Fast Generation and Mixing of Random Graphs in Peer-to-Peer Networks

Olivier Beaumont, Marcin Dojwa, Philippe Duchon, Robert Elsässer, Ralf Klasing, Miroslaw Korzeniowski

Fast Generation and Mixing of Random Graphs in Peer-to-Peer Networks

Olivier Beaumont
INRIA Bordeaux Sud-Ouest
Bordeaux, France
olivier.beaumont@labri.fr

Marcin Dojwa
LiveChat Software
Wroclaw, Poland
m.dojwa@livechatinc.com

Philippe Duchon
University of Bordeaux
Bordeaux, France
philippe.duchon@labri.fr

Robert Elsässer
University of Paderborn
Paderborn, Germany
elsa@uni-paderborn.de

Ralf Klasing
CNRS, University of Bordeaux
Bordeaux, France
ralf.klasing@labri.fr

Miroslaw Korzeniowski
Wroclaw University of Technology
Wroclaw, Poland
miroslaw.korzeniowski@pwr.wroc.pl

***Abstract*—In this work we show how to generate and rapidly mix uniform random graphs in a model where incoming and outgoing degrees of nodes are defined in advance. We show how to use a previous result on Dating Service working on top of any Distributed Hash Table so that a random graph is generated in logarithmic number of rounds and mixed so that two snapshots of the graph taken in logarithmic time distance are independent with high probability. We consider two models of graphs: directed graphs and undirected graphs where some nodes are behind firewalls. We consider a synchronous model of computation but show how to adapt it to a highly dynamic and asynchronous environment such as peer-to-peer networks.**

***Keywords*-peer-to-peer, distributed hash tables, heterogeneous P2P, random graph generation and mixing**

I. Introduction

Peer-to-Peer networks have proven their efficiency for storing and retrieving data at large scale. Networks based on Distributed Hash Tables (DHT) such as *Pastry* [23], *Tapestry* [14], *Chord* [24] or *CAN* [22] provide efficient routing mechanisms in time logarithmic in the number of nodes, but due to hashing, their use is limited to exact query searches. Moreover, none of the known algorithms takes heterogeneity of the network into account while large scale platforms exhibit a high level of heterogeneity in terms of processing and communication resources.

In this paper, we try to solve a few problems of communication in Peer-to-Peer networks. The first is how to optimally connect nodes in the network using knowledge about their incoming and outgoing degrees (taking heterogeneity of the network into account). The second one is how to use the solution to adapt it in both synchronous and non-synchronous network environments.

We describe distributed *dating service* and *mixing* algorithms that can be implemented over existing DHT systems. They are used to organize communication in a heterogeneous network, so that communication capabilities of nodes are not exceeded. The abstract purpose of this scheme is to randomly connect demands and supplies of some resource of many nodes into couples. In a round it produces a matching between demands and supplies which is of linear expected size (compared to the optimal one) and is chosen uniformly at random from all matchings of this size. The *dating service* algorithm is the only one, to the best of our knowledge, that can connect nodes taking into account their degree. It works in logarithmic time (expected and with high probability) and can also be run by many peers in a round yielding a linear number of connections distributed like balls thrown into n bins. The *mixing* algorithm is used here to improve the specific properties of built connection graphs. It works in logarithmic time as well so after logarithmic time we get the graph with properties compared to randomly chosen one (which is close to the optimal one) with high probability.

In the next sections, we describe both how the algorithms work with proof of their efficiency and correctness and the practical implementation with test run results. We have implemented the algorithms in two different environments:

- synchronous environment - where we show that the theoretical analysis of *dating service* and *mixing* algorithms is confirmed in practice,
- non-synchronous environment - where we implement suitably adapted *dating service* and *mixing* algorithms in a non-synchronous environment over a *Chord*-like DHT network layer, in order to verify how they perform in real networks where each node can join or leave the network and there is no centralized mechanism controlling the execution of the algorithm.

A. Our Results

We prove the algorithms' correctness and efficiency in a synchronous environment in both theory and practice. We show results of tests that have validated the logarithmic time complexity of the *dating service* algorithm and the efficiency of the *mixing* algorithm. They show that the *mixing* algorithm copes with fairly badly connected graphs, such as loops and cycles, very fast and makes their properties similar to random graph properties. The second result is an

implementation of these algorithms in a non-synchronous environment and a proof of their efficiency. We show that *dating service* and *mixing* algorithms make properties of connection graphs similar to expected values (random graph properties) very fast even if the network is static only for a short time. Thus, the simulations show short stabilization time after a node joins or leaves the network.

II. Related Work

The idea of using random processes to build overlay networks for P2P systems can be found in such fundamental designs as e.g. the Gnutella network [1] and JXTA of Sun Microsystems [4]. Important topological properties for P2P overlays include high connectivity, low degree, high expansion, and small graph distances between nodes. Random type topologies with small degree naturally arise in P2P systems, which are generated and maintained according to a Markov process as described in this paper (e.g. [20]). Several research groups have recently designed a variety of random like networks for P2P systems (e.g. WARP of [15]), and there is a considerable amount of work devoted to the generation and maintenance of random (regular) graphs (e.g. [6], [7], [11], [18], [21]).

One of the simplest randomized protocols was introduced in [9]. In this gossip-based process, each node exchanges messages with a small set of partners, chosen randomly from the set of all players. This process is highly scalable, robust against edge or node failures [10], and only requires a small amount of messsage transmissions [16].

