by V the set of peers in the network, and by n the size of V ($|V| = n$). We assume that each peer has some local knowledge of the network, i.e., each node knows a subset of nodes in the graph. Let us denote by N_u the knowledge of node u, the *partial view* of u. In this work, we assume that all partial views have the same size, which does not depend on n. It is denoted by c, $|N_u| = c$ for all node u.

The set of edges E is defined by the partial views of the nodes. E contains a directed edge (u, v) if and only if the node v is in the partial view of u, i.e, $(u, v) \in E \Leftrightarrow v \in N_u$. We assume that $\mathcal{G}$ does not contain loops, $u \notin N_u$. Also, we assume that $\mathcal{G}$ does not contain parallel edges[1], $\forall v, k \in N_u, v \neq k$. As explained in the introduction, self loops and duplicated neighbors prevent the application using the RPS to contact c real different peers. Hence, with these assumptions, $\mathcal{G}$ is a directed graph without loops, without parallel edges, and with constant out-degree equal to c.

We assume that the partial views contain sufficient information to start a communication process with each of the nodes on it, i.e., IP address, identifier, etc. Hence, a node is able to initiate a communication process with each of their neighbors. Also, a node can continue a communication process with all the nodes from which it has received a message.

We assume that the time is divided in synchronous steps. On the other hand, we do not consider arbitrary behaviors. All peers answer all queries they receive, and the answer is true. We also assume that there are no failures in the communication process; all the sent messages will be received by its addressee at the same time-step the message is sent. The last assumption is only for analysis purposes because RPS are known to cope with messages losses.

[1]In this work, we refer two edges as parallel if they are the same edge, with the same tail and head. We do not consider two edges as parallel if they are in opposite directions.

III. Generic Random Peer Sampling Service

GRPS uses as building block a gossip operation that randomly exchanges information between two nodes in the network. We first present the gossip operation, called *random exchange*, and describe their properties. Then, constructively, we introduce GRPS. For the presentation of the random exchange and GRPS, we define two functions. The function *request*(N) is the function used to request information N, i.e., if node p sends *request*(N) to node r, then r answers by sending N to p. The function *choose*(l, N) is the function that chooses uniformly at random l different elements over set N, by abuse of notation we use $\{$*choose*$(l, N)\}$ to denote the set of chosen elements. Therefore, the active thread on GRPS uses *choose*(l, N) function for the peer selection and data processing, and random exchange function for data exchange. The precise description is presented in the following subsection.

A. Random Exchange

We call *random exchange* the interaction between two nodes in the network described in Algorithms 1 (active thread) and 2 (passive thread). The goal of the random exchange, as its name says, is to randomly exchange information between the two participants.

Algorithm 1 Random Exchange Petitioner p: Active Thread

1: Set $N := \{\emptyset\}$ and $M := \{\emptyset\}$;
2: Send *request*(N_r) to the *replier* r;
3: Wait for the answer during t time-steps;
4: **if** The *replier* answer within the t time-steps **then**
5: **if** $p \in N_r$ **then**
6: Set $N_r := N_r - \{p\}$;
7: **end if**
8: Set $N := N_p \cup N_r$;
9: Set $M := \{$*choose*$(c, N)\}$;
10: Send back to the *replier* M and $N \backslash M$;
11: UPDATE $N_p := M$, the new partial view of p;
12: **else** $\{r$ has not answered after t time-steps are elapsed$\}$
13: Abort the *random exchange*;
14: **end if**

A random exchange involves two nodes: the *petitioner* and the *replier*. The petitioner is the node initializing the

Algorithm 2 Random Exchange Replier r: Passive Thread

1: If performing, finish the previous *random exchange*;
2: Set $K := \{\emptyset\}$;
3: Send N_r to the *petitioner*, wait for the answer;
4: **if** $r \in N \backslash M$ **then**
5: Set $N \backslash M := N \backslash M - \{r\} \cup \{p\}$;
6: **end if**
7: Set $K := \{choose(c - |N \backslash M|, M - \{r\})\}$;
8: UPDATE $N_r := N \backslash M \cup K$, the new partial view of r;

exchange and running the active thread. The replier answers the requests of the petitioner by running the passive thread. This is done periodically. The petitioner p starts the exchange by sending a request to the replier r asking for its partial view N_r, p uses function *request*(N_r) to do this (line 2 of Algorithm 1). Due to the model, p can start a random exchange only with one of the nodes in its partial view. Hence, we assume that r is in N_p. The replier r is chosen uniformly at random by p using the function *choose*$(1, N_p)$. Since r has received *request*(N_r), it can and shall answer. Once r answers the request sending its partial view N_r (line 3 of Algorithm 2), p checks if its own name is in N_r. In that case, p deletes its name in order to avoid creating a loop (lines 5 - 7 of Algorithm 1). Then, p computes the union as a set function of N_r and its own partial view, i.e., $N = N_p \cup N_r$ is computed without repetitions (line 8 of Algorithm 1). Among them, p chooses uniformly at random c different elements using the function *choose*(c, N), to compose its new partial view (line 9 of Algorithm 1). Finally, p sends to r its new partial view and the list of nodes that were not included on it (line 10 of Algorithm 1).

On the other hand, r also updates its partial view. In order to minimize the loss of information (swapper mode), r must keep on its partial view the elements that were not included in p's partial view. Therefore, it takes the nodes that were not kept by p as part of its new partial view, and also r completes its partial view up to c elements by choosing uniformly at random the rest of the nodes among the set M (line 7 of Algorithm 2). If p does not keep r in its partial view, then r is forced to keep p in its own (lines 4 and 5 of Algorithm 2). Furthermore, if p keeps r in its partial view, then r avoids creating a loop (line 7 of Algorithm 2). The algorithms for the petitioner and the replier are precisely described in Algorithms 1 and 2, respectively.

