A Multicast-based Bootstrap Mechanism for Self-organizing P2P Networks

Simone Cirani and Luca Veltri
Department of Information Engineering
University of Parma
Viale G. P. Usberti 181/A, 43100 Parma - Italy
Email: simone.cirani@tlc.unipr.it, luca.veltri@unipr.it

***Abstract*—The peer-to-peer (P2P) network paradigm has been introduced in order to overcome some shortcomings of the client-server architecture by providing such features as decentralization, self-organization, scalability, and fault-tolerance. Bootstrapping is the initial process through which new nodes can join an existing P2P overlay network. Typically, a joining peer must first contact a bootstrap peer, which is a peer already enrolled in the overlay. The bootstrap peer is responsible for admitting the new peer by passing information about other peers so that the new peer can actively participate in the overlay. Finding a suitable bootstrap peer is therefore a critical issue. Although different P2P systems have been defined and deployed, the problem of bootstrapping has usually been solved by introducing such mechanisms as the use of a pre-configured list of nodes, caching, or server-based discovery. Unfortunately, although they work in P2P applications running over the Internet, they show some problems when applied to very dynamic and self-organizing intranet or enterprise network scenarios. In fact, in these cases all nodes may join and leave the network very dynamically, without the possibility of guaranteeing any sort of permanent centralized service as current bootstrap solutions may require. In this paper, we propose a multicast-based bootstrapping mechanism for dynamic and self-organized P2P networks that allows a joining peer to discover a proper bootstrap peer in a real distributed manner. The proposed mechanism uses an unsolicited approach and performs well in terms of scalability, load-balancing, and a mean frequency of information exchange. The paper defines the algorithm and proposes an implementation of a suitable communication protocol.**

I. Introduction

Peer-to-peer (P2P) overlay networks are used in those scenarios where decentralization, self-organization, and fault-tolerance are desired. However, P2P systems are never fully distributed as they typically rely on some centralized network elements or prior knowledge for bootstrapping, that is, to let new nodes join the overlay. Therefore, actual decentralization and self-organization cannot be achieved. Although this is not a problem in current P2P applications over the Internet where some (super-)nodes can be considered as permanently connected and sufficiently reliable, this becomes a problem when applied to very dynamic and self-organizing intranet or enterprise networks where all nodes may have a very dynamic behavior, leading to the impossibility to guarantee any sort of reliable and centralize bootstrap service.

Generally, a peer which is willing to join any P2P network needs to discover the location of a bootstrap peer to send its join request to. Current solutions include the use of:

- cached mechanisms - a peer maintains a list of previously discovered peers and tries to contact them; this approach does not solve however the problem when the peer is trying to join the overlay for the first time as its cached list would be empty;
- server-based mechanisms - a peer contacts a pre-configured server node (or list of nodes); this approach has the obvious disadvantage of being server-centric, which is an antithetical solution for the goal of a purely distributed network and would create a bottleneck in the network and a possible point of failure; it is important to say that the failure of the bootstrap server does not affect the behavior of the P2P overlay, but only prevents new peers from joining the network;
- multicast-based mechanisms - multicast support is exploited as proposed in this paper.

The first two approaches have drawbacks. For instance, it might not always be possible to know in advance a list of nodes that are always active to be used for bootstrapping, caching does not work in case the new node is joining the overlay for the very first time or if cached nodes are no longer enrolled in the overlay, and server-based mechanism might be useless if the server is unreachable. The use of mechanisms that combine pre-configured lists which are hardcoded into the protocol and caching has proved to work in practice (i.e. eMule bootstrapping), but it is still potentially exposed to the risk of failure. Moreover, such mechanisms do not work in all those scenarios in which the P2P networks are built in a complete distributed and self-organized manner (for example in case of server-free distributed enterprise networks or ad-hoc networks). This paper describes a new mechanism for discovering a bootstrap node in a P2P network. The mechanism, named BANANAS (BootstrAp Node NotificAtion Service), is based on multicast communications, and provides a completely distributed, self-organizing and scalable discovery service. Although the use of IPv4 multicast is currently not supported amongst the public Internet, it is implemented in several ISPs, private, or enterprise networks and it is expected in the future to be supported within more and more IPv4 and IPv6 networks.