Since modern P2P systems are built out of a huge number of peers, we cannot assume that all nodes know all the other players in the system. Therefore, plenty of recent papers concentrated on the design and analysis of gossip protocols, where each node is only aware of a small subset of peers in the system (e.g. [5], [11], [12], [17]).

There is a huge amount of work dealing with random processes for constructing and maintaining P2P overlays. Here, we only describe some recent papers in this field, which are closely related to our work. In [2], Allavena, Demers, and Hopcroft present a scalable gossip-based algorithm, in which each node has a (restricted) local view that is recalculated continously during the execution of the algorithm. The authors show that the expected time until the graph becomes disconnected is at least exponential in the square of the view size. Furthermore, they also develop probabilistic bounds on the in-degree of the nodes, and argue that the undirected connectivity is an expander.

In [7], Cooper, Klasing, and Radzik describe a randomized algorithm for assigning neighbours to vertices joining a dynamic distributed network. The aim of the algorithm is to maintain connectivity, low diameter and constant vertex degree. On joining each vertex donates a constant number of tokens to the network. These tokens contain the address of the donor vertex. The tokens make independent random walks in the network. A token can be used by any vertex it is visiting to establish a connection to the donor vertex. This allows joining vertices to be allocated a random set of neighbours although the overall vertex membership of the network is unknown. The network obtained in this way is robust under adversarial deletion of vertices and edges and actively reconnects itself.

Feder, Guetz, Mihail, and Saberi analyze in [11] the mixing time of a so called switch Markov chain on regular graphs, where switches are only allowed between edges being at distance 1. That is, two edges (i, j) and (k, l) are allowed to be replaced by (i, k) and (j, l) if i and l are adjacent to each other. This rule defines a move in this Markov chain (cf. [19]). The main statement of [11] is that this Markov chain is rapidly mixing, however, the exponent in the corresponding polynomial is quite high (≥ 45). This result has been improved in [5], where it is shown that the corresponding Markov chain has a mixing time of $O(d^{20}n^{14}(dn \log dn + \log \epsilon^{-1}))$ with d being the degree of the network and ϵ shows basically how well-mixed the Markov chain is.

Gurevich and Keidar consider gossip-based membership protocols on graphs in which nodes and edges can fail [13]. First, they define some basic properties, which should be fulfilled by membership protocols. Then, they propose a process, which satisfies these properties, if the graph is fully reliable. Finally, they analyze the protocol in the presence of edge/node failures, and show that the algorithm provides the desired properties of a membership service.

In [17] Kermarrec, Leroy, and Thraves analyze a generic peer sampling service that ensures for any peer small views, uniform sample, small in-degree, and independence. In fact, they show that the proposed protocol eventually results in a random simple connected graph with the same out-degree for any starting graph. Then, they use empirical evaluation to rigorously analyze their protocol.

III. Dating Service

The purpose of the dating service [3] is to produce random matchings between two types of requests which, for convenience, we name supplies and demands. The routine works in rounds and in each round all peers of the application send information about their supplies and demands to the dating service, their requests are joined into couples and the peers can use these connections somehow and proceed to the next round.

In our description of the dating service, nodes taking part in providing the service are called servers, to distinguish them from their role as users. Note that this difference between servers and users only stands to ease the description of the dating service. In practice, all the nodes will be used as servers and none of them are more important than others.

The centralized dating service based on a single server works as follows. All nodes of an application submit infor-

mation about all their supplies and demands to the server, the server chooses uniformly at random a maximum matching and sends information about connections to all nodes of the application. Producing a matching chosen uniformly at random is actually trivial in this setting: it is sufficient to permute uniformly at random both sequences of supplies and demands and match the i-th supply with the i-th demand for all i between 1 and $m = \min\{\#supplies, \#demands\}$.

In the distributed setting, where the whole set of nodes is used to produce the random matching, the service works as follows. Each node sends each request, either demand or supply to a random server chosen according to a fixed probability distribution $P = (p_1, p_2, ...p_n)$. Then, the server is responsible to build a local random matching between the demands and supplies it has received.

In the above description, it is not necessary that the random choice of servers be uniform only that all requests are sent using the same distribution P. This randomness is a load-balancing factor; as an extreme case, sending all requests to a single server would result in a centralized scheme.

Theorem 1. *[3] Let $X_1, X_2, \ldots, X_n$ be random variables denoting the number of dates organized by servers $1, \ldots, n$. Let $X = X_1 + \ldots + X_n$. Then $X = \Omega(m)$, with high probability.*

IV. Directed Random Graphs

In this section, we present and analyze an algorithm, RANDOM-DIGRAPH, for the following problem. We are given n nodes in a distributed network, each of them has a demand for its incoming degree and outgoing degree. The former for node i is denoted as $\mathrm{b}^{in}(i)$ and the latter as $\mathrm{b}^{out}(i)$. We denote the total demand for incoming degrees as $\mathrm{B}^{in} = \sum_i \mathrm{b}^{in}(i)$ and similarly for outgoing degrees $\mathrm{B}^{out} = \sum_i \mathrm{b}^{out}(i)$. All nodes can communicate freely, i.e. they can for example build a Distributed Hash Table. Our goal is to construct quickly in a distributed way a set of connections among nodes so that:

1) the in-degree of node i is at most $\mathrm{b}^{in}(i)$,
2) the out-degree of node i is at most $\mathrm{b}^{out}(i)$,
3) the total number of edges constructed in the graph is maximal, that is equal to $\min\{\mathrm{B}^{in}, \mathrm{B}^{out}\}$,
4) the constructed graph is uniformly chosen from all graphs fulfilling conditions 1–3.

