A Generic, Self-organizing, and Distributed Bootstrap Service for Peer-to-Peer Networks

Conference Paper · September 2007

DOI: 10.1007/978-3-540-74917-2_7 · Source: DBLP

CITATIONS
21

READS
339

2 authors:

Michael Conrad
Karlsruhe Institute of Technology
29 PUBLICATIONS **83** CITATIONS

SEE PROFILE

Hans-Joachim Hof
Technische Hochschule Ingolstadt
110 PUBLICATIONS **657** CITATIONS

SEE PROFILE

A Generic, Self-organizing, and Distributed Bootstrap Service for Peer-to-Peer Networks

Michael Conrad and Hans-Joachim Hof

Institute for Telematics, Universität Karlsruhe (TH), Germany
{conrad,hof}@tm.uka.de

Abstract. In many scenarios, self-organization is the driving force for the use of a peer-to-peer (p2p) network. However, most current p2p networks are not truly self-organizing, as little attention has been paid on how new nodes join a p2p network, the so-called *bootstrapping*. Current p2p network protocols rely on prior-knowledge of nodes like a list of IP addresses of bootstrap servers or like a list of known peers of a p2p network. However, this kind of prior knowledge conflicts with the self-organization principle and the distributed character of p2p networks. In this paper, we present the design of a *generic, self-organizing, and distributed bootstrap service* which can be used to bootstrap p2p networks of arbitrary size, even very small, private p2p networks. This bootstrap service works in today's Internet and it can be easily integrated into existing p2p applications. We present an evaluation of the proposed bootstrapping service showing the efficiency of our approach.

1 Introduction

Nowadays, peer-to-peer (p2p) networks are used in many applications, e.g. for VoIP, Instant Messaging, or filesharing. For most of these applications, decentralized control and self-organization are desired. However, most current p2p networks do not achieve true self-organization or true decentralized control because they often use well-known central servers or a list of known p2p network member nodes to bootstrap new nodes. In this paper, we propose a generic, distributed and self-organizing bootstrap service which allows nodes to join into arbitrary p2p networks. The proposed bootstrap service itself uses a p2p network, the bootstrap p2p network, for distributed storage of bootstrap information. The bootstrap information may include nodes which can be used to join the p2p network of the corresponding p2p application. For example if the user of a filesharing application starts the filesharing client for the first time, the client joins the bootstrap network and retrieves bootstrap information for the filesharing network. Then, it joins the filesharing network itself. Of course, this approach only shifts the problem of joining a p2p network to joining the bootstrap p2p network.

However, if the bootstrap service is implemented in more than one application, synergy effects may be used to join the bootstrap network. The synergy effect results from the larger number of nodes in the bootstrap network, which

D. Hutchison and R.H. Katz (Eds.): IWSOS 2007, LNCS 4725, pp. 59–72, 2007.

occurs because all nodes join the same bootstrap p2p network. One method for bootstrapping is Random Address Probing, which probes randomly chosen IP addresses to find active nodes. These nodes are used to establish initial contact with the p2p network. We use Local Random Address Probing for the bootstrapping of the bootstrap p2p network. While Random Address Probing may be successfully used by very large p2p networks, it is not efficient for small to medium networks, as many probes are necessary. To evaluate the performance of our bootstrap p2p network, we collected real world data about the distribution and number of peers of a deployed, large p2p filesharing network (eDonkey). Assuming that all eDonkey clients would use our bootstrap service, we evaluate the performance of our protocol.

This paper is structured as follows: in section 2, requirements for a bootstrap service are defined. Section 3 reviews related work. In section 4, the design of our generic, distributed, and self-organizing bootstrap service is presented. The bootstrap service is evaluated in section 5. Section 6 concludes this paper.

2 Requirements for a Generic Bootstrap Service

A generic bootstrap service for p2p applications and p2p networks must fulfill the following requirements:

- *Self-Organization and Distributed Control (R1):*
  A p2p network can only be self-organizing if every network protocol step, including the bootstrapping, is self-organizing. To support the distributed character of most p2p networks, a bootstrap service may not rely on prior knowledge (e.g. a list of IP addresses of bootstrap servers).
- *Heterogeneity (R2):*
  A decentralized bootstrap service should support p2p networks of arbitrary size and function. The service should provide bootstrap support for very small (private) p2p networks, only consisting of a few nodes, as well as for huge p2p networks with up to millions of active nodes.
- *Scalability and Robustness (R3):*
  The decentralized bootstrap service itself should scale well with an increasing number of nodes. The bootstrap service should also scale with the number of participating networks which is especially important if the bootstrap service is used to bootstrap a huge number of small, private p2p networks. The bootstrap service should work as expected, even in case of malfunction of several nodes.
- *Practicability (R4):*
  The bootstrap service should be designed for today's Internet. Hence, the bootstrap service may not rely on currently undeployed protocols like multicast etc.
- *Seamless Integration (R5):*
  It should be easy to integrate the bootstrap service into an existing p2p application.