The paper is organized as follows: in section II related work is reviewed. In section III, the solicited and unsolicited service approaches are discussed. In section IV we present a simple algorithm for the bootstrap service and explain some of its inefficiencies. In section V, an enhanced version of the algorithm is presented in order to overcome the limitations of the simple algorithm shown in section IV. Section VI proposes a simple implementation that we have realized through a suitable protocol. Finally, in section VII we report our conclusions.

II. Related Work

Cramer et al. [1] have discussed some possible mechanisms for the bootstrapping process, such as based on: static bootstrap servers, dynamic web caches, random access probing, multicast, or IPv6 anycast. The first two mechanisms suffer of the well known centralization, low reliability, and non-self-organization problems; while the random access probing, that consists in trying several random entry points until a success is reached, may result in large number of failures and large amount of network traffic. Anycast is also considered as mechanism for acquiring an IP address of a potential bootstrap peer, however it is also pointed out that such mechanism just moves the problem of node selection at the network layer, and has as drawback that may limit the requesting nodes' freedom of choice. The authors also considered multicasting as possible mechanism combined with the expanding ring search (ERS). ERS in turn works by searching successively larger areas in the network centred around the source of broadcast. Searching areas may be limited by using increasing values of TTL (Time To Live). However this approach has some limitations such as:

- TTL scoping requires "successive containment" property and will not work with overlapping regions;
- by increasing the TTL, the multicast scope may rapidly expand to a large portion of the entire network resulting in flooding query packets to a very large number of nodes;
- for each TTL value, a proper maximum round-trip time (RTT) has to be considered (measured or pre-configured).

The authors in [2] propose a bootstrap service based on random access probing. The bootstrap service relies on a separate, dedicated, and unique P2P bootstrap overlay where bootstrapping information are stored. The bootstrap overlay is used in order to exploit random access probing, which proved to be more efficient in large P2P networks, and can be accessed through two basic methods (*lookup()* and *publish()*). The bootstrap overlay is based on a Distributed Hash Table (DHT) to achieve load balancing among the participating nodes. However, the bootstrap service still suffers some drawbacks:

- it simply shifts the problem of joining the P2P overlay network to that of joining the P2P bootstrap overlay;
- it is based on DHTs, which may suffer of some security issues, such as poisoning or Sybil attacks;
- it relies on a PULL approach, that is, joining nodes issue a request and receive a response with bootstrapping information; even though a load balancing effort has been made in order to avoid overloading nodes responsible for the key of a popular overlay, each node possibly needs to handle an unpredictable number of requests.

Because of these reasons, we propose a different mechanism which is PUSH-based, that is, the joining node does not issue any request and bootstrap information are notified by the service. Moreover, our approach does not require any information storage system such as DHTs in order to keep bootstrapping information since each node simply notifies its own presence, thus avoiding the risk of overloading nodes.

III. Solicited vs. Unsolicited Approach

In this section, we will briefly discuss about the implication of the usage of the PUSH and PULL approaches in a multicast-based service. Let us consider the case in which all peers that act as bootstrap nodes or that directly know one or more bootstrap nodes are enrolled in a multicast group. There are two possible approaches for the discovery of a bootstrap peer in a multicast fashion.

A. Solicited approach

A peer which tries to join the overlay sends one multicast request to all the nodes in the group asking for bootstrap nodes; the nodes that would respond to such request would all be candidates to admit the peer. However, such a solicited mechanism would cause an overload of the network, especially when many bootstrap nodes are already in the overlay because all notification responses are sent for each joining node. Note that, amongst all response messages, only one is used by the requesting peer since only one bootstrap peer is needed to join the overlay. Moreover, also limiting the total number of responses from bootstrap nodes does not limit the total amount of messages spread over the network since it strictly depends on the total joining rate, multiplied for the cardinality of the multicast group.