Notice that, using GRPS, the size of a new partial view is equal to c. In line 7 of Algorithm 2, r chooses $c - |N \backslash M|$ elements to complete its new partial view, plus the $|N \backslash M|$ elements it must keep as neighbors. Consequently, the size of its new N_r is equal to c. Also, by line 9 in Algorithm 1 p chooses c elements for its new N_p. Other observations about GRPS follows.

Remark 3.1: There are no repeated nodes in a new partial view, i.e., the new network topology does not contain parallel links.

The fact that GRPS does no produces repeated nodes in new partial views comes from the definition of the function *choose*(l, N), and from Algorithms 1 and 2. Because *choose*(l, N) chooses different elements, and the algorithms use set union, hence there are no duplicated elements neither in N nor in M, which is equal to the new partial view of p. Moreover, the new N_r is created by the union of $N \backslash M$ with a subset of M, therefore, N_r does not contain duplicated elements.

Remark 3.2: There are no self citations in a new partial view, i.e., the new network topology does not contain self-loops.

This follows from lines 5 to 7 in Algorithm 1 and line 7 in Algorithm 2. In both cases, p and r delete themselves, if required, to avoid creating self-loops.

Remark 3.3: Nodes p and r are connected in the new network topology.

Remark 3.3 follows from lines 4 and 5 of Algorithm 2. The condition of the **if** in line 4 asks if p has r in its new partial view. If it does not contain r, then r must include p in its new partial view (line 5 of Algorithm 2).

Remark 3.4: The union of the new N_p and new N_r is equal to the union of the original N_p and N_r, except, maybe, for p and r.

Remark 3.4 follows from the construction of N in line 8 of Algorithm 1. At that point, p computes the union of the original N_p and N_r. Then, in the rest of the random exchange, the new N_p and new N_r are build such that their union is equal to N (lines 8 to 11 in Algorithm 1 and lines 7 and 8 in Algorithm 2). Now, we state the first lemma.

Lemma 3.5: Let $\mathcal{G}$ be a simple *connected* directed graph with constant out-degree equal to c. If $\mathcal{G}'$ is a directed graph obtained by executing one random exchange in two nodes of $\mathcal{G}$. Then, $\mathcal{G}'$ is a simple *connected* directed graph with constant out-degree equal to c.

Proof: Part of the lemma comes from the previous remarks, hence we only have to prove the connectivity of $\mathcal{G}'$. Let u and v be two nodes in $V(\mathcal{G}) = V(\mathcal{G}')$. Since we assume that $\mathcal{G}$ is connected, there exists a path (not necessarily directed) connecting v with u. Then, we prove that there is a path connecting v and u in $\mathcal{G}'$. Let us denote by P_{vu} the shortest path connecting v with u. Figure 1 illustrates the notation of the proof.

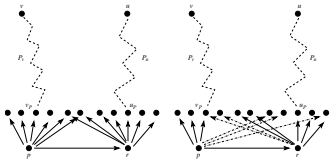

Figure 1. Example of the structures described in proof of Lemma 3.5.

Since a random exchange between p and r only modifies

the partial views of p and r, then, in the case that neither p nor r are contained in P_{vu}, the whole P_{vu} also exists in $\mathcal{G}'$. Hence, we assume that p and/or r are contained in P_{vu}. Furthermore, if both p and r are contained in P_{vu}, then one is followed by the other in P_{vu}. Therefore, without loss of generality, P_{vu} can be described as $v, e_1, p_1, e_2, p_2 \dots v_P, e_l, p, (p, r), r, f_1, u_P, f_2, f_2, \dots, u$, where e_i and f_i are links in E, and p_i and r_i are nodes in V. Let us split P_{vu} in three parts. The first part is $P_v = v, e_1, p_1, e_2, p_2 \dots p_l$, the part between v and the neighbor of p denoted by v_P. The second part is $P_u = u_P, f_2, f_2, \dots, u$ the part between the neighbor of r denoted by u_P and u. And lastly, $P_{pr} = v_P, e_l, p, (p, r), r, f_1, u_P$ the part that contains p and r.

Note that P_u and P_v are contained in $\mathcal{G}'$, since they are not incident neither with p nor with r. Also, note that due to remark 3.3 and 3.4, there exists a path between v_P and u_P in $\mathcal{G}'$. Therefore, the union of P_u, P_v and the path between u_P and v_P is a path connecting u with v in $\mathcal{G}'$. ■

Another characteristic of the random exchange is the fact that once the random exchange has been executed, and then $\mathcal{G}$ has been transformed to $\mathcal{G}'$, it is possible to re obtain $\mathcal{G}$ by applying another random exchange to $\mathcal{G}'$. This is important for the posterior analysis and to prove our main result about GRPS. This property is better explained and proved in the following lemma.