– *Modularity and Extensibility (R6):*
A bootstrap service should be extendable to be able to react on changes of the network environment and to react on innovations. If, for example, multicast gets widely deployed in the Internet one day, it should be easy to extend the bootstrap service.

In this paper, we do not consider other requirements like privacy issues of the bootstrapping service etc.

3 Related Work

Cramer et al. [1] compare different bootstrapping techniques for p2p networks, including static bootstrap servers, out-of-band node caches, random address probing, and network layer mechanisms using any- or multicast. The results of the Random Address Probing method are of interest for our work. We enhance this mechanism for the proposed bootstrap service. The focus of [1] is on locality aware bootstrapping to offer an optimal topology for the join of new peers. However, no detail is given about the design of a generic bootstrap service. We fill this void with the proposed bootstrap service.

The idea of a generic bootstrap service for p2p networks was already discussed in [2] and [3]. The first paper is proposing an approach relying on a distributed hash table running on top of a structured overlay network, whereas the second paper uses a prefix based routing on top of a structured overlay network. The universal ring, which was proposed in [2], relies on a distributed hash table to store and query informations about services and to provide bootstrap information to use these services. While this approach can be used to provide bootstrap support for p2p networks, it does not meet some of the requirements of section 2: At first, the requirements R1 (Distributed Control) and R4 (Practicability) can not be met. To join the universal ring, a server or a globally known multicast address are used. A central server is prohibitive and multicast is a mostly undeployed technology in today's Internet. In addition, the scalability of the proposed for huge peer-to-peer networks seems unclear. Providing bootstrap support for large p2p networks using a distributed hash table results in storage of a huge amount of data on a very small set of nodes, hence overloading these nodes. In our opinion, a special distribution of bootstrapping information for huge peer-to-peer network is necessary to avoid overloading nodes of the bootstrapping nodes. The paper [3] lacks information on how nodes join the bootstrap service. The authors propose to use the NEWSCAST protocol [4] for data storage instead of a distributed hash table. As NEWSCAST relies on multicast, it does not meet requirement R4 (Practicability).

Even existing public peer-to-peer frameworks like JXTA [5] use static bootstrapping nodes (so-called seeds) to integrate new nodes into the peer-to-peer network. The JXTA framework also includes decentralized bootstrapping support using multicast, however due the missing deployment of multicast in the Internet infrastructure, this method is inapplicable for the use in the public Internet.

4 Design

This section presents the design of the proposed generic, distributed, self-organizing bootstrap service. The design meets all requirements of section 2.

4.1 Overview

Instead of creating a stand alone bootstrap mechanism for each existing peer-to-peer network (p2p network) we propose a single dedicated p2p network for the bootstrapping of all peer of arbitrary p2p networks. This so called *bootstrap p2p network* provides bootstrap information for peers which want to join a p2p network. Our bootstrap p2p network provides a service similar to the domain name system (DNS): while DNS resolves domain names to IP addresses, our bootstrap p2p network resolves p2p network names to bootstrap information (which is in most cases an IP address of a peer or a list of peers already connected to the p2p network). These bootstrap information can be used by a node to join the p2p network. After successfully joining a p2p network, the peer publishes bootstrap information to the bootstrap p2p network to supporting queries for bootstrap information for future peers. The peer also joins the bootstrap p2p network itself.

A dedicated bootstrap p2p network shifts the problem of joining an arbitrary p2p network to joining the bootstrap p2p network. However, as only one bootstrap p2p network exists for all p2p networks and all peers join the bootstrap p2p network, the bootstrap p2p network is larger than any other p2p network, allowing bootstrapping techniques, which are prohibitive for smaller networks, for example Random Address Probing. Especially very small p2p networks consisting of only tens or hundreds of peers can profit from this bootstrap p2p network, but it is also of benefit for large p2p networks because it simplifies the initial deployment of any p2p network.