B. Unsolicited approach

All the bootstrap nodes in the multicast group send unsolicited messages to all nodes in the group to advertise their presence in the overlay as well as their bootstrap information. When a node needs to discover a bootstrap node, it simply joins the multicast group and listens for these messages. This approach potentially may still have scalability issues due to the large number of messages sent over the network. However in this case, differently from the previous approach, such total amount of sent messages may be limited, regardless of the actual joining rate. This can be achieved if the nodes cooperate in order to ensure that the total amount of messages sent over the multicast group is constant or upper bounded, regardless of the effective number of collaborating bootstrap nodes. This second approach is the one followed in this work and appears to be the most efficient for the reasons reported above.

IV. Simple Algorithm

The goals of the service are to provide a service characterized by an almost constant rate of messages received and to fairly balance the number of messages sent by nodes. Both bootstrap and joining nodes join the same multicast group. Each node sees a timeline divided into slots of length T, where $f = \frac{1}{T}$ is the average rate at which a message is to be received by any node in the group. A simple algorithm, which will be described next, can be used in order to achieve these goals.

A. Synchronized case

Suppose all nodes are synchronized, that is, their time slots are perfectly aligned. At the beginning of the slot, each node computes a random time t_i, uniformly distributed in the $[0, T]$ interval. When time t_i is reached, the node decides whether to actually send the message or not, depending on the fact that a message has already been received between time 0 and time t_i. If no message was received, then the node sends its message, otherwise it waits for the next slot, and repeats the above procedure. In a given time slot, the message will be sent by the node which computed the shortest time t_{min}, and all other nodes will cancel their scheduled message.

Let $\mathbf{t_1}$, $\mathbf{t_2}$, ..., $\mathbf{t_n}$ be a set of n independent random variables, uniformly distributed in the interval $[0, T]$.

$$f_{T_i}(t_i) = \begin{cases} \frac{1}{T} & \text{if } 0 \leq t_i \leq T \\ 0 & \text{elsewhere} \end{cases}$$

Integration of $f_{T_i}(t_i)$ yields the cumulative distribution function of the random variable $\mathbf{t_i}$:

$$F_{T_i}(t_i) = \begin{cases} 0 & \text{if } t_i < 0 \\ \frac{t_i}{T} & \text{if } 0 \leq t_i \leq T \\ 1 & \text{if } t_i > T \end{cases}$$

Let $\mathbf{t_{min}} = min(\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_n})$.

We wish to find the cumulative distribution function and mean value of the random variable $\mathbf{t_{min}}$.

$$\{\mathbf{t_{min}} \leq t\} = \{\mathbf{t_{min}} > t\}' = \left\{\bigcap_{i=1}^{n} \{\mathbf{t_i} > t\}\right\}'$$

Therefore

$$F_{T_{min}}(t) = \begin{cases} 0 & \text{if } t < 0 \\ 1 - \left(1 - \frac{t}{T}\right)^n & \text{if } 0 \leq t \leq T \\ 1 & \text{if } t > T \end{cases}$$

Derivation of $F_{T_{min}}(t)$ yields the probability density function $f_{T_{min}}(t)$:

$$f_{T_{min}}(t) = \begin{cases} \frac{n}{T}\left(1 - \frac{t}{T}\right)^{n-1} & \text{if } 0 \leq t \leq T \\ 0 & \text{elsewhere} \end{cases}$$

The mean value of the random variable $\mathbf{t_{min}}$ is:

$$\epsilon = E\{\mathbf{t_{min}}\} = \int_{-\infty}^{+\infty} f_{T_{min}}(t)\, dt = \frac{T}{n+1}$$

As the number of nodes increases, the mean departure time value generated by the elected node decreases, and it will tend to 0 as the number of nodes tends to infinity. Therefore, a message will be sent at the beginning of each slot and all other messages will be dropped (not sent). The average rate of messages sent to the group would be $f = \frac{1}{T}$. Moreover, since each round of computation of the random time t_i is independent from the previous ones and from the other nodes, no assumption can be made about which node will be elected in a given round, so the probability of a node to be elected will be $\frac{1}{n}$, if n nodes are participating in the group.