Lemma 3.6: Let $\mathcal{G}'$ be a network obtained by applying a random exchange to $\mathcal{G}$. Let us denote by $P(\mathcal{G}|\mathcal{G}')$ the probability to obtain again $\mathcal{G}$ by applying a random exchange to $\mathcal{G}'$ on the same nodes. Then, this probability is strictly greater than zero,

$$P(\mathcal{G}|\mathcal{G}') > 0.$$

Proof: From Remark 3.3 (either $(p, r) \in \mathcal{G}'$, or $(r, p) \in \mathcal{G}'$, or both (p, r) and (r, p) are in $\mathcal{G}'$) it is possible to execute a random exchange in $\mathcal{G}'$ with the same nodes p and r that executed the random exchange in $\mathcal{G}$ producing $\mathcal{G}'$. Note that, this new random exchange may be executed either with p as the petitioner and r as the replier, when $(p, r) \in \mathcal{G}'$, or with r as the petitioner and p as the replier, when $(r, p) \in \mathcal{G}'$. In both cases, since the union of the partial views of p and r in $\mathcal{G}'$ is equal to the union of the partial views of p and r in $\mathcal{G}$ (remark 3.4), the probability to re obtain the original N_p and N_r is strictly greater than zero. Hence, the probability to re obtain $\mathcal{G}$ form $\mathcal{G}'$ by executing a random exchange in the same nodes is strictly greater than zero, $P(\mathcal{G}|\mathcal{G}') > 0$. ■

At that point, we would like to explain the reason why Algorithm 1 uses a timer, and also why it is split in two parts, when the replier answers and when it does not. The random exchange is executed in parallel by several peers in the network. In order to keep an order among the parallel random exchanges, none of the peers can be involved in more than one random exchange at the same time. Thus, the replier does not answer any random exchange until it has finished the random exchange it is executing, if this is the case (line 1 Algorithm 2). Therefore, a chain of peers may be waiting for an answer to be produced. That could lead to a deadlock. The timer is introduced precisely to break such situations by deciding abort a random exchange. This situation is better explained in Subsection III-B, and also in the main algorithm GRPS.

B. The GRPS Algorithm

In this subsection, we present GRPS, the random peer sampling service based on the random exchange explained in the precedent subsection. First, we will denote by *random.exchange*(r) the function that executes a random exchange described in Subsection III-A with node r, where the executing node plays the role of the petitioner (Algorithm 1) and r plays the role of the replier (Algorithm 2). GRPS is precisely described in Algorithm 3.

Algorithm 3 GRPS Algorithm

```
1: Decide with probability q to be a petitioner node;
2: if p has decided to be a petitioner then
3:     Set r := {choose(1, N_p)};
4:     Execute random.exchange(r);
5:     Pass to the next round;
6: else {p has not decided to be a petitioner}
7:     Pass to the next round;
8: end if
```

In GRPS, each node periodically executes Algorithm 3. The algorithm starts by deciding randomly if the node starts a random exchange with one of their neighbors (line 1 of Algorithm 3). If the node decides to execute a random exchange, then it chooses one of the nodes in its partial view uniformly at random. The chosen neighbor is the replier of the random exchange (lines 3 and 4 of Algorithm 3). If the node decides not to execute a random exchange, it passes to the next round.

To start the analysis of GRPS, we would like to show that, although this is a parallel process, it allows a sequential representation, and hence can be analyzed as a sequential process. To do that, we first define a random exchange as *real* if and only if the exchange has taken place, i.e., the petitioner and the replier have updated their partial views. On the other hand, every set of real random exchanges $\mathcal{RE}$ is associated with an implicit order, the order in which the real random exchanges in $\mathcal{RE}$ are executed. We say that $\mathcal{RE}$ admits a *sequential order* if and only if there exists a total order of $\mathcal{RE}$, i.e., a bijection between the set $\mathcal{RE}$ and the set $\{1, 2, 3, \dots, |\mathcal{RE}|\}$, such that the resulting network after executing $\mathcal{RE}$ on its implicit order (not necessarily total) is equal to the network obtained after executing $\mathcal{RE}$ following the total order induced by the bijection. Therefore, we state the following lemma.

Lemma 3.7: Let $\mathcal{G}$ be a network executing GRPS, i.e., each peer in $\mathcal{G}$ is constantly executing Algorithm 3. Let $\mathcal{RE}(t)$ be the set of all the *real* random exchanges executed until time-step t. Then, for all time-step t, $\mathcal{RE}(t)$ admits a *sequential order*.

Proof: The proof is by induction over t. The base of the induction is to give a total order on $\mathcal{RE}(1)$, the set of random exchanges executed in the first time-step. $\mathcal{RE}(1)$ is a set of parallel random exchanges, i.e., all of them are executed at the same time. Due to the characteristics of GRPS, none of the nodes can execute random exchanges at the same time. Therefore, none of the random exchanges in $\mathcal{RE}(1)$ can share common nodes. Consequently, any sequential order in $\mathcal{RE}(1)$ shall produce the same network than the network produced when all of them are executed in parallel.

Now, we assume that $\mathcal{RE}(t-1)$ admits a sequential order. Since, all the random exchanges in $\mathcal{RE}(t)/\mathcal{RE}(t-1)$ are executed at the same time, none of them share a node, again due to the nature of the algorithm. Hence, starting from the network at time-step $t-1$, any sequential order in $\mathcal{RE}(t)/\mathcal{RE}(t-1)$ produces the same network than the network produced by all of them executed in parallel. Therefore, the lemma is proved. ■

Corollary 3.8: It follows from Lemma 3.7 that GRPS can be analyzed as if it has been executed sequentially.

IV. Satisfying the Desirable Properties

In this section, we present the analysis proving that GRPS satisfies the desirable properties. We start with *small views* and *load balancing* properties.

Small views: From the design of GRPS, it follows that the size of the partial views of every node is a parameter of the system denoted by c, and it keeps constant during any execution of GRPS. It can be set at the beginning as a value that depends or not on the total size of the network n. Hence, it follows that GRPS satisfies the desirable property of small partial views. Furthermore, since it is proved that GRPS keeps connectivity of the network all over an execution without matter the size of c, therefore GRPS does not require large views to provide probabilistic guarantees about connectivity.