Our algorithm works in rounds and in each round of random graph construction one round of the dating service algorithm is executed and some number of edges are constructed. We denote the current (in round t) degrees of node i in such a partial graph as $\deg_t^{in}(i)$ and $\deg_t^{out}(i)$. Then, node i in round t (still) participates in the dating service DHT if $\deg_t^{in}(i) < \mathrm{b}^{in}(i)$ or $\deg_t^{out}(i) < \mathrm{b}^{out}(i)$, i.e. if it still needs some incoming or outgoing edges. Each node i submits to the dating service $\mathrm{b}^{in}(i) - \deg_t^{in}(i)$ requests for incoming edges and $\mathrm{b}^{out}(i) - \deg_t^{out}(i)$ requests for outgoing edges. The dating service produces a number of edges linear in the total number of submissions, some nodes get new incoming edges, some nodes get new outgoing edges, they update their incoming and outgoing degrees and proceed to the next round.

The scheme is meant to run continously in a dynamic system but if it were meant to run once, one might also want to include an explicit termination mechanism. In this case we introduce two additional DHTs. In the first one, only nodes with $\deg_t^{in}(i) < \mathrm{b}^{in}(i)$ participate, in the second one only those with $\deg_t^{out}(i) < \mathrm{b}^{out}(i)$. Each node is able to verify quickly if any of these DHTs is empty. If this is the case, it means that the total number of already constructed edges is $\min\{\mathrm{B}^{in}, \mathrm{B}^{out}\}$, i.e. that the algorithm has finished building a graph.

We first prove that the algorithm finishes in a logarithmic number of rounds (in n, B^{in}, and B^{out}).

Lemma 2. *After T rounds of the RANDOM-DIGRAPH algorithm are executed, $\min\{\mathrm{B}^{in}, \mathrm{B}^{out}\}$ edges are constructed with high probability if $T = \Omega(\log n + \log \min\{\mathrm{B}^{in}, \mathrm{B}^{out}\})$.*

Proof: If $\min\{\mathrm{B}^{in}, \mathrm{B}^{out}\} = \mathrm{B}^{in}$ then in each round a request for an incoming edge is fulfilled with constant probability. If $\min\{\mathrm{B}^{in}, \mathrm{B}^{out}\} = \mathrm{B}^{out}$ then in each round a request for an outgoing edge is fulfilled with constant probability. In both cases, after $\mathrm{O}(\log \min\{\mathrm{B}^{in}, \mathrm{B}^{out}\})$ rounds such a request is still not fulfilled with probability at most $\frac{1}{\min\{\mathrm{B}^{in}, \mathrm{B}^{out}\}}$ and after additional $\mathrm{O}(\log n)$ rounds with probability $\frac{1}{n^c \cdot \min\{\mathrm{B}^{in}, \mathrm{B}^{out}\}}$ for a chosen constant c. Summing over all $\min\{\mathrm{B}^{in}, \mathrm{B}^{out}\}$ requests of the rarer type, we get the probability of existence of a non-fulfilled request being at most $\frac{1}{n^c}$ ∎

The property that the graph is chosen uniformly at random comes from the fact that the dating service produces matchings chosen uniformly at random. It is formalized in the following lemma.

Lemma 3. *The graph of connections produced by the RANDOM-DIGRAPH algorithm is chosen uniformly at random from the set of all graphs fulfilling the defined properties.*

Proof: It follows from symmetry of results produced by the Dating Service. If we consider any two outgoing links l_i and l_j (they may be links outgoing from two different or just one node; each of them may have a connection or not, the latter may happen in case when $\mathrm{B}^{out} < \mathrm{B}^{in}$), and repeat the whole process with the same random choices but with exchanged roles of l_i and l_j, then the resulting graph will be the same except the other ends of l_i and l_j exchanged. This reasoning extended to any permutation on all outgoing links proves that the resulting graph is symmetric. ∎

V. Undirected Random Graphs with Firewalls

In this section we consider the following scenario. Participants of the network declare their generic (equal in and out) bandwidth. However, some of them are hidden behind firewalls, which means that they can only participate in connections initiated by themselves. We are interested in building a random graph in such a model such that the number of edges is maximized. Notice that in two extreme cases we may have to build a standard random graph (when there are no firewall-constrained nodes) and a bipartite random graph (when the number of firewall-constrained nodes equals the number of all nodes).

We are given two sets of nodes: n_u *universal* nodes forming a set $\mathcal{U}$ and n_b *bounded nodes* forming a set $\mathcal{B}$; the total number of nodes is denoted by $n = n_u + n_b$. Each of the n nodes has a parameter $\mathrm{b}(i)$ denoting its bandwidth, i.e. the degree it would like to have. We extend the bandwidth notation to sets of nodes, i.e. $\mathrm{b}(\mathcal{U}) = \sum_{i \in \mathcal{U}} \mathrm{b}(i)$ and $\mathrm{b}(\mathcal{B}) = \sum_{i \in \mathcal{B}} \mathrm{b}(i)$.

Our goal is to construct in a distributed way a set of connections among nodes so that:

1) the degree of node i is at most $\mathrm{b}(i)$,
2) a node from $\mathcal{B}$ may be either disconnected or connected to a node from $\mathcal{U}$,
3) the total number of edges constructed in the graph is maximal,
4) the constructed graph is chosen uniformly at random from all graphs fulfilling conditions 1–3.

Additionally, in the process of creating the graph no two nodes from $\mathcal{B}$ can communicate to each other.