B. Unsynchronized case

Let's remove the hypothesis about the synchronization of the time slots among the nodes. In this case, we use the same approach seen above, with one main difference: the reception of a message is used as a synchronization event among the nodes. To do so, instead of computing the time of the next scheduled message every T, we compute it starting from one slot after the time of reception or sending a message. This approach eliminates the need for synchronization among the nodes, but has the disadvantage that it may increase the mean time of a message being sent in the group. The time between two successive messages is the minimum computed time t_{min} plus T. Again, let $\mathbf{t_{min}} = min(\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_n})$. The mean time between two successive messages is:

$$E\{\mathbf{t_{min}} + T\} = E\{\mathbf{t_{min}}\} + T = \epsilon + T = T \cdot \frac{n+2}{n+1}$$

However, as n increases, the average time tends to T.

C. Problems with the simple algorithm

The algorithms sketched above assume that all nodes are able to detect immediately a received notify message and at the same time stop the sending process scheduled for the same timeslot. Although this is applicable in case of zero network delay, that is, the case in which the time needed for delivering a message is almost zero, it does not apply to real-world networks, because of the actual physical time requirements for a message to be delivered from end to end. At the contrary, it is possible that a message is still sent by one node that has computed a higher random value but that has not yet received the message from the real winning (elected) node. We call τ_i the *vulnerability interval* for the i-th node the time needed for the i-th node to receive the message sent from the elected node. Since in general such τ_i depends on node i, to the network topology, and to the current network traffic, we consider a *system vulnerability interval* of length τ, which is the worst-case estimation of the delivery time of a message. Due to the difficulty to estimate the worst-case network delivery time, any pre-configured upper bound can be considered. We refer to a “collision” as the event that a message is sent after the message sent from the actual elected node. A collision occurs any time a node that scheduled a departure time greater than by the elected node does not receive the message sent by elected node before its own departure time, due to network delay. We wish to calculate the mean number of collisions for a network consisting of n nodes. Let t_{min} be the minimum amongst the values $t_1, t_2, \ldots, t_n$ generated by the n nodes. Since the minimum value is t_{min}, all the nodes that have not generated t_{min} must have generated a value $\mathbf{t} > t_{min}$. Let's calculate the conditioned distribution of $\mathbf{t}$, given the event $\{\mathbf{t} > t_{min}\}$. Let $\bar{\mathbf{t}} = \{\mathbf{t} | \mathbf{t} > t_{min}\}$.

$$F_{\bar{T}}(t) = \begin{cases} 0 & \text{if } t < t_{min} \\ \frac{t - t_{min}}{T - t_{min}} & \text{if } t_{min} \leq t \leq T \\ 1 & \text{if } t > T \end{cases}$$

Derivation of $F_{\bar{T}}(t)$ yields:

$$f_{\bar{T}}(t) = \begin{cases} \dfrac{1}{T - t_{min}} & \text{if } t_{min} \leq t \leq T \\ 0 & \text{elsewhere} \end{cases}$$

The probability that one of the $n-1$ nodes that have not generated the minimum value t_{min} generated a value in the interval $[t_{min}, t_{min} + \tau]$ is:

$$F_{\bar{T}}(t_{min} + \tau) = \begin{cases} 0 & \text{if } t_{min} < 0 \\ \dfrac{\tau}{T - t_{min}} & \text{if } 0 \leq t_{min} \leq T - \tau \\ 1 & \text{if } t_{min} > T - \tau \end{cases}$$

