Decentralized Bootstrapping in Pervasive Applications

CONFERENCE PAPER · MARCH 2007

DOI: 10.1109/PERCOMW.2007.36 · Source: DBLP

Decentralized Bootstrapping in Pervasive Applications

Mirko Knoll*, Arno Wacker*, Gregor Schiele†, and Torben Weis*

*University of Stuttgart
70569 Stuttgart, Germany
{ knollmo | wackerao | weistn }@ipvs.uni-stuttgart.de

†University of Mannheim
68131 Mannheim, Germany
gregor.schiele@uni-mannheim.de

***Abstract*— Deploying world-scale pervasive applications usually requires substantial investments in server infrastructure. A better option is to use P2P techniques, because it distributes the load from the servers to the peers. However, this creates a new problem. Users do not know each others upfront and we do not want to rely on a central server infrastructure and therefore a single point of failure. Thus, we must provide a bootstrapping protocol that allows users to discover at least one peer of the overlay network. In this paper we give a survey of possible bootstrapping protocols and propose one approach that satisfies all of our requirements.**

I. Introduction

Specialized P2P systems are very useful for pervasive applications as shown in [1], [2]. Most work on P2P systems concentrates on efficiently searching for data objects or message routing in the overlay network. However, when the user starts the client software of a pervasive application, the PC, PDA, or smartphone must connect with the corresponding P2P overlay network. This is especially complicated since a) the users do not know each other and b) we do not want to rely on a central server infrastructure. This problem of how to include a new peer into a P2P overlay network is known as the bootstrapping problem [3]. Today's P2P systems provide only partial solutions for the bootstrapping problem or omit it at all by assuming that each new peer knows at least one existing peer at start-up. In this paper, we analyze existing approaches and discuss their shortcomings. In addition, we present a new approach, which reuses an existing internet service to provide a robust, scalable, and efficient bootstrapping solution.

The paper is structured as follows: first, we describe the bootstrapping problem in more detail and define the terminology. Then, we state our requirements. In Section 4 we present existing approaches and in Section 5 we discuss our distributed bootstrapping protocol. Section 6 features the conclusions and an outlook on future work.

II. Problem Statement

Before giving our problem statement we describe our assumptions. We assume the existence of a connected *communication network*. We call members of this network *nodes*. This communication network provides unique addresses for each node. Furthermore, it provides routing mechanisms i.e. an arbitrary node can send messages to any other node of this network. An example for such a communication network is the global Internet.

We further assume the existence of a connected *P2P overlay network* on top of the communication network. We call the members of this network *peers*. Clearly all peers are also nodes with respect to the communication network. Nodes may join the overlay at any time thereby becoming additionally a peer. Analogously, a peer may leave the overlay network at any time, loosing its peer role. That means, the P2P network can vary in its size from zero to all nodes in the communication network dynamically.

With theses assumptions in mind we define the bootstrapping problem as follows: Given a node n which wants to join the overlay network, i.e. become a peer, how can n find a peer which is already in the overlay network, if there is any. Our goal is to solve the bootstrapping problem for an Internet-based overlay network. Once this is achieved, n can use the found peer to integrate itself into the overlay, e.g. by joining a virtual ring.

III. Requirements

To solve the bootstrapping problem we aim at developing a bootstrapping protocol that allows nodes to detect existing peers and to join the overlay network reliably, even if there are only a few peers currently available, or none at all. Such a protocol must fulfill a number of requirements to guarantee functionality at all times. We discuss these requirements in the following.

A. Robustness against Failure

Robustness is one of the most important properties of a bootstrapping process. The system and all its components should be completely decentralized and there should be no single point of failure. If the network relies on a single entity, for example by providing a server, where host lists (containing IP addresses of recent peers) can be downloaded, attackers would be empowered to restrain the network functionality dramatically. In case of a DoS attack on the server, further nodes could easily be prevented from joining the overlay network. Even more damage in terms of privacy is done when the server is hijacked. This enables phishers to collect personal information by re-routing all data over subverted servers.

B. Robustness against Security Appliances

Another requirement is the robustness in the presence of security appliances. Personal firewalls and routers using network address translation (NAT) pose a barricade for some bootstrapping mechanisms, e.g. connections on specified ports. The bootstrapping process must finish successfully regardless of the network structure and present devices.

C. Robustness against External Interference

Lastly, we attach importance to robustness in terms of resistance against external interference. All necessary components for the bootstrapping should be completely decentralized. This prevents an authority, such as the Recording Industry Association of America (RIAA) in the case of Napster, to shut down the entire overlay network. On the other hand, the system should still continue to work in case the initiator of the system decides to leave. If so, a handover process has to be started, which allows other participants from the overlay to take over the tasks of the initiator.

D. Efficiency

We consider a system to work *efficiently*, when it guarantees a fast and slim bootstrapping process. This includes the necessity to be able to join the network within a reasonable amount of time while limiting the network traffic on the communication network to a minimum.