To solve this problem we propose two dating service schemes working in parallel. One of them we denote with $D_{\mathcal{U}}$ and the other with $D_{\mathcal{B}}$. The former is responsible for creating connections between nodes from $\mathcal{U}$ and the latter for connections between nodes from $\mathcal{U}$ and $\mathcal{B}$. They work in parallel in rounds and in each round each node submits some information to both of them. Naturally the DHTs being the base of the Dating Service can only by run using nodes from $\mathcal{U}$.

The first Dating Service scheme is a degenerated one in this sense, that only one type of information is submitted to it and instead of working on two sets of submitted requests, it works on just one and produces as many pairs as possible, i.e. if it receives r requests, it produces a matching of size $\lfloor r/2 \rfloor$ chosen uniformly at random. This behavior of Dating Service preserves the symmetry of the original Dating Service and its efficiency (the size of the produced matching) may only be higher.

The second Dating Service is a standard one, i.e. it receives two types of requests: one type from universal nodes and another type from bounded nodes. It connects such requests into couples in a standard way. Bounded nodes send requests for their yet unfulfilled outgoing connections in a standard way but universal nodes send requests for two types of their outgoing links: those which are still unfulfilled and those which are connected to another universal node.

The symmetry property of the produced solution is the same as in the previous model. We concentrate on the running time, i.e., the number of rounds until the maximal number of edges is generated. Even though both schemes run in parallel, in the analysis we start considering the running time of the first Dating Service only after the second one has stabilized (all bounded nodes have fulfilled all their requested connections).

Lemma 4. *With high probability the two Dating Service schemes produce the maximum number of edges in $O\left(\log n + \mathrm{b}(\mathcal{U}) + \mathrm{b}(\mathcal{B})\right)$ rounds. The resulting graph is chosen uniformly at random from all graphs fulfilling the desired properties.*

Proof: As the second dating service scheme is standard and has priority over the first one, according to Lemma 2 $O\left(\log n + \mathrm{b}(\mathcal{U}) + \mathrm{b}(\mathcal{B})\right)$ rounds are sufficient to fulfill connections of either all nodes behind firewalls or all free nodes. In the latter case the procedure is finished as it is not possible to create more connections. In the former, we can again use Lemma 2, as the degenerated dating service is at least as fast as the normal one. With high probability $O\left(\log n + \mathrm{b}(\mathcal{U})\right)$ additional rounds suffice to fulfill all unfulfilled edges of free nodes. ∎

VI. Mixing

Generating a random graph once has its drawbacks. First of all, random graphs have some desired properties with high probability but it still may happen that the graph we generated is not optimal. If we use it for example for network construction and we use the same graph for the whole lifetime of our network, we want to have the possibility to change the graph if it has any bad properties.

Also, in a dynamic system, if we do not specifically incorporate into our algorithms mixing of old nodes with new ones, we may end up in a situation where we create many separate random graphs whose structure reflects which nodes joined at which time.

The first solution that comes to mind is to generate a random graph, use it for some time and in the background of main computations generate a new version. As soon as the new version is ready, we replace the old one with it. This solution has a few drawbacks too: changes between versions are rapid (practically no similarities) and may be costly, they are also hard to synchronize.

What has been proposed in the literature is to mix the graph, i.e. exchange connections among nodes so that in a single round there are not too many of them (e.g. each edge is replaced with a small constant probability) but on the other hand after some number of rounds (usually logarithmic in the number of nodes) the starting and resulting graphs are

independently chosen random graphs with high probability. Here, Dating Service helps also together with a result by Czumaj and Kutylowski [8].

Theorem 5 (Czumaj and Kutylowski)**.** *For an n-element permutation S, if an algorithm A in each round i generates a matching M_i such that $|M_i| = \Theta(n)$ and M_i is chosen uniformly at random from all matchings of size $|M_i|$, elements joined by edges of M_i are exchanged with probability $1/2$, then after $O(\log n)$ rounds S is a uniform random permutation with high probability, i.e. the algorithm A is rapidly mixing.*

The above paper considers the problem of permutation routing. The authors show that if for n elements in each round we are able to generate a matching of size $\Omega(n)$ which is chosen uniformly at random from all matchings of this size, then by exchanging positions of elements connected by edges of this matching in each round, we generate a uniform random permutation within $O(\log n)$ rounds, with high probability. Together with the Dating Service scheme it allows for mixing graphs so that starting in an arbitrary situation, it generates a uniformly chosen random graph independent of this situation within $O(\log n)$ rounds.

A. Mixing Directed Random Graphs

For the case of directed random graphs, we proceed as follows. To the whole scheme we add two additional Dating Service structures, both of them degenerated (i.e. serving just one type of items instead of two) in which all peers participate all the time. To the first of these structures in each round each node submits its incoming connections, whether they are currently connected or loose. Similarly, to the second structure, a peer submits all its outgoing connections. It would be sufficient to use just one of these structures — the one responsible for a higher number of connections (incoming if $\mathrm{B}^{in} > \mathrm{B}^{out}$ or outgoing otherwise) but it is impossible to decide this reliably in a distributed way and also the situation may change, since we assume the system is dynamic.

Based on Theorem 5 we conclude that from the point of view of the more abundant type of connections, the other ends and missing connections are with high probability permuted in a logarithmic number of rounds and we come to the following theorem.