Load balancing: In order to ensure the load balancing property for GRPS, we rely on a previous work. As mentioned in Subsection I-B, Bonnet et al. in [15] studied Cyclon in-degrees distribution, i.e., they studied the in-degrees distribution in a network produced after Cyclon is executed and has reached a steady state. GRPS is a special case of Cyclon, hence we can use their results. The authors of I-B characterize such a distribution as a normal distribution centered in the out-degree of the nodes, c. They proved that the variance of the distribution is $c + O(1/n)$. Therefore, GRPS also satisfies the second desirable property, load balancing.

A. The Markov Process and Consequences

GRPS can be seen as a Markov process. Then, let us denote by $\mathcal{M} = (\mathcal{V}, \mathcal{E})$ the directed graph of the Markov chain produced by GRPS. The set of nodes $\mathcal{V}$ is the set of all the simple directed graphs with constant out-degree equal to c. There exists a directed edge going from $\mathcal{G}$ to $\mathcal{G}'$ if there exists a random exchange that transforms $\mathcal{G}$ in $\mathcal{G}'$, i.e., $\mathcal{E} = \{(\mathcal{G}, \mathcal{G}') : P(\mathcal{G}|\mathcal{G}') > 0\}$, where $P(\mathcal{G}|\mathcal{G}')$ denotes the probability to produce $\mathcal{G}'$ from $\mathcal{G}$ through a random exchange. By Lemma 3.6, we have that for every edge in $\mathcal{E}$ also the inverse edge is in $\mathcal{E}$, i.e., $(\mathcal{G}, \mathcal{G}') \in \mathcal{E} \Leftrightarrow (\mathcal{G}', \mathcal{G}) \in \mathcal{E}$.

Remark 4.1: The probability to obtain $\mathcal{G}'$ from $\mathcal{G}$ by a random exchange is the same than the probability to obtain $\mathcal{G}$ from $\mathcal{G}'$ by a random exchange, $P(\mathcal{G}|\mathcal{G}') = P(\mathcal{G}'|\mathcal{G})$.
To understand the previous remark, it is necessary to go back to Remark 3.4 and to Algorithms 1 and 2. From Remark 3.4, if $\mathcal{G}'$ is obtained from $\mathcal{G}$ by a random exchange between p and r, then the union of N_p and N_r in $\mathcal{G}$ and $\mathcal{G}'$ are equal. Since the random exchange chooses the new partial views uniformly at random over the union of the two involved partial views, then $P(\mathcal{G}|\mathcal{G}') = P(\mathcal{G}'|\mathcal{G})$.

Due to the fact that $(\mathcal{G}, \mathcal{G}') \in \mathcal{E} \Leftrightarrow (\mathcal{G}', \mathcal{G}) \in \mathcal{E}$, and the fact that $P(\mathcal{G}|\mathcal{G}') = P(\mathcal{G}'|\mathcal{G})$, we can consider $\mathcal{M}$ as an undirected Markov chain graph. Also, from design of the random exchange, for every $\mathcal{G} \in \mathcal{V}$ the probability to go from $\mathcal{G}$ to the same node $\mathcal{G}$ is greater than zero, $P(\mathcal{G}|\mathcal{G}) > 0$. Therefore, the Markov chain $\mathcal{M}$ is aperiodic. On the other hand, since the sum of all the probabilities to go from $\mathcal{G}$ to a different network is equal to 1 for every network $\mathcal{G} \in \mathcal{V}$, $\sum_{\mathcal{G}' \in \mathcal{V}} P(\mathcal{G}|\mathcal{G}') = 1$. Therefore, the Markov chain $\mathcal{M}$ is regular, let us say $1/q$-regular for some q[2]. Therefore, the graph $\mathcal{M}$ of the Markov chain produced by GRPS is a $1/q$-regular multigraph, where for all $P(\mathcal{G}|\mathcal{G}') = c(\mathcal{G}|\mathcal{G})\dot{q} > 0$ there are $c(\mathcal{G}|\mathcal{G})$ edges going from $\mathcal{G}$ to $\mathcal{G}'$, each edge represents a fraction q of its respective probability.

Four basic operations are determined by the random exchange. These four operations allow to construct any network, regardless of the initial network, by using repeatedly random exchange function i.e., these four operations allow us to prove that $\mathcal{M}$ is connected. Figure 2 makes a graphical representation of the four operations.

Operation 1: Exchange of tails between two neighbors. Operation 1 is described as follows: if (a, v), (a, b), and (b, u) are three links connecting nodes a, b and u, then, after apply Operation 1, the new links are (a, u), (a, b), and (b, v). Operation 1 is obtained by the following random exchange. Node a plays the role of the petitioner, node b of the replier. Node a chooses its new partial view N_a equal to its present partial view but exchanging v with u. After that, the partial view of node b is equal to the present partial

[2]Since, the probabilities are values in $\mathbb{Q}$ (the set of rational numbers), then q can be defined as the maximum common divisor among all the probabilities

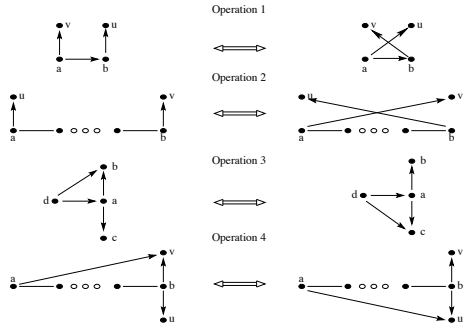

Figure 2. Four basic operations to change the in-degrees distribution.

view but exchanging u with v.