E. Scalability

Another aspect of great importance is the *scalability* of the bootstrapping protocol. In this context, two kinds of scalability are important. First, scalability with respect to the number of nodes, i.e. the protocol is useable in large communication networks. Second, scalability with respect to the number of peers, i.e. the protocol is able to handle a large overlay network with many peers.

IV. Existing Approaches

There are different approaches to enable a node to detect a peer dynamically. We distinguish between peer-based approaches and mediator-based approaches. In the following we discuss these two classes in more detail.

A. Peer-based approaches

Peer-based approaches try to directly detect and contact peers. To do so, they send messages to ordinary peers in the overlay network. In this paper, we present three such approaches: peer caches, multicasting, and random probing.

1) Peer Cache: One simple but effective example for such approaches are *peer caches*. A peer cache contains peers that were active in the past. Before leaving the overlay, a peer stores all other peers that it is connected to in its cache and saves it on stable storage. The next time the node wants to join the overlay, it loads its peer cache and tries to contact all peers in it directly. If one or more peers answer, the node can use them to join. This is a simple, robust, efficient, and scalable approach. Unfortunately it only works if at least one peer in the cache is currently present in the overlay. Otherwise, peer caches fail to solve the bootstrapping problem.

2) Multicast: Another reliable and convenient method for bootstrapping is to use *multicast*. As an example, Cramer et al [4] propose to let all peers in a local area network (LAN) join a common multicast group. When joining the overlay, a node sends a query to the multicast group to receive the IP address of other participants and connect to them afterwards. This approach is both simple and efficient. However, in current networks multicast traffic usually is not routed beyond the local network domain. Therefore, this approach cannot be used for bootstrapping in a world-spanning communication network.

3) Random Probing: The third peer-based approach that we discuss in this paper is *random probing*. Here, a node searches for peers by randomly selecting nodes in the communication network and sending them messages over a specified port [4]. If the contacted node is a peer, it answers, otherwise a new IP is generated and contacted. While this procedure might work in very dense networks with a high peer to node ratio, it is completely unsuitable for starting a new overlay network. Nodes suffer from a poor hit rate due to the many, necessary connection attempts until a peer is found. Furthermore, connection over a specified port is not always possible, as personal firewalls or NAT routers prevent the connection from being established. Security appliances lack heuristics to distinguish between the bootstrapping process and a virus or worm attack.

B. Mediator-based approaches

The second class of approaches uses special mediators to find peers. A mediator can be a peer in the overlay or an external node. Mediators act as directory services for peers and constitute a *well known entry point* (WKEP), which is hard-coded into the bootstrapping protocol. When a node wants to join the overlay network, it contacts a mediator requesting the addresses of one or more currently active peers. The main challenge for mediator-based approaches is to keep the mediator information up to date and the mediators reachable. Several different approaches have evolved over time in various products, but only a few are still in use.

1) Napster Server: In Napster, bootstrapping is done by contacting a hard-coded central server (farm) [5]. This server maintains a database containing information about all files that peers are willing to share. At runtime the server mediates between peers as needed. The main drawback of this approach is its reliance upon the central server. Without it, the overlay network is not operational, making the system very fragile.

2) Gnutella Host Caches: The original Gnutella client uses a hard-coded URL to identify a server used for bootstrapping, the so-called *host cache* [6]. After connecting to the host cache, the client receives a list of recent peers, which it can try to contact. A similar approach was presented with YOID [7], which was later used in CAN [8] and other recent P2P networks. In times when the host cache was unreachable, users manually exchanged peer addresses using chat rooms. Unlike the Napster server, the host cache is only needed for bootstrapping the network, not for its continuous operation. Nevertheless, it constitutes a bottleneck in the system.

3) eDonkey Server Lists: The eDonkey software uses a web based approach to distribute information about active servers in the network. Users download a so-called *server list*, which includes recent information. To eliminate the need to download recent server lists daily, the client can receive updates over the eDonkey network. However, this feature is often abused by Anti-P2P clients, which emit fraudulent information.

4) Bittorrent: Bittorrent [9] is a file sharing tool that lets clients upload file fragments among each other. To download a file via Bittorrent a special *.torrent* file is needed, which includes some file information and the URL of the so-called *tracker*. A downloading client contacts the tracker to inform it about the file it is downloading and on what port it listens to incoming requests. Thus, clients, which want to download at a later time, can contact the tracker to delegate them to other clients, which already have downloaded parts of the file. This procedure adds robustness as the original file provider can disappear once the file has been completely uploaded. However, the location of the torrent file has to be known to the user before being able to download a file. In addition, the tracker must be available.

V. Our Approach

As we have seen, there are a number of existing approaches to solve the bootstrapping problem in P2P overlay networks. However, none of these approaches fulfills all our requirements. Peer-based approaches are either not generally usable (peer cache), not scalable (random probing) or not usable in large communication networks due to current network restrictions (multicast). Mediator-based approaches typically rely on the availability of a hard-coded mediator (Napster, Gnutella) or require the user to manually provide information about the WKEP by downloading a data file (eDonkey, Bittorrent). Neither of these approaches is able to automatically and robustly solve the bootstrapping problem. Therefore, we decided to develop our own bootstrapping protocol. In the following, we present our current work in progress on this protocol.