Theorem 6. *The described algorithm for generating directed graphs generates a maximal graph (i.e. such that no more edges can be added) after logarithmic number of rounds. The mixing algorithm generates an instance of a directed graph independent of the instance from $\Theta(\log n)$ steps before, with high probability.*

B. Mixing Undirected Random Graphs with Firewalls

In case of our second problem, i.e. undirected random graphs where some nodes may be behind firewalls, we proceed similarly concerning mixing. We have two degenerated dating service structures but as in the process of building the graph, only nodes not behind firewalls take part in building the service itself. One service is used to mix edges of nodes behind firewalls and the other for nodes not behind firewalls. Again, based on Theorem 5 we conclude that within logarithmic number of rounds, the graphs is a new random instance of a graph fulfilling the desired properties, independent of the initial graph with high probability.

Theorem 7. *The described algorithm for generating undirected graphs with firewalls generates a maximal graph (i.e. such that no more edges can be added) after logarithmic number of rounds. The mixing algorithm generates an instance of an undirected graph with firewalls independent of the instance from $\Theta(\log n)$ steps before, with high probability.*

VII. Non-Synchronous Model and Experimental Setting

A. Graph Properties

As in a large-scale dynamic environment such as Peer-to-Peer networks it is virtually impossible to proceed in synchronized rounds, we need to adapt our algorithms and develop their not so synchronous versions. Unfortunately, the proofs given for a synchronous execution are no longer valid. Instead we simulate the execution of our algorithms and check how they behave in a randomly modeled dynamic system. Naturally, it is also impossible to validate that a generated graph is actually randomly chosen. For this purpose we use well-known techniques of checking specific graph properties of generated graphs:

1) Connectivity coefficient
2) Average path length
3) Average clustering coefficient

The first of them, the connectivity coefficient, expresses the number of pairs of nodes in the generated graph, which are directly or indirectly connected. The second one, the average path length, expressess the expected length of the shortest path between any two connected nodes. Finally, the last property, the average clustering coefficient, stands for average over all nodes of the ratio of the direct neighborhood of a node to its neighborhood of radius 2 (i.e. it shows how well neighbors of a random node are connected among themselves).

B. Non-synchronous Algorithm Concept

In this section we describe the concept of applying synchronous dating service and mixing algorithms to non-synchronous environment. We assume that there are no global rounds, each node performs its own routine of sending demands and supplies for dating service and mixing algorithms and produces a random matching of incoming resources simultaneously.

1) Underlying network layer: As a DHT based network layer we use a construction similar to *Chord* [24] here. A very important condition is that the network is faultless. Thus, each node is obliged to inform its neighbors that it is leaving the network, it cannot just disappear. The used underlying network layer is based on a ring in which each node is placed according to its unique 160-bits random identifier $id \in [0, 1)$. We use TCP protocol to let nodes communicate with one another. We assume that there is at least one stable (it cannot leave) known "bootstrap" node in the network so each node knows its IP address and can use it to join the network. Each node keeps a connection to its successor, predecessor and a few nodes $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and $\frac{1}{16}$ away from it in both forwards and backwards directions.

2) Dating service and mixing algorithm routines: As it is not a synchronous implementation, each node runs algorithm routines independently of other nodes and there are no rounds. Furthermore, each node sends its own demands and supplies for the *dating service* and *mixing* algorithms independently of doing *dating service* and *mixing* algorithms with incoming demands and supplies. Thus, each node executes two algorithms simultaneously:

- sending demands and supplies for *dating service* and *mixing* algorithms,
- doing *dating service* and *mixing* algorithms with incoming demands and supplies.

The first algorithm shown in Figure 1 is based on two states: sending resources for dating service algorithm and sending resources for mixing algorithm. It has two parameters: P and T where the former is the probability distribution over all participating peers resulting from the built DHT and the latter is a time threshold parameter which should be larger than the maximal routing time in the built DHT.

Each peer also works as a server and gathers supplies and demands from other nodes in both dating service and mixing routines. We consider two modes of operating in this role: *mode 1* means that within each period of time of length T supplies and demands are gathered and then coupled randomly in a standard way; *mode 2* means that each time a serving node receives a resource, it starts waiting for a bounded time T and connects this resource to another one if it arrives within this time or releases this resource without a connection after this period.

3) Statistics monitoring: Apart from the algorithm routines, we have to collect statistics about the produced connection graphs to determine their properties such as:

- Connectivity coefficient
- Average path length
- Average clustering coefficient

To realize this functionality, we run the *monitoring process* which is responsible for collecting information about the connections of each demand and supply of each node. Its address is known to each node in the network and each node has to keep connection to the *monitoring process* (independently of the connections to other nodes and using a separate protocol). The *monitoring process* performs a certain number of checks at certain intervals. A single check is shown in Figure 2. After some number of checks we have information about the connections of each demand and supply of each node after each statistic check so we can use the algorithms described in Section VII-A in order to determine the graph properties described above.

Figure 1. Sending demands and supplies by the single node algorithm in non-synchronous environment (probability distribution $P = (p_1, ..., p_n)$)

$state = DATING\_SERVICE\_ALGORITHM$
while working **do**
  if $state = DATING\_SERVICE\_ALGORITHM$ **then**
    for each unconnected demand and supply **do**
      send a resource to a node chosen according to P
    end for
    wait for all sent resources to come back
    $state = MIXING\_ALGORITHM$ after time T
  else if $state = MIXING\_ALGORITHM$ **then**
    for each supply **do**
      send a supply to a node chosen according to P
    end for
    wait for all sent supplies to come back
    $state = DATING\_SERVICE\_ALGORITHM$
    after time T
  end if
end while

Figure 2. Monitoring process. Single statistics check.

for each connected node **do**
  send $STATS\_LOCK$ message
end for
wait for all nodes to take lock confirmation
for each locked node **do**
  send GET_STATS message
end for
wait for all nodes' statistics
save demand and supply of each node connections information
for each connected node **do**
  send $STATS\_UNLOCK$ message
end for

VIII. Experimental Results

In this section, we show how the described algorithms work in a non-synchronous environment. We show the way they were implemented and the results of the performed tests as well.