We define

$$p(t_{min}) = F_{\bar{T}}(t_{min} + \tau)$$

The probability is function of the minimum value t_{min}. The probability that k of the the $n-1$ nodes generated a value in the interval $[t_{min},\, t_{min} + \tau]$ (that is, the probability to get k collisions) is:

$$\left(\begin{array}{c} n-1 \\ k \end{array} \right) \cdot p(t_{min})^k \left(1 - p(t_{min})\right)^{n-1-k}$$

This probability is function of n, t_{min}, and k. Given n and t_{min}, the distribution of the probability is a discrete binomial distribution, whose mean value is $(n-1)p(t_{min})$. We can then state that at each round, there will be k collisions in average, where:

$$E\{k\} = \begin{cases} (n-1)\dfrac{\tau}{T - t_{min}} & \text{if } 0 \leq t_{min} \leq T - \tau \\ n - 1 & \text{if } t_{min} > T - \tau \end{cases} \tag{1}$$

As n increases, t_{min} tends to 0, and therefore the number of collisions tends to the ratio $\frac{\tau}{T} \cdot (n-1)$.

V. Enhanced Algorithm

In order to avoid the problems outlined in the previous section, an enhanced version of the algorithm is used. The algorithm first estimates the number of nodes that are currently enrolled in the group, and then exploits this information to schedule the time of sending of the message.

A. Estimation of the number of collaborating nodes

Let us suppose that, each time a node sends a notification message, it includes also its scheduled departure time t_i. The knowledge of the minimum t_{min} of a set of uniformly distributed random variables and the number of random variables k whose value belongs to the interval $[t_{min}, t_{min} + \tau]$, is used to estimate the number of such random variables.

We can invert (1) to get the number of nodes as a function of the minimum generated time t_{min} and the mean number of collisions $E\{k\}$:

$$n = \begin{cases} E\{k\}\dfrac{T - t_{min}}{\tau} & \text{if } 0 \leq t_{min} \leq T - \tau \\ E\{k\} + 1 & \text{if } t_{min} > T - \tau \end{cases}$$

If we assume that the number of messages that were received in the interval $[t_{min}, t_{min} + \tau]$ coincides with the mean number of k ($k = E\{k\}$), then it is possible to invert (1) to estimate n:

$$\hat{n} = \begin{cases} k\dfrac{T - t_{min}}{\tau} & \text{if } 0 \leq t_{min} \leq T - \tau \\ k + 1 & \text{if } t_{min} > T - \tau \end{cases}$$

Since the assumption that $k = E\{k\}$ was made, the estimation works well if k is big, and therefore if n is big. $\hat{n}$ tends to overestimate the actual value of n. We handle the possibility of undelivered messages by considering the probablity of a lost message $p_{loss} = \Pr\{\text{message is lost}\}$. The number of collisions must be therefore adjusted by dividing it by p_{loss}.

B. Scheduling

We now want to exploit the information about the number of nodes that are currently participating in the group to generate a schedule in order to fairly balance the number of messages among the nodes while respecting the goal of having a constant rate of sent messages within the group.

Let $\hat{n}$ be the estimated number of nodes. Any node in the group randomly selects a number between 1 and $\hat{n}$. The selected number will correspond to a particular time slot self-assigned to the node. The node will wait until its time slot and will send its message at a randomly selected time in that interval. The departure time within the slot is randomized in order to let that, in case two or more nodes will select the same slot, one node would send its message first and the other nodes may detect such message and reschedule their message in a successive timeslot.

We will now evaluate this scheduling mechanism. Let n be the number of participants to the algorithm. Each participant selects a number between 1 and n. Let's define the event

$$E_{n,k} = \{k \text{ different values are selected out of } n\}.$$

We wish to calculate the mean value of the k different numbers selected by the n participants. Since the selection of the number is independent from participant to participant, it may happen that a number is selected by different participants. If n is the number of participants and each participant selects a number between 1 and n, the probability that a total of k different numbers are selected can be written as:

$$\Pr\{E_{n,k}\} = P_{n,k} = \frac{X_{n,k}}{N} \tag{2}$$

where N is the total number of possible outcomes of the event:

$$E_N = \{n \text{ particpants select a number between 1 and } n\}$$

and $X_{n,k}$ is the total number of outcomes of the event $E_{n,k}$.