Operation 2: An extension of Operation 1. In that case, the exchange of tails takes place between two nodes connected by a path. Operation 2 is described as follows: let a and b be two nodes connected by the path P (not necessarily a directed path). Also, let (a, v) and (b, u) be two links in the network. Then, after applying Operation 2, links (a, v) and (b, u) are changed by links (a, u) and (b, v). Operation 2 is obtained by applying Operation 1 along the path P connecting a and b. Let $p_1, p_2, \ldots, p_l$ be the nodes in P from a to b. Assuming that c is greater than 3, for each node p_i there exists a pivot edge (p_i, v_i). Then, applying Operation 1 first between a and p_1, then between p_1 and p_2, and so on until b, the link (a, v) is changed by link (b, v). After that, repeating the procedure going back from b to a, link (b, u) is changed by (a, u), and the rest of the links return to their original positions.

Operation 3: Change the head of a link. Operation 3 is described as follows: if (a, b), (a, c), (a, d), and (d, b) are four links in the network. Then, after applying Operation 3, the new links are (a, b), (a, c), (a, d), and (d, c). Operation 3 is obtained by applying a random exchange between nodes a and d. Node a acts as the petitioner and node b as the replier. Node a chooses the same partial view. Node d updates its partial view by exchanging nodes b with c. Note that, this operation is also valid if nodes a and d are connected by a link going from d to a, in that case the petitioner is d and the replier is a, but both nodes choose their new partial views as described before.

Operation 4: An extension of Operation 3. Operation 4 is described as follows: nodes a and b are connected by a path (not necessarily directed), and c is the first node in the path that connects a with b going from b to a. Links (b, v), (b, u), and (a, v) are in the network. Then, after applying Operation 4, the new links are (b, v), (b, u), and (a, u), moreover the path connecting a and b is not affected. Operation 4 is obtained by applying the first part of Operation 2 between nodes a and c. It creates a situation in which Operation 3 is applied for nodes b and c. Finally, the second part of Operation 2 gives back the tail of edge (at this moment) (c, u) to node a. Also, the way back of Operation 2, gives back to the original positions the pivot links used in the first part.

Lemma 4.2: The $1/q$-regular undirected multigraph $\mathcal{M}$ of the Markov chain produced by GRPS is *connected*.

Proof: To prove that $\mathcal{M}$ is connected, it is required to show that, for every $\mathcal{G}$ and $\mathcal{G}'$ in $\mathcal{E}$, there is a path in $\mathcal{M}$ connecting $\mathcal{G}$ with $\mathcal{G}'$. To do so, first notice that operations 3 and 4 allow us to modify the in-degree distribution of a network. Since for all pair of nodes in V there exists a path connecting them, with Operations 3 and 4 it's possible to move head of links from nodes overloaded with head of links to nodes lacking of heads of links. Due to the conservation of links, if there are nodes lacking of heads, then also there are overloaded nodes. Therefore, by using Operations 3 and 4, it is possible to change $\mathcal{G}$ by an intermediate network $\mathcal{G}''$ with the same in-degree distribution that $\mathcal{G}'$, i.e., the in-degree of u in $\mathcal{G}''$ is equal to the in degree of u in $\mathcal{G}'$, and that is true for all u in V.

Secondly, notice that Operations 1 and 2 allow us to exchange tails of links. Thereafter, when the right in-degree distribution is obtained, by using Operation 1 and 2, the tails of links can be moved to the right nodes. Let us say that node u receives a wrong link, i.e., $u \in N_v$ in $\mathcal{G}''$ for some v, but $u \notin N_v$ in $\mathcal{G}'$ for the same v. Since the in-degree of u in $\mathcal{G}''$ is equal to the in degree of u in $\mathcal{G}'$, then there exists node r such that $u \notin N_r$ in $\mathcal{G}''$, but $u \in N_r$ in $\mathcal{G}'$. Furthermore, since every out degree in $\mathcal{G}'$ and $\mathcal{G}''$ is equal to c (particularly r's out-degree), then r is pointing a wrong node, i.e., there exists $p \in N_r$ in $\mathcal{G}''$, but $p \notin N_r$ in $\mathcal{G}'$. Now, since $\mathcal{G}''$ is connected, then the tail of (v, u) can be exchanged with the tail of (r, p) by using operations 1 or 2, therefore create (r, u) and (v, p). Consequently, the number of wrong edges is decreased by one. Applying repeatedly that exchanges $\mathcal{G}'$ is obtained from $\mathcal{G}''$. Hence, there exists a path in $\mathcal{M}$ connecting $\mathcal{G}$ to $\mathcal{G}'$. ■

Now, using the properties of the Markov chain graph $\mathcal{M}$ obtained by GRPS, it is possible to state the following Theorem.

Corollary 4.3: The stationary distribution of the Markov chain process defined by the normalized adjacency matrix of $\mathcal{M}$ is the uniform distribution;

$$u = (1/|\mathcal{V}|, 1/|\mathcal{V}|, 1/|\mathcal{V}|, \ldots, 1/|\mathcal{V}|).$$

In other words, If we denote by $\pi_t(\mathcal{G}|\mathcal{G}')$ the probability to obtain a simple directed network $\mathcal{G}'$ at time t, when the initial simple directed network is $\mathcal{G}$, then:

$$\pi_t(\mathcal{G}|\mathcal{G}') \overset{t\to\infty}{\longrightarrow} \frac{1}{|\mathcal{V}|} \quad \forall\mathcal{G} \quad \text{and} \quad \forall\mathcal{G}' \quad \in \mathcal{V}.$$

The corollary comes directly from well know results about Markov chains. For the interested readers, we recommend the book Probability and Computing by M. Mitzenmacher and E. Upfal [19]. The corollary tells us, in terms of a

random walk in $\mathcal{M}$, that regardless of where the random walk has started, after a certain number of steps, it has the same probability to be in any node.