4.2 Components of the Bootstrap Service

To achieve a flexible and extensible design our *bootstrap service* consists of two separate modules: The first module implements the initial bootstrapping of the bootstrap p2p network. The second module is responsible for providing bootstrap information (like a list of IP addresses of active peers) to nodes which want to join a distinct p2p network. Figure 1 shows the public interface and the schematic composition of the bootstrap service.

The public interface of the bootstrap service offers two methods. The method `lookup(name)` is used by a new node to search for bootstrap information of a p2p network whose identifier is `name` (e.g. *ed2k* for eDonkey peers). The method returns a list of `BootstrapData` objects. Each of these object contains a bootstrap information record (e.g. the IP address of one active node of the p2p network). After a node successfully joined a p2p network, it uses the method `publish(name,info)` to publish bootstrap information (`info`) about the p2p network `name`.

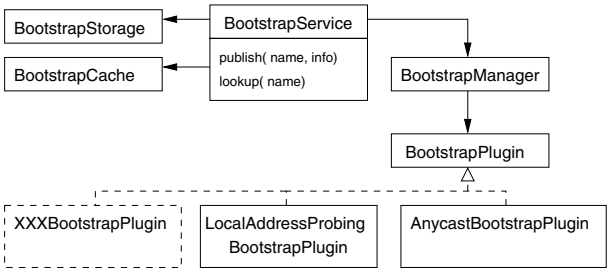

Fig. 1. Schematic composition of the bootstrap service

Every bootstrap service instance runs the *BootstrapManager*, which is responsible for the initial bootstrapping of the bootstrap p2p network. The `BootstrapManager` offers the interface *BootstrapPlugin* to support plugins to meet requirement R6 (Modularity and Extensibility). Each plugin implements one distinct bootstrap mechanism, e.g. Random Address Probing. Plugins are not limited to pure self-organizing bootstrapping methods. For example, there may be bootstrap plugins which use a node cache, or there may even be server-based plugins. However, it is recommanded to offer at least one self-organizing bootstrap plugin as fallback to maintain the self-organizing character of the boostrap service.

The component *BootstrapStorage* is responsible for storing bootstrap information. The proposed bootstrap service uses a soft sate approach, hence old bootstrap information will be deleted after a given time. The component *BootstrapCache* is used by the bootstrap service to cache information about peers of bootstrap p2p network to simplify the bootstrapping in case of a reconnect to bootstrap p2p network.

The design of the proposed bootstrap service does not contain any constraints about programming languages or other programming paradigms. Therefore, it can be integrated into arbitrary p2p applications. Hence, seamless integration (requirement R5) can be achieved. We expect that the easy and seamless integration will lead to a rapid deployment of our bootstrap service, especially if the open source community can be convinced to use the bootstrap service. Hence, the critical mass for the bootstrap service can be easily achieved.

The following sections give a detailed overview of the two modules `BootstrapManager` and `BootstrapStorage`.

4.3 BootstrapManager: Bootstrapping of the Bootstrap p2p Network

In our proposed bootstrap service, the *BootstrapManager* is responsible for the initial bootstrapping of a new node. To join the bootstrap p2p network, the BootstrapManager instance of the bootstrap service running on the node may uses one or more of its bootstrap plugins to join the bootstrap p2p network.

A number of different bootstrapping techniques are possible. Starting from simple bootstrapping plugins, which query static bootstrap servers (hence do not meet requirement R1), plugins using multicast or anycast based bootstrapping (`AnycastBootstrapModule`) (hence not meeting requirement R4) are possible. To meet all the requirements of section 2 (especially Practicability (R4)) a more advanced bootstrapping technique is needed. We propose to use *Local Random Address Probing*, a variant of Random Address Probing, as the default bootstrap technique. Random Address Probing sends probe messages to random IP addresses hoping to find one active peer of the desired p2p network. The performance of Random Address Probing [1] strongly depends on the number of peers and the distribution of these peers in the IP space. For most p2p networks, Random Address Probing is prohibitive because these networks do not have enough users, resulting in a very poor performance of Random Address Probing.

However, the proposed bootstrap service uses the bootstrap p2p network, which is potentially very large. Hence, Random Address Probing may be used. *Local Random Address Probing* is similar to the classical Random Address Probing, but instead of probing IP addresses uniformly distributed over the complete IP address space, the Local Random Address Probing limits the probed IP range to the local range around the current IP address of the user. This behavior of Local Random Address Probing improves the performance of Random Address Probing because local communication may be faster than remote communication and because the distribution of p2p nodes may not be uniform. Hence, the improved performance of Local Random Address Probing is partly based on the assumption that most users of p2p applications are private users which get Internet access via dialup networks (DSL, cable). Hence, we expect a higher locality of p2p users in dialup networks than in other network ranges. We verify this assumption in section 5.1. We exploit the higher density of p2p network peers in dialup networks to further improve the performance of the initial bootstrapping. Using Local Random Address Probing for our bootstrap service, requirements R1 and R4 can be met as there are no more dependencies to a central infrastructure. In contrast to other bootstrap services (see section 3), our proposed bootstrap service can be deployed in today's Internet.