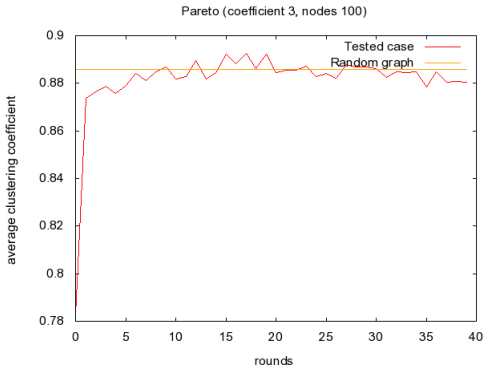

Figure 3. Average clustering coefficient, pareto distribution, mode 1, static model

A. Environment Implementation

The testing environment consists of a few scripts used to parse output data and draw charts (written in Perl scripting language) and two types of applications:

- a single node application written in *C++*, that is responsible for single node routines (e.g. *dating service* and *mixing* algorithms)
- a monitoring application written in *C++*, that is responsible for collecting connections information between demands and supplies of all nodes to get connections graphs of all nodes (they are used to calculate certain graph properties after the tests end).

The implementation of the single node application consists of two layers:

- a *Chord*-like DHT based transport layer, that manages node insertion and departure and network messages routing,
- an application logic layer, that executes *dating service* and *mixing* algorithms.

They communicate with each other, so the logic layer gets messages from the transport layer and it can call interface methods of the transport layer. The logic layer decides when the transport layer can leave the network, so it cannot leave without a specific action from the logic layer. For simplicity we assumed in the transport layer implementation that the network is faultless so there are no broken connections. This gives us control over node insertion, departure and communication. We do not lose messages being routed and what is more important we do not have to implement any structure recovery algorithms (this becomes necessary if we assume that any node can lose the connection to its successor or predecessor). The faultless network is ensured by using TCP network protocols for connections and communication. Moreover, we run all the node applications on a single machine. Thus, we can assume here that one node can leave the network only if it is not waiting for any demand and supply connection information (after sending them for

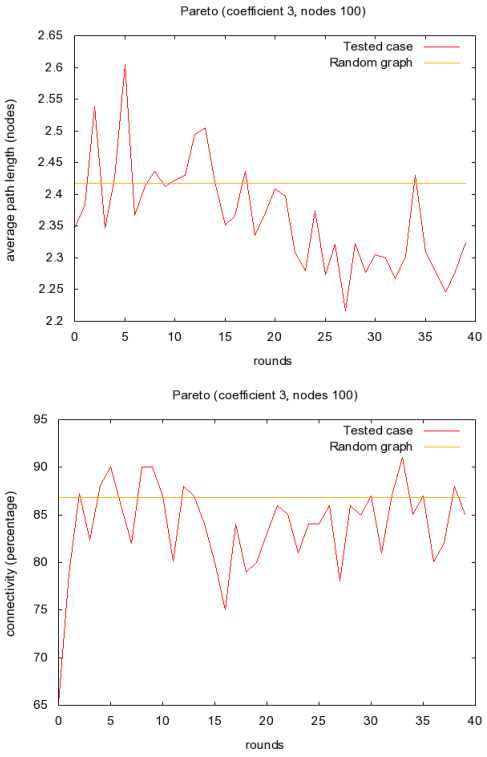

Figure 4. Average path length and connectivity coefficient, pareto distribution, mode 1, static model

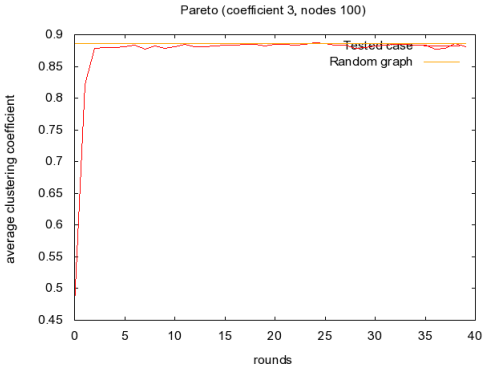

Figure 5. Average clustering coefficient, pareto distribution, mode 2, static model

the *dating service* or *mixing* algorithms) and if it has no messages to route to other nodes. The next section shows simulation results including the static model, where we have a certain number of nodes in the network and none of them can leave, and the dynamic model, where each node can join the network and stays in the network a random (according to a uniform distribution) number of seconds.

B. Simulations

We have run tens of tests for the static and the dynamic model and for both modes of doing *dating service* and

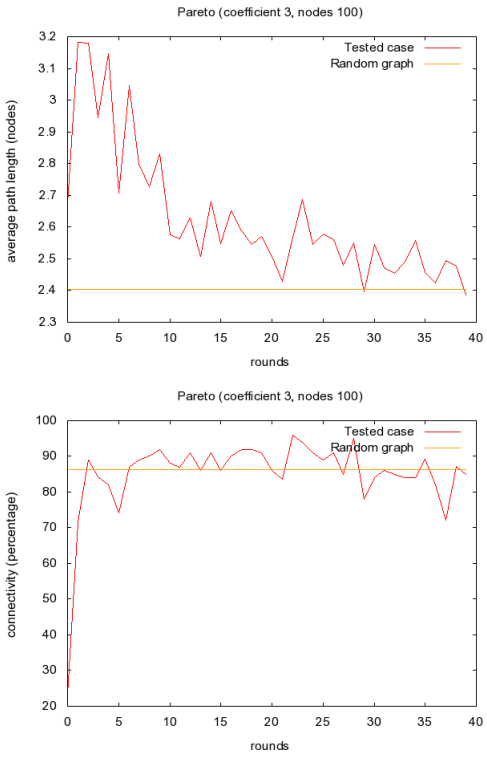

Figure 6. Average path length and connectivity coefficient, pareto distribution, mode 2, static model