Uniform sample: Uniform sample for GRPS is stated in the following theorem.

Theorem 4.4: Let $\mathcal{G} = (V, E)$ be a network obtained by repeating sufficiently many times GRPS. Let us denote by $P(u, v)$ the probability of the event $(u, v) \in E$, the link (u, v) is in network $\mathcal{G}$. Then, all the directed links rooted in u have the same probability to be in the network, i.e.,

$$P(u, v) = P(u, r) \quad \forall v \quad \text{and} \quad \forall r \quad \in V.$$

Proof: The proof starts by computing the probability for a link to be in the network $\mathcal{G}$. By symmetry, the number of networks with a link (u, v) is the same that the number of networks with a link (u, r), and that is for every node r. In other words, if we denote by $\mathcal{E}(u, v)$ the set of all the networks containing link (u, v), then $|\mathcal{E}(u, v)| = |\mathcal{E}(u, r)|$ for all nodes v and r in V. To show this claim, we construct a bijection between $\mathcal{E}(u, v)$ and $\mathcal{E}(u, r)$. The bijection, first, sends every element in $\mathcal{E}(u, v) \bigcap \mathcal{E}(u, r)$ on itself. Then, every network $\mathcal{G}$ in $\mathcal{E}(u, v)$ will be sent to $\mathcal{G} - \{(u, v)\} \cup \{(u, r)\}$, the same network without (u, v) but with (u, r).

Now, the probability for a link to be in the final network $\mathcal{G}$ is given by the number of networks containing the link, divided by the size of the set of all the networks, $P(u, v) = |\mathcal{E}(u, v)|/|\mathcal{V}|$. Therefore, using the previous claim, we can conclude that $P(u, v) = |\mathcal{E}(u, v)|/|\mathcal{V}| = |\mathcal{E}(u, r)|/|\mathcal{V}| = P(u, r)$. ∎

Therefore, it is possible to conclude that GRPS also satisfies the uniform sample property.

Independence: Using Corollary 4.3, it is possible to conclude that, regardless of the initial network, after recursively executing GRPS, the probability to obtain any $\mathcal{G}$, simple network with out-degree equal to c, is approximately equal to $\frac{1}{|\mathcal{V}|}$, where $|\mathcal{V}|$ is the total number of simple networks with constant degree equal to c. Therefore, independence from the initial network is obtained after the Markov process converges to its stationary state.

Albeit, it is possible to obtain bounds on the convergence and independence time using Markov chain theory, usually those bounds might be too loose to assess what really happens in practice. Therefore, we decided to empirically test convergence and independence time. Furthermore, in that way we can compare GRPS with other RPS already present in the literature.

B. Empirical Analysis: Convergence and Independence

We experimentally study GRPS as well as two other RPS protocols, namely *send and forget* [8] and *pointer-push&pull* [14]. These experiments were conducted with a P2P simulator and involve 500 peers building random views of size 10. *Send and forget* was configured with $dL = 2$ and $s = 18$ which, according to the paper, should produce an average out-degree of 10.1. At each cycle, each peer executes one RPS action and a snapshot of the network is saved for further analyzis. We show the results from a given execution, to avoid the smooth produced by averaging, but were confirmed over several executions.

Convergence speed: In the first experiment, we study the convergence speed of the protocols. We consider two different network configurations as starting points. In the first one, the peers form a ring structure and each node is connected to the 10 following peers on the ring. In the second one, the core of the network consists in a clique of size 11 and all the other peers are connected to 10 peers in the clique. Both these networks are very structured and exhibit a high clustering coefficient. We evaluate the randomness of the networks through the usual metrics, namely the average shortest path between nodes, the diameter of the graph and the clustering coefficient. Since GRPS guarantees connectivity (as opposed to strong connectivity), we perform these measures on the undirected version of the network. Figure 3 shows the evolution of the clustering coefficient of the network. For the ring configuration, GRPS clearly outperforms the other protocols. On the cluster configuration however, *pointer-push&pull* evolves slightly faster at the beginning. This is caused by the particular starting configuration of the network. Since all nodes almost share the same views, at the very beginning, a peer outside the cluster exchanging neighbors with a peer from the cluster has a 0.5 probability not to change the overlay in GRPS, while *pointer-push&pull* will always generate a modification. But as soon as some randomness has been introduced in the network (after cycle 5), GRPS quickly becomes much more efficient. *Send and forget* has a lot of difficulties to mix the network with the cluster configuration. Indeed, the nodes at the center become overloaded and drop the links that are sent to them, resulting in many deletions and a diminution of the average out-degree to around 3. The out-degree later increases but this result shows how sensitive to the starting configuration *send and forget* is.

Independence: The second experiment highlights the ability of the protocols to quickly generate independent networks. Starting from a random configuration (once the protocols have converged) at cycle 0, we measure the number of differences between this reference configuration and the graph at each cycle, namely the number of edges present in one graph and not in the other. The results, depicted on Figure 4 shows that GRPS is able to modify the network much faster. The number of differences is normalized with respect to its maximal value, namely $10,000$. Since the networks are random, there can always be a few edges in common between the two graphs, so the difference never reaches 1 in this experiments. Still, the results show that in as few as 4 cycles, GRPS produces a graph completely independent from the starting configuration while it takes respectively 40 and 120 cycles to competitors to reach the

same result.