4.4 BootstrapStorage: Efficient Distribution of Bootstrap Information

Regarding the potentially large number of nodes in the bootstrap p2p network, the distributed management must be very efficient to avoid overloading of peers. Management duties includes storage of potentially many published bootstrap information and an efficient search for existing bootstrap information. These requirements can be met by using distributed hash tables based on structured overlays like Chord [6], CAN [7] or Pastry [8]. Distributed hash tables offer a distributed storage of data and an efficient search. They scale well with an increasing number of peers. Another advantage of structured peer-to-peer overlays is a common abstract interface described in [9]. Hence, the bootstrap service

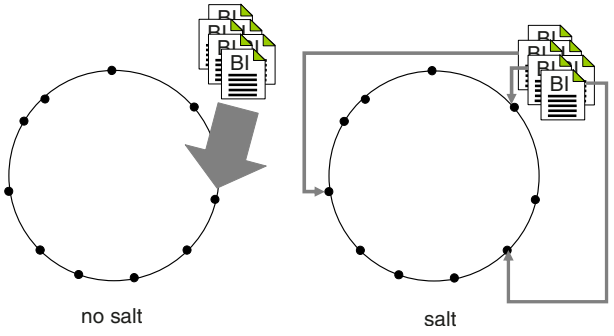

Fig. 2. The use of a salt value for our bootstrap service

can be implemented independent from the used overlay network, offering the possibility to change it later.

The *BootstrapStorage* module uses the `put(key,data)` method to store bootstrap information in the bootstrap p2p network. The bootstrap information `data` is stored under the key `key`. The key is calculated from the name of the p2p network. The bootstrap information can be retrieved using the `data=get(key)` method. Both methods are provided by the common API of [9]. However, the usage of these methods implies, that all bootstrap information for one distinct p2p network is stored under the same key, as the key is derived from the name of the p2p network in which a peer wants to join (e.g. "ed2k" for eDonkey). Figure 2 ("no salt") shows this problem. Regarding very large networks like the eDonkey filesharing network, it is clear, that the bootstrap information of this network would surely overload the peer of the bootstrap p2p network which stores all bootstrap information. We propose an adaptive algorithm for a better distribution of bootstrap information over the bootstrap p2p network:

Instead of using only the name of the requested p2p network we propose to include an additional random value, the so called *salt*, in the calculation of the key. The key is calculated using the name of the p2p network concatenated with the salt as input of a hash function H:

$$\text{key} = H(\text{name} + \text{salt})$$

Using this generation of the key results in a uniform distribution of bootstrap information of one distinct p2p network across the whole bootstrap p2p network. To search for bootstrap information, a node randomly selects a salt, calculates the key and queries the bootstrap p2p network for that key. While the improved distribution avoids overloading single peers of the bootstrap p2p network, it is not suitable for small p2p networks, as the probability of a successful search query is very small, hence resulting in many search queries with different salts. To provide a solution for this problem, we propose an *Adaptive Salt Window Algorithm*, which is able to adjust the distribution of bootstrap information automatically to the size of the particular p2p network without knowing the number of peers. The Adaptive Salt Window Algorithm uses an interval (the so

```
01   publish( name, info) {
02
03     for( window=0, i=n; i>3 && window==0; i--) {
04       salt = random( 2^i);
05       key = hash( name + salt);
06
07       result = p2p_lookup( key);
08
09       if ( |result| >= 10 ) { // sufficient data available
10         window = i + 1 // increase window
11       } else if ( |result| > 0 ) { // data available
12         window = i;
13       }
14     }
15
16     salt = random( max( 2^window, 8));
17     key = hash( name + salt);
18
19     dht_put( key, info);
20   }
```

```
lookup( name) {

  for( i=n; i>3; i--) {
    salt = random( 2^i);
    key = hash( name + salt);

    result = p2p_lookup( key);

   if ( |result| > 0 ) { // data available
     return result
    }
  }

  for( i=0; i<8 && |result|==0; i++) {
    key = hash( name + random( 8));

    result = dht_get( key);
  }
  return result
}
```

Fig. 3. Pseudo code of `publish` and `lookup`

called Salt Window) in which all salt values are contained. The Salt Window is adapted to the current number of peers of the p2p network. Small p2p networks only have a small window, whereas large p2p networks have a large window.