E_N is clearly the event of generating all the dispositions of n elements from a set of n numbers. The number of all the outcomes is $N = n^n$.

If $k = 1$, $X_{n,k} = n$ since there are n possible sets of n values that contain exactly one value (one for each value). If

$k \geq 2$, $X_{n,k}$ can be written by the following formula:

$$X_{n,k} = \frac{n!}{(n-k)!} \sum_{i_1=0}^{n-k} k^{i_1} \sum_{i_2=0}^{n-k-i_1} (k-1)^{i_2} \cdots \sum_{i_{k-1}=0}^{n-k-\sum_{j=1}^{k-2} i_j} 2^{i_{k-1}} \tag{3}$$

Let's focus on the case $k \geq 2$. If we define

$$\mathbf{I_{n,k}} = \left\{ \underline{\mathbf{i}} \in \mathbf{N}^{k-1} : \|\underline{\mathbf{i}}\|_1 = \sum_{j=1}^{k-1} i_j \leq n-k \right\}$$

and the exponentiation of two vectors as

$$(a_1, a_2, \ldots, a_n)^{(b_1, b_2, \ldots, b_n)^T} = \prod_{i=1}^{n} (a_i)^{b_i}$$

then we can rewrite (3) as:

$$X_{n,k} = \frac{n!}{(n-k)!} \sum_{\underline{\mathbf{i}} \in \mathbf{I_{n,k}}} (\underline{\mathbf{K}})^{\underline{\mathbf{i}}^T} \tag{4}$$

where

$$\underline{\mathbf{K}} = (k, k-1, \ldots, 2)$$

and

$$\underline{\mathbf{i}} = (i_1, i_2, \ldots, i_{k-1}).$$

It is possible to find the cardinality $\theta_{n,k}$ of the set $\mathbf{I_{n,k}}$ by the following formula:

$$\theta_{n,k} = \frac{(n-1)!}{(n-k)! \cdot (k-1)!} = \begin{pmatrix} n-1 \\ k-1 \end{pmatrix} \tag{5}$$

Now we can define a matrix $Y_{n,k} \in \mathbf{M}(\theta_{n,k}, k-1)$ as:

$$Y_{n,k} = \begin{pmatrix} \underline{i_1} \\ \underline{i_2} \\ \vdots \\ \underline{i_{\theta_{n,k}}} \end{pmatrix} \tag{6}$$

We can extend the definition of vector exponentiation to the case of exponentiating a vector to a matrix. Let $\underline{a}$ be a vector of n elements and B a matrix $\in \mathbf{M}(n, m)$. Then, the result of exponentiating vector $\underline{a}$ to matrix B is a vector of m elements:

$$\underline{a}^B = (a_1, a_2, \ldots, a_n)^{(\underline{b_1}, \underline{b_2}, \ldots, \underline{b_m})} = \left(a^{\underline{b_1}}, a^{\underline{b_2}} \ldots a^{\underline{b_m}}\right)$$

where by $\underline{b_i}$ we denote the i-th column of B ($\underline{b_i}$ has n elements).

Now we can write

$$\underline{\mathbf{K}}^{Y_{n,k}^T} = \left(\underline{\mathbf{K}}^{\underline{i_1}^T}, \underline{\mathbf{K}}^{\underline{i_2}^T}, \ldots, \underline{\mathbf{K}}^{\underline{i_{\theta_{n,k}}}^T} \right) = \underline{\gamma_{n,k}} \tag{7}$$

and finally

$$\Gamma_{n,k} = \left\| \underline{\gamma_{n,k}} \right\|_1 = \sum_{j=1}^{\theta_{n,k}} \underline{\mathbf{K}}^{\underline{i_j}^T} = \sum_{\underline{\mathbf{i}} \in \mathbf{I_{n,k}}} (\underline{\mathbf{K}})^{\underline{\mathbf{i}}^T} \tag{8}$$