These results show that GRPS produces an network which is very dynamic. It converges quickly from very clustered configurations and is able to produce independent topologies in just a few cycle. This property is due to the fact that GRPS exchanges several links at each RPS operation, while the two other protocols only modify two. Due to space limitations, we do not show results for different view sizes, but since the number of links exchanged in GRPS depend on the view size, larger views lead to even better results for GRPS. We also considered slightly modified version of *send and forget* and *pointer-push&pull* ensuring properties similar to GRPS, i.e. no loops and no parallel edges, but the results we obtained were very similar.

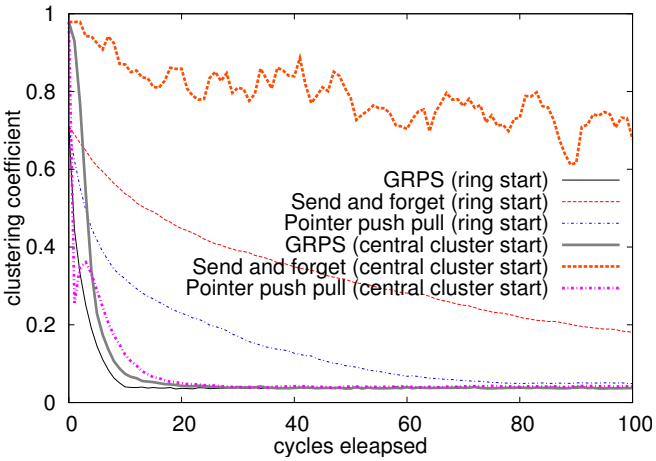

Figure 3. Evolution of the clustering coefficient from a structured network to a random network.

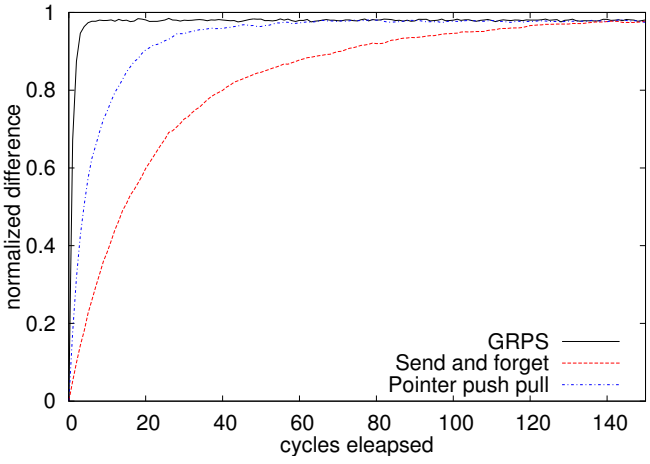

Figure 4. Differences between successive states of the network.

V. Extending the Peer Sampling Service

In this section, we propose one extension to GRPS. We present a small modification that, besides the above explained properties, preserves in and out-degrees of all the nodes during all the execution. Finally, we discuss how to receive new nodes in the network, and we give details about how to include them in the system. An RPS that preserves the initial in and out-degrees of the nodes while producing a random overlay network topology is interesting as it is then possible to generate a natural load balancing mechanism for the network. For instance, each node can initially choose its in-degree and out-degree according to its capabilities. The modification we propose to GRPS impacts the random exchange. The intuition is to keep the repetitions of the nodes in the union of the two partial views. We call that extension GRPSd.

Note that, due to the no loops and no parallel edges assumptions, when N is defined in the random exchange by the petitioner, N can not include a node more than twice. On the other hand, note that the size of N as a multi-set (i.e., counting each element with its repetitions) is equal to $2c$. To preserve the in-degrees, the function that chooses the new partial view of the petitioner has to be slightly modified. This function is called $choose'(c, N)$. It chooses c different elements in the multi-set N, and ensures that if an element appears twice in N, it must be chosen by $choose'(c, N)$.

The union computed in the random exchange is now a multi-set union, i.e., if an element appears in both sets, then it appears twice in the union (line 6, Algorithm 4). Therefore, the petitioner will choose its new partial view among the multi-set generated by the union of their current partial view and the partial view sent by the replier. To do so, the petitioner uses $choose'(c, N)$ (line 7 of Algorithm 4). Finally, the petitioner sends to the replier the remaining c elements (line 8 of Algorithm 4), hence the replier has not to choose its new partial view, but keep as new partial view what the petitioner has sent it (line 7 of Algorithm 5). In this case, the petitioner must keep the replier as a neighbor. If the replier has the petitioner in its list, then the replier has to keep it as well. This prevents the creation of loops. The new random exchange is described precisely in Algorithm 4 (active thread) and Algorithm 5 (passive thread). With this new algorithms for the random exchange, GRPSd is changing the network topology among networks with n nodes, constant out-degree equal to c for all nodes, and in-degrees determined by the initial network.

How to Deal with New Nodes?: P2P networks are known to be dynamic, i.e., nodes join and leave the system continuously. Hence, for any service designed to work in a P2P network, dealing with churn is of the utmost importance.