Figure 3 shows the pseudo code of the `publish` method and of the `lookup` method using the Adaptive Key Window Algorithm.

The `publish` method and the `lookup` method both automatically detect the current Salt Window size of the requested p2p network. The publish method starts with a maximum length interval of $[0, 2^n]$ in which a random salt is contained. In each step, the publish method halves this interval and tries to retrieve data using the name of the p2p network and a random salt in $[0, 2^i]$. This step is repeated unless bootstrap information is found. The corresponding interval is the Salt Window used for the storage of the bootstrap information. The size of the Salt Window is slightly adopted if more than 10 values to allow for an increase of the Salt Window size.

When a node uses the lookup method to query for bootstrap information of a p2p network the range of the salt value starts from the maximum value and will be halved until one valid bootstrap information was found.

The adaptive adjustment of the key window guarantees the support for arbitrary p2p network independent of their number of peers. It prevents overloading single peers hence results in an efficient storage of bootstrap information. The Adaptive Salt Window Algorithm allows to react on an increasing or decreasing number of peers. If the number of peers changes significantly, the Salt Window will be resized with high probability. Therefore the proposed bootstrap service meets requirements R2 (Heterogeneity) and R3 (Scalability and Robustness). Other load balancing mechanisms will be addressed in future work.

5 Evaluation

In this section, we evaluate the performance of the proposed bootstrap service which uses Local Random Address Probing for the initial bootstrapping of the bootstrap p2p network. We also analyze the scalability of the bootstrap p2p network.

5.1 Performance of Local Random Address Probing

In section 4.3 we assumed a non-uniform distribution of p2p network users in the IP address space. Furthermore we assumed, that computers connected to the Internet via dialup networks (e.g. DSL, cable) have a higher probability of running a p2p application than computers in the rest of the Internet. In this section, we provide strong evidence that these assumptions are justified. To verify these assumptions, we measured the distribution of active nodes of the eDonkey network[1], one of the biggest filesharing networks in Germany today. By inspecting a real-world p2p network, we provide a lower bound for the expected performance of the proposed bootstrap service, because we expect several p2p networks and not just one to integrate our generic, distributed and self-organizing bootstrap service. The eDonkey protocol runs as default on Port 4662. To detect eDonkey nodes, we used nmap[2] to perform a TCP SYN scan on port 4662. This scan provides a lower bound of active eDonkey nodes because it scans only for the default port. However, it is easy to change the default port and some users do this to prevent rate regulation of their provider. We also do not take the blocking of the default port into consideration, which is done sometimes by several Internet service providers.

We limit our examination of active eDonkey nodes to the German IP address space to avoid distortion of the results by different time zones and similar effects. We use the public daily database snapshot of the European Internet Registry (RIPE)[3] to get the currently allocated IP address space of Germany. In April 2007 this list contained about 160.000 IP ranges allocated with the German country code. These IP ranges are equivalent to 450.000 /24 networks, each network containing about 250 valid IP addresses. We used this list (GIPL, German IP List) as basis of our experiment. The list of German dialup networks (GDUL, German dialup network List) was created manually from a set of 38 dialup IP ranges, consisting of 29.000 /24 networks.

The experiment itself runs on a standard desktop computer connected via 100 MBit Ethernet to the campus network. Every 6 minutes a set of 5 /24 networks was extracted randomly from both lists (GIPL and GDUL). Each of these 5 networks was scanned for active eDonkey nodes as described above. Starting from the given network address, nmap was configured to scan the corresponding /20 network, which consists about 4000 valid IP addresses.

Our experiment was running from April 10th till April 18th. We scanned 21.600 /20 networks (containing 80 million ip addresses) for active eDonkey nodes, 10.800 from each of the lists (GIPL and GDUL).

Probing ranges from GIPL, 2.987 out of 10.800 probes found at least one active eDonkey node, whereas 9.193 out of 10.800 probes in the range of GDUL found active eDonkey nodes.

From over 1.100.000 online computers found in networks of GIPL only 4.39% (48.525) run the eDonkey software. The probing of the dialup networks of GDUL

[1] original website down, see `http://en.wikipedia.org/wiki/EDonkey2000`
[2] network mapper - website: `http://insecure.org/nmap`
[3] website: `http://www.ripe.net/ripe/index.html`

discovered about 880.000 online computers out of which 11,46% (100.781) were identified as active eDonkey node.