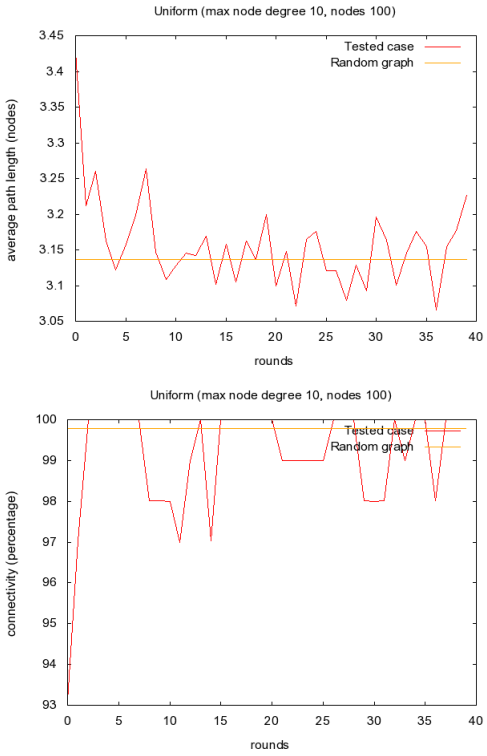

Figure 7. Average path length and connectivity coefficient, uniform distribution, mode 1, static model

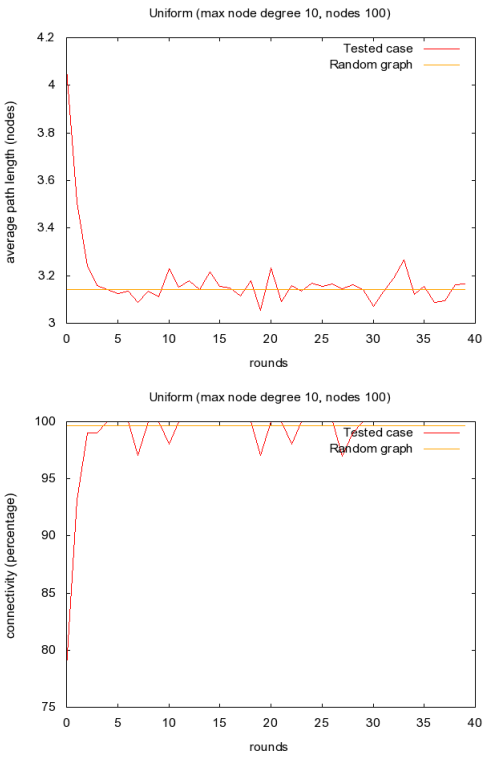

Figure 8. Average path length and connectivity coefficient, uniform distribution, mode 2, static model

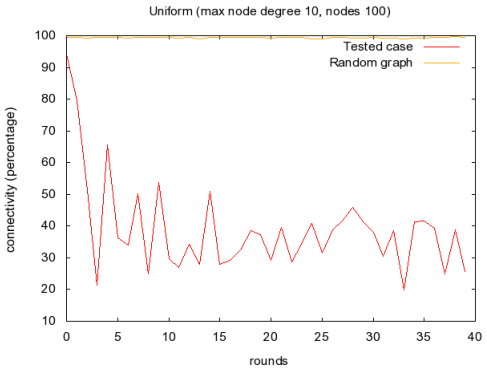

Figure 9. Connectivity coefficient, uniform distribution, mode 1, dynamic model

mixing algorithms with incoming demands and supplies. We tested cases where the number of seconds to join the network is in $[5, 15]$ and the number of seconds of staying in the network is in $[20, 120]$ which generates fairly dynamic networks. We have assumed the number of seconds between states of sending resources for the *dating service* and *mixing* algorithms to be 5 seconds and the number of seconds to wait for other demands and supplies to do *dating service* and *mixing* to be 10 seconds. The input data for certain cases has been generated randomly using

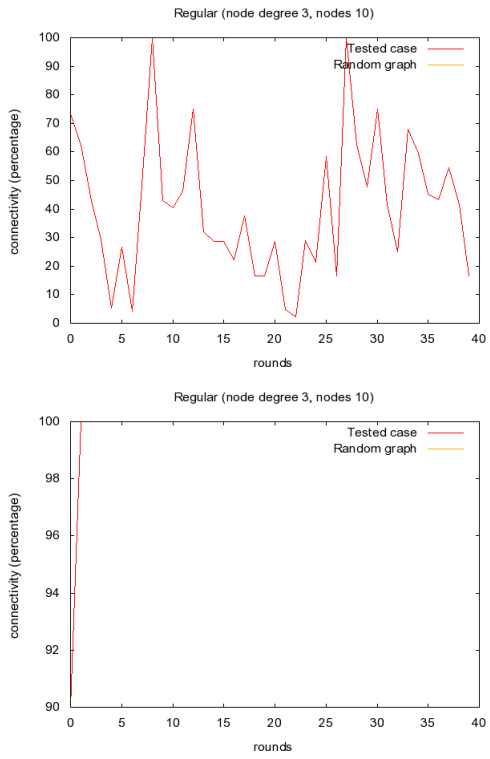

Figure 10. Connectivity coefficient comparison of dynamic (first graphs) and static (second graph) models, regular graph, mode 1

different distributions according to nodes degrees (number of demands and supplies). The monitoring application does 40 statistic checks (called rounds here) at 20 seconds intervals. We tested cases with 10 and 100 nodes. In the figures, we denote by *mode 1* the first mode described in Section VII-B and by *mode 2* the second mode described in Section VII-B.