Therefore we can write:

$$X_{n,k} = \frac{n!}{(n-k)!} \cdot \Gamma_{n,k} \tag{9}$$

which lets us find the number of different sets of n elements that contain k different elements when those elements are drawn from a set of n different elements. Since $\Gamma_{n,k}$ is clearly function of both n and k, our goal is to find a function $\phi : \mathbf{N}^2 \to \mathbf{N}$ such that $\phi(n, k) = \Gamma_{n,k}$. It is banal to see that $\Gamma_{n,n} = 1$.

We can get the value of $\Gamma_{n,k}$ by reducing the problem to all the sub-problems of lower degree, in a recursive fashion. Such a reduction leads to an iterative formulation of the value of $\Gamma_{n,k}$:

$$\begin{cases} \Gamma_{n,k} = \sum_{i=0}^{n-k} k^i \cdot \Gamma_{n-1-i,k-1} \\ \Gamma_{n,n} = 1 \\ \Gamma_{n,2} = \sum_{i=0}^{n-2} 2^i = 2^{n-1} - 1 \end{cases} \tag{10}$$

With some calculations and index substitution it is easy to find an alternative and computationally more efficient representation for $\Gamma_{n,k}$. Given the above definition of $\Gamma_{n,k}$, we can write:

$$\Gamma_{n-1,k} = \sum_{j=0}^{n-k-1} k^j \cdot \Gamma_{n-2-j,k-1}$$

Therefore it is easy to demonstrate that:

$$\Gamma_{n,k} = \Gamma_{n-1,k-1} + k \cdot \Gamma_{n-1,k}$$

Even though there is no closed form to express $\Gamma_{n,k}$, a numerical computation shows that the mean of the number of non-empty slots tends to be approximately $k \approx 0.6n$. Therefore, the scheduling proposed leaves approximately half slots empty. In order to reduce the number of empty slots the picking could be made between 1 and $m = f(n)$ (with $m \leq n$), but collisions would be more luckily to occur. The handling of empty slots and collisions is discussed in the following section.

C. Algorithm description

The enhanced version of the algorithm requires two different stages:

1) a **synchronization round**: this stage is needed in order to let nodes learn about other nodes, that is, to let them estimate the number of nodes in the group ($\hat{n}$); the length of this stage is T;
2) a **notification round**: this stage is composed of $m = f(\hat{n})$ intervals of length T, during which nodes send the information about bootstrap nodes in the group;

Synchronization messages are received during synchronization rounds, which occur with variable period. A synchronization round's length is T. In this time interval, all synchronization messages are received. Synchronization messages are ordered by the reported t_i time generated by the sender. After all synchronization messages are received, the node counts all the distinct messages that report a t_i value that falls in the $[t_{min}, t_{min} + \tau]$ range (including its own if the t_i is in the range). Using the number k of such messages and the value of t_{min}, the node can estimate the number of nodes $\hat{n}$ that are present in the network using formula (1). The next synchronization round will occur after $m = f(\hat{n})$ slots.

The generation of a schedule can be made in a slightly different way than by picking a random value between 1 and m. Indeed, the statistics of the process remain the same if each node decides at the i-th slot whether to send the message or not with probability $p = \frac{1}{m-i}$. The node(s) that decide to send the message compute a random time t_i in the range $[0, T]$ to send the message. If a message is received prior to t_i, all the nodes that decided to send the message cancel their scheduled send. After sending a message, a node waits until the next synchronization round.

After each node has determined when to send its own message, it also generates a random instant in the corresponding interval at which to send the message. By doing so, nodes that have picked the same slot do not send their message simultaneously. If a node that has picked a slot receives a message during that slot, it cancels its scheduled message. In order to reduce the number of empty slots, a node that has picked an interval x that was also picked by another node, after cancelling the sendind of the message, it can pick another slot between x and m.