These results show that the distribution of p2p applications in dialup network is higher than in other networks. Hence, our assumptions are justified at least for the eDonkey network. Furthermore, in over 85% of probed dialup networks of GDUL at least one active eDonkey node was found whereas this is the case for only 27.65% of networks in GIPL. At the same time, the amount of active eDonkey nodes in dialup networks is twice higher than in other networks.

Figure 4 shows the distribution of eDonkey nodes across online computers for random networks (GIPL) and dialup networks (GDUL). It can be easily seen that the distribution of active eDonkey nodes is higher for dialup networks than for random networks. The maximum of active eDonkey nodes will be reached on the weekend (14-th and 15-th of April), corresponding with the Internet usage of dialup users.

As we will use Local Random Address Probing for the bootstrapping of the bootstrap p2p network, the number of probes before the first active bootstrap node stands for the overhead which our proposed bootstrap service generates and, more important, the number of probes is directly correlated to the time a user has to wait before it can join the p2p network which uses our bootstrapping service.

Figure 5 shows that in average about 600 probes are necessary to find an active node of the eDonkey p2p network in dialup networks (GDUL). Finding the first active eDonkey node requires in average less than 20 seconds. For random networks the equivalent value can not be given, because only about 30% of the network scans discover at least one active eDonkey node.