1) Pareto Distribution: The input data has been generated randomly using *Pareto* distribution for $p = 2$. The generated graph properties in successive checks of the monitoring application are shown in figures from Figure 3 to Figure 6. Comparing Figure 4 and Figure 6 we can see that both mode 1 and mode 2 lead to almost the same results but mode 1 is much faster while looking at the average path length of the generated graphs.

2) Uniform Distribution: The input data has been generated randomly using a uniform distribution where the number of demands and supplies were at most 10. The generated graph properties in successive checks of the monitoring application are shown in figures from Figure 7 to Figure 8. As we can see, in this case in mode 1 and mode 2 the results and stabilization time are almost equal. Figure 9 shows the graph properties in the presented dynamic model. We can see that when nodes are joining and leaving the network very often the graph properties are not satisfied.

3) Uniform Distribution (equal total number of demands and supplies): The input data has been generated randomly using a uniform distribution where the number of demands and supplies were at most 10. The further restriction has been added that the total number of demands and the total number of supplies have to be equal. We do not include figures for this case as the results are very similar to the case with standard uniform distribution.

4) Regular Graphs: In this case our tests have coped with data where each node has the same number of demands and supplies and it is equal to some constant k. We have executed tests for $k \in [1, 3]$. In Figure 10, we compare dynamic and static models with identical configurations.

References

[1] Gnutella. the gnutella protocol specification v.0.4.

[2] André Allavena, Alan Demers, and John E. Hopcroft. Correctness of a gossip based membership protocol. In *Proc. of ACM PODC '05*, pages 292–301, 2005.

[3] Olivier Beaumont, Philippe Duchon, and Miroslaw Korzeniowski. Heterogenous dating service with application to rumor spreading. In *Proc. of IPDPS '08*, pages 1–10, 2008.

[4] Sherif M. Botros and Steve R. Waterhouse. Search in JXTA and other distributed networks. In *Proc. of P2P '01*, pages 30–35, 2001.

[5] Colin Cooper, Martin Dyer, and Andrew J. Handley. The flip markov chain and a randomising P2P protocol. In *Proc. of ACM PODC '09*, pages 141–150, 2009.

[6] Colin Cooper, Martin E. Dyer, and Catherine S. Greenhill. Sampling regular graphs and a peer-to-peer network. *Combinatorics, Probability & Computing*, 16(4):557–593, 2007.

[7] Colin Cooper, Ralf Klasing, and Tomasz Radzik. A randomized algorithm for the joining protocol in dynamic distributed networks. *Theor. Comput. Sci.*, 406(3):248–262, 2008.

[8] Artur Czumaj and Miroslaw Kutylowski. Delayed path coupling and generating random permutations. *Random Struct. Algorithms*, 17(3-4):238–259, 2000.

[9] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. of ACM PODC '87*, pages 1–12, 1987.

[10] Robert Elsässer and Thomas Sauerwald. On the runtime and robustness of randomized broadcasting. In *Proc. of ISAAC '06*, pages 349–358, 2006.

[11] Tomás Feder, Adam Guetz, Milena Mihail, and Amin Saberi. A local switch markov chain on given degree graphs with application in connectivity of peer-to-peer networks. In *Proc. of FOCS '06*, pages 69–76, 2006.

[12] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2):139–149, 2003.

[13] Maxim Gurevich and Idit Keidar. Correctness of gossip-based membership under message loss. In *Proc. of ACM PODC '09*, pages 151–160, 2009.

[14] Kirsten Hildrum, John D. Kubiatowicz, Satish Rao, and Ben Y. Zhao. Distributed Object Location in a Dynamic Network. In *Proc. of the 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 41–52, 2002.

[15] Suresh Jagannathan, Gopal Pandurangan, and Siriam Srinivasan. Query protocols for highly resilient peer-to-peer networks. In *Proc. of ISCA PDCS '06*, pages 247–252, 2006.

[16] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vocking. Randomized rumor spreading. In *IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[17] Anne-Marie Kermarrec, Vincent Leroy, and Christopher Thraves. Converging quickly to independent uniform random topologies. In *Proc. of PDP '11*, pages 159–166, 2011.

[18] Ching Law and Kai-Yeung Siu. Distributed construction of random expander networks. In *Proceedings of the 22nd INFOCOM*, pages 2133–2143, 2003.

[19] Peter Mahlmann and Christian Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. In *Proc. of SPAA '05*, pages 155–164, 2005.

[20] Peter Mahlmann and Christian Schindelhauer. Distributed random digraph transformations for peer-to-peer networks. In *Proc. of SPAA '06*, pages 308–317, 2006.

[21] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter peer-to-peer networks. In *Proc. of FOCS '01*, pages 492–499, 2001.

[22] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A Scalable Content Addressable Network. In *Proc. of the ACM SIGCOMM*, pages 161–172, 2001.

[23] Antony Rowstron and Peter Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of the 3rd IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[24] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. of the ACM SIGCOMM*, pages 149–160, 2001.