The presence of a synchronization round causes a slight decrease of the rate of information exchange since some round is dedicated to collecting information about the number of nodes that are participating. However, this decrease is not significant in the case of a high number of participating nodes.

VI. Implementation

In order to achieve the desired behavior, two type of messages are needed:

- **SYNC** messages, to be used during the synchronization round; they must, at least, include information about the sender and the generated t_i value.
- **NOTIFY** messages, to be used during the notification round; they must, at least, include information about the sender, the time of the next synchronization round, and the bootstrap nodes that the node is willing to advertise.

Messages are processed in the way described by the defined algorithm. The multicast nature of the proposed protocol forces us to use UDP as transport protocol. The definition of the protocol is very simple and lightweight. An early implementation of the BANANAS protocol has been realized based on the Java programming language and successively integrated in a mjSIP-based [6] Kademlia DHT system [4], [5] which we have developed. The BANANAS protocol is implemented according to a simple text-based HTTP-like protocol.

The usage of the BANANAS protocol is actually very easy: a node that is willing to learn about a suitable bootstrap node to join the desired overlay simply needs to listen on a well-known UDP port for traffic set to the BANANAS multicast group address. As result of our implementation, any node that wants to participate to bootstrap notification procedure, has just to start the implemented notification protocol that in turn starts computing the estimation of the number of collaborating node and the scheduling of the SYNC and NOTIFY message according to the simple algorithm specified in the previous sections. If a node wants to join the P2P network (a Kademlia-based P2P overlay in our implementation) it has to start the client side of our protocol implementation that in turn simply joins the BANANAS multicast group and listen for NOTIFY messages containing information about possible bootstrap peers. Such NOTIFY messages, according to the proposed and implemented algorithm, are received with mean frequency $f = \frac{1}{T}$, letting the joining node to wait a mean time T before being able to effectively join the P2P overlay.

VII. Conclusions

In this paper, we have discussed some issues regarding the bootstrapping problem in peer-to-peer overlay networks. The bootstrapping process requires that the a new joining node must first contact an existing node (the bootstrap node) that is already being enrolled in the overlay. Bootstrapping problems raise when a bootstrap node needs to be found. Some mechanism may include the use of nodes that are known in advance, caching, or server-based discovery. All of these approaches have pitfalls though.

In order to solve this problem, we have proposed a multicast-based approach for bootstrap node discovery called BANANAS. The described mechanism is based on an unsolicited algorithm, in which bootstrap node advertisements are not triggered by incoming requests but are automatically sent by the collaborating nodes to a specific multicast address. This mechanism has several advantages such as: complete distribution and self-organization, scalability, simplicity, guarantee of a mean frequency of advertisements and of a fair balance between all nodes. In the paper we presented the protocol description and the mathematical justification of each step and performance evaluation.

Finally, we have described our Java-based implementation of the algorithm.

References

[1] C. Cramer, K. Kutzner, T. Fuhrmann. Bootstrapping locality-aware P2P networks. In *Proceedings of the 12th IEEE International Conference on Networks (ICON 2004), 2004.*, Volume 1, Nov. 2004.

[2] M. Conrad and H.J. Hof. A Generic, Self-organizing, and Distributed Bootstrap Service for Peer-to-Peer Networks. In *Lecture Notes in Computer Science, Self-Organizing Systems, Volume 4725/2007, Pages 59-72*, Aug. 2007.

[3] P. Maymounkov and D. Mazires. Kademlia: A Peer-to-Peer Information System Based on the XOR metric. In *1st International Workshop on Peer-to-peer Systems*, 2002.

[4] S. Cirani and L. Veltri. A Kademlia-based DHT for Resource Lookup in P2PSIP. Internet-Draft draft-cirani-p2psip-dsip-dhtkademlia-00, IETF, October 2007.

[5] S. Cirani. Kademlia implementation, 2007. *http://www.mjsip.org/projects/p2psip/p2psip_dsip_071025.zip*

[6] L. Veltri. mjSIP project, 2007. *http://www.mjsip.org*