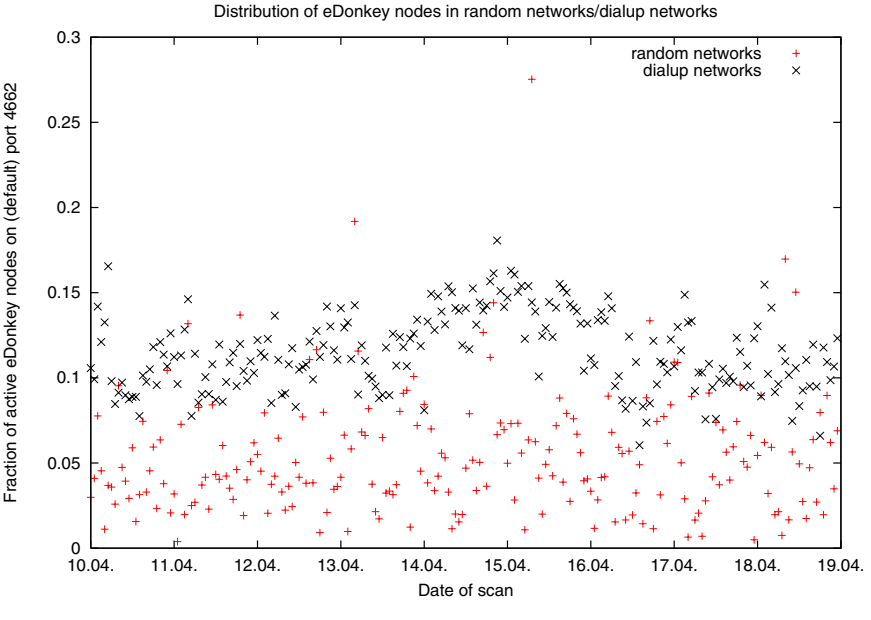

Fig. 4. Distribution of active eDonkey nodes in random or dialup networks

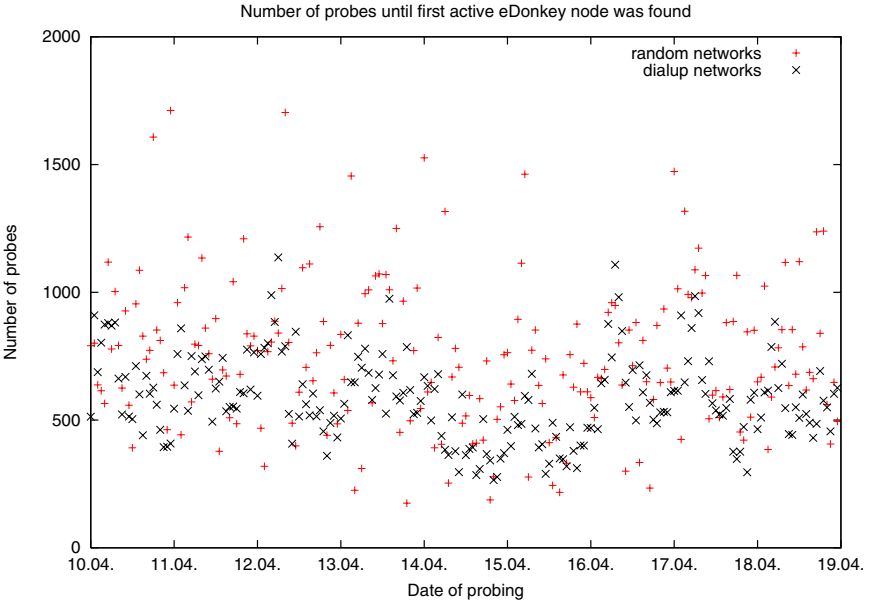

Fig. 5. Number of required probes to find first eDonkey node

In comparison with random networks (GIPL) a lower number of probes is required in dialup networks (GDUL) until the first active eDonkey nodes is found. At the same time, the deviation is significantly lower. This result support our assumption, that local random address probing is well suited for finding other peer nodes in dialup networks. The lowest number of required probes for dialup networks was reached in our scan on the weekend of 14-th/15-th April.

Figure 6 shows the cumulative distribution of time needed for probing until the first eDonkey node was found in dialup networks (GDUL) and in random networks (GIPL). Thereby the results for random networks only rely on the 30% of successful probes. It shows the efficiency of our approach. For example, after 20 seconds, over 80% of nodes found the first eDonkey node. For our bootstrap service this is equivalent to a successful bootstrapping.

Analyzing the eDonkey p2p network as a synonym for a distributed bootstrap service the initial bootstrapping into the bootstrap overlay can be satisfyingly realized by Local Random Address Probing if the bootstrap service has a large enough number of nodes. These findings show that Local Random Address Probing can be efficiently used for the bootstrapping of our bootstrap p2p network.

5.2 Performance of the Bootstrap Information Storage

This section evaluates the performance of the Adaptive Salt Window Algorithm (see section 4.4) used during storage and retrieval of bootstrap information.

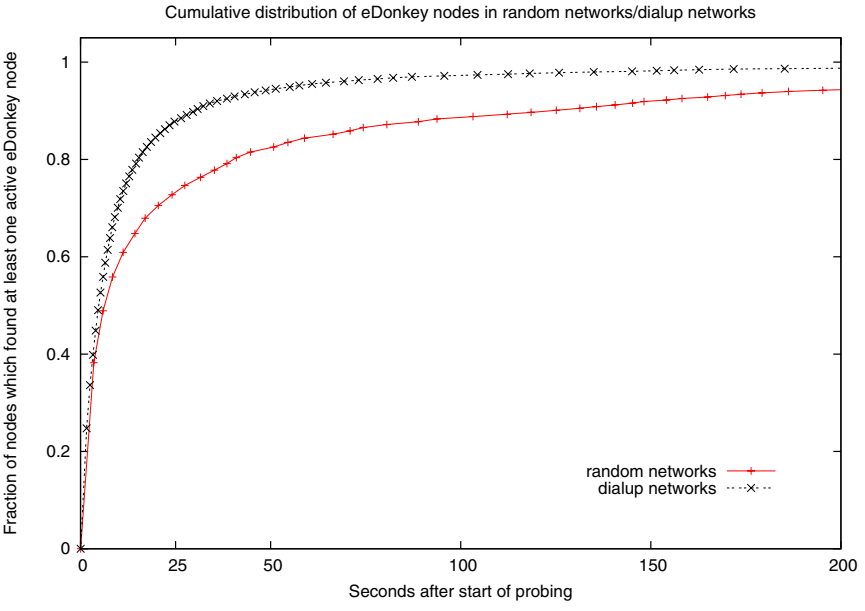

Fig. 6. Cumulative distribution of eDonkey nodes

We define 4 different scenarios of p2p networks distinguished by their number of users (10^6, 10^5, 10^4 and 10^3). Starting from a running bootstrap p2p network with 4.200.000 (2^{22}) peers, all nodes bootstrap into one network using the proposed bootstrap service. Therefore new peers use to the bootstrap p2p network to retrieve bootstrap information by generating a lookup query. In all scenarios the number of overlay queries, resulting from lookup-queries, were inspected. After a successful join into the desired p2p network each node publish bootstrap information for that p2p network.

Figures 7 shows the number of overlay queries generated by lookup requests for a large p2p network with 10^6 nodes, where all nodes joining the bootstrap p2p network subsequently. For the other scenarios, only differing in the number of joining nodes, similar results for the average number of overlay queries generated by lookup requests, will be archived.