Our first approach is to define an insertion protocol used by the joining nodes. We define the function $new.request(N)$ equivalently to function $request(N)$, but it does not start a random exchange. Instead, it requests the set N, but, the node receiving this request answers only, without starting a more complex process. Also, we assume that a joining node knows some node in the network to bootstrap. Then, the

Algorithm 4 GRPSd Petitioner p: Active Thread

Set $N := \{\emptyset\}$ and $M := \{\emptyset\}$;
Send *request*(N_r) to the *replier* r;
Wait for the answer during t time-steps;
if The *replier* answer within the t time-steps **then**
    Set $N_r := N_r - \{p\} - \{r\}$;
    Set $N := N_p \cup N_r$, the multi-set union;
    Set $M := \{choose'(c-1, N)\}$;
    Send back to the *replier* $N \backslash M$;
    UPDATE $N_p := M \cup \{r\}$, the new set of neighbors of p;
else $\{r$ has not answered after t time-steps are elapsed$\}$
    Abort the *random exchange*;
end if

Algorithm 5 GRPSd Replier r: Passive Thread

If performing, finish the previous *random exchange*;
Set $K := \{\emptyset\}$;
Send N_r to the *petitioner*, wait for the answer;
if $|N \backslash M| = c - 1$ **then**
    Set $N \backslash M := N \backslash M \cup \{p\}$;
end if
UPDATE $N_r := N \backslash M$, the new set of neighbors of r;

joining node sends a *new.request*(N_r) to r, the node it knows previously. The joining node simply takes as neighbors the same set of neighbors it receives.

On the other hand, it is desirable to accept joining nodes in the network while preserving the degrees, then the new nodes must use in-degree and out-degree equal to c. That is due to fact that the sum of the in-degrees is always equal to the sum of the out-degrees, also equal to the number of links (each link has one head and one tail). That's a natural characteristic of directed networks. Hence, when a node joins the network, it first defines its in-degree equal to its out-degree, therefore the conservation of links is ensured in the network. Then, it builds as many loops as the desired in-degree. Thereafter, the joining node contacts the node it previously knows in the network, and exchanges one of their loops with a node in the partial view of that node. The new node repeats this process with the new node on its partial view. After, some random exchanges the loops will be spread over the network, and the joining node will be *absorbed* by the network.

VI. Conclusions and Future Work

In this work, we have presented GRPS, a generic RPS that guarantees the desirable properties for a robust RPS. We prove that GPRS provides each node with a *uniform sample* of all nodes. Furthermore, the resulting topology does not depend on the initial topology of the network. We provide theoretical analysis to prove the properties of GRPS. On the other hand, we empirically show fast convergence of GRPS to a random topology, and also that GRPS provides a fast independence regardless of the initial network. Thereafter, we extend GRPS to a service that preserves the initial degrees of the network. The main challenge that still remains open is to model churn in the system, and more specifically departing nodes to provide the same uniformity guarantees.

Acknowledgments: The authors would like to thank Francois Bonnet, Davide Frey, Marek Klonowski and Miroslaw Korzeniowski for fruitful discussions and comments on the work. This research was supported in part by French ANR project Shaman.

References

[1] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. E. Sturgis, D. C. Swinehart, and D. B. Terry, "Epidemic algorithms for replicated database maintenance," *Operating Systems Review*, vol. 22, no. 1, pp. 8–32, 1988.

[2] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 3, pp. 219–252, 2005.

[3] M. Jelasity, A. Montresor, and Ö. Babaoglu, "A modular paradigm for building self-organizing peer-to-peer applications," in *Proc. of the Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering*, 2003, pp. 265–282.

[4] S. Voulgaris and M. van Steen, "An epidemic protocol for managing routing tables in very large peer-to-peer networks," in *Proc. of DSOM*, 2003, pp. 41–54.

[5] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking, "Randomized rumor spreading," in *Proc. of FOCS*, 2000, pp. 565–574.

[6] Q. Sun and D. C. Sturman, "A gossip-based reliable multicast for large-scale high-throughput applications," in *Proc. of DSN*, 2000, p. 347.

[7] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, pp. 8–43, 2007.

[8] M. Gurevich and I. Keidar, "Correctness of gossip-based membership under message loss," in *Proc. of PODC*, 2009, pp. 151–160.

[9] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: byzantine resilient random membership sampling," in *Proc. of PODC*, 2008, pp. 145–154.

[10] Z. Bar-Yossef, R. Friedman, and G. Kliot, "Rawms -: random walk based lightweight membership service for wireless ad hoc network," in *Porc. of MobiHoc*, 2006, pp. 238–249.

[11] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," *P2P Computing Systems*, vol. 63, no. 3, pp. 241–263, 2006.

[12] C. Avin, M. Koucký, and Z. Lotker, "How to explore a fast-changing world (cover time of a simple random walk on evolving graphs)," in *Proc. of ICALP*, 2008, pp. 121–132.

[13] C. Cooper, M. Dyer, and A. J. Handley, "The flip markov chain and a randomising p2p protocol," in *Proc. of PODC*, 2009, pp. 141–150.

[14] P. Mahlmann and C. Schindelhauer, "Distributed random digraph transformations for peer-to-peer networks," in *Proc. of SPAA*, 2006, pp. 308–317.

[15] F. Bonnet, F. Tronel, and S. Voulgaris, "Brief announcement: Performance analysis of cyclon, an inexpensive membership management for unstructured p2p overlays," in *Proc. of DISC*, 2006, pp. 560–562.

[16] A. Bakker and M. van Steen, "Puppetcast: A secure peer sampling protocol," in *Proc. of EC2ND*, 2008, pp. 3–10.

[17] S. Voulgaris, D. Gavidia, and M. van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, 2005.

[18] P. Mahlmann and C. Schindelhauer, "Peer-to-peer networks based on random transformations of connected regular undirected graphs," in *Proc. of SPAA*, 2005, pp. 155–164.

[19] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. New York, NY, USA: Cambridge University Press, 2005.