At the beginning, when only few bootstrap information for the requested p2p network are available, up to 25 overlay queries are necessary for each lookup query. With an increasing number of nodes which join the new p2p network and publishing bootstrap information the number of overlay queries decreases significantly.

For 10^3 nodes the average number of lookup queries is about 12, for 10^4 or 10^5 peers 9 respectively 6 lookup queries are required in average. For large p2p networks with 10^6 nodes in average only 3 lookup queries are necessary. The simulation shows, that the proposed distribution of bootstrap information scales with a increasing number of peers. At the same time, small p2p networks are also supported, although a higher number of overlay queries is required.

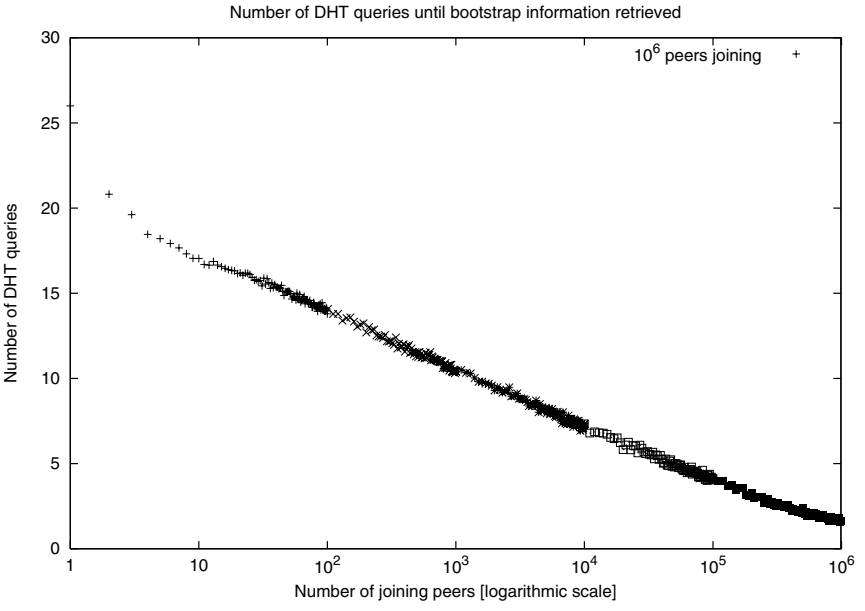

Fig. 7. Number of DHT queries to obtain bootstrap information

6 Conclusion and Future Work

We presented a generic, distributed, and self-organizing bootstrap service for arbitrary peer-to-peer networks (p2p networks) which can be used in today's Internet. Our proposed bootstrap service offers bootstrapping for small private p2p networks as well as for large p2p networks. The bootstrap service is easy to integrate into existing p2p applications and can be extended by plugins.

For storage of bootstrap information, our bootstrap service uses a distributed hash table. The Adaptive Salt Window Algorithm is used to achieve an efficient distribution of bootstrap information across the nodes which run the bootstrap service, hence preventing the overloading of single nodes.

We evaluated the proposed bootstrap service using real world data of a large p2p network, the eDonkey filesharing network. The results show that our bootstrap service can be efficiently used to bootstrap arbitrary p2p networks.

Future work will address a simulator implementation and a prototype implementation of the proposed bootstrap service.

References

1. Cramer, C., Kutzner, K., Fuhrmann, T.: Bootstrapping locality-aware p2p networks. In: Proceedings of the IEEE International Conference on Networks (ICON 2004), Singapore, November 16–19 2004, vol. 1, pp. 357–361. IEEE Computer Society Press, Los Alamitos (2004)

2. Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron, A.: One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In: EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC, Saint-Emilion, France, pp. 140–145. ACM Press, New York, NY, USA (2002)
3. Jelasity, M., Montresor, A., Babaoglu, O.: The bootstrapping service. In: ICDCSW '06: Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems, Washington, DC, USA, p. 11. IEEE Computer Society Press, Los Alamitos (2006)
4. Jelasity, M., van Steen, M.: Large-scale newscast computing on the Internet, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam. Technical Report IR-503. Amsterdam, The Netherlands (October 2002)
5. Gong, L.: Project JXTA: A technology overview (August 2001), `http://www.jxta.org`
6. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM'01, San Diego, California, USA (2001)
7. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, United States, pp. 161–172. ACM Press, New York, NY, USA (2001)
8. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
9. Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., Stoica, I.: Towards a common API for structured peer-to-peer overlays. In: 2nd Int. Workshop on P2P Systems (2003)