Our bootstrapping protocol consists of two parts. First, we use peer caches to try to reconnect to already known peers. In most cases, this will allow the node to join the overlay. If no peer in the peer cache can be contacted, or if the peer cache is empty, we use a mediator-based approach as fall back. The basic idea of this approach is to use an *existing* – preferably distributed – Internet service to perform the mediation for us. This way, we do not have to provide specific servers for bootstrapping. At this time, we are analyzing existing services with respect to their usability for our cause. We discuss some possible services, their strength and shortcomings, in the remainder of this section.

A. Search Engines

Search engines (Google, Microsoft Live, etc.) are very powerful and efficient when it comes to finding information on the net. This makes them well suited for the bootstrapping process. To use a search engine for bootstrapping, we can e.g. embed a list of recent peers (preferably with high uptime) on some web pages using microformat [10]. By accessing the web page, a web-service is triggered which updates the list on the web page and adds the requesting node to it. To find a peer, a node contacts the search engine and searches for a set of specific keywords. It then extracts the addresses of recent peers from the found page and tries to connect to one of them.

As search engines are used every day for billions of queries, this approach will neither affect their functionality nor run into scalability problems. The presence of firewalls and NAT-routers pose no problem, as the use of a search engine usually is not restricted in any way. Thus, the demands for scalability (requirement E) and robustness against security appliances (requirement B) are met. Furthermore, the approach produces little traffic for the search for the microformatted data. Upon joining the overlay, a single entry is added to the peer list, keeping it up to date and the bootstrapping phase slim and efficient (requirement D).

However, the approach is vulnerable against external interference (requirement C). The search engine provider can easily block queries for the web page. In addition, if the web page is hosted on a pre-defined web server, the system is not robust against the failure of this server (requirement A). The latter can be fixed by hosting the web page on a dynamically elected peer. However, this would require the search engine to update its information every time that a new peer takes over hosting the web page. Typically, such updates will not be available immediately but may take several hours or even days.

B. Dynamic DNS

The dynamic domain name service (DDNS) in the Internet emerged by the demand for hosting a server on a computer with a dynamic IP address. DDNS, just like the normal domain name service (DNS), maps domain names to IP addresses. A DDNS provider typically offers a web interface to allow computers to update the mapping of a certain domain name dynamically. This way, a server can change the mapping any time its IP address changes.

To use DDNS for bootstrapping, the provider of an overlay network reserves a number of domain names at the DDNS provider, e.g. peer01.homeip.net, peer02.homeip.net and so on. When a node wants to join the overlay, it checks, if it can contact one of these domains. If so, it has successfully detected a peer. Otherwise, it is one of the first peers in the network. In this case it contacts the DDNS provider's web interface and redirects one of the reserved domain names to its own IP address. During normal operation, peers check periodically if all DDNS names are still "occupied" by other peers. If a name is unoccupied, the corresponding peer has left the overlay and the detecting peer takes its place (with the necessary measures in order to not conflict with other peers, which also detected this).

The DDNS approach is very efficient (requirement D), since a node normally only needs to perform a single DNS request to detect an existing peer. It is robust against failure (requirement A), as leaving peers are detected and others take their place.

Security appliances pose no problem (requirement B), as DNS can be used behind firewalls and NAT-routers. In addition, the approach is highly scalable (requirement E) with respect to the number of nodes. However, special care must be taken to provide scalability with respect to the number of peers. Otherwise, the periodic checking of the DDNS names may overload the corresponding peers. Most importantly, the approach is not robust against external interference (requirement C), since the DDNS provider could easily deactivate the account for the DDNS peers and thereby hinder new peers in joining the overlay network.

C. Internet Relay Chat

As the name already suggests, the internet relay chat (IRC) [11] is a service which provides an Internet-wide chat service. The IRC network is comprised of a number of servers, which are (statically) interconnected. Due to the interconnection of all servers, any client can connect to any of theses servers and communicate with any other client connected to a – possibly different - server. Today many thousands of IRC networks coexist in the Internet. A client connected to one IRC network can only communicate with clients of the same IRC network.

With its highly decentralized architecture, the IRC service is a good choice for bootstrapping a P2P network. To do so, the designer of the overlay network decides for one of the bigger IRC networks and has joining nodes connect to one of the various servers of this specific IRC network. The list of suitable IRC servers can either be provided directly with the client software or automatically downloaded from a web site with the use of search engines as mentioned above.

After connecting to an IRC server a node enters a well-known chat room, a so-called *channel*. All other participants, which are also in this channel are members of the overlay network. Obviously, the peers cannot be connected with an IRC server and a specific channel at all times, as this would increase the load on the IRC network and hinder the usage of its original purpose. Hence, we suggest that only a fixed and small subset of peers stays in the IRC channel, in order to act as an entry point for new nodes. When the size of the subset drops under a certain threshold (e.g. since peers left the P2P network and thus also the IRC channel), peers from the overlay network are invited to take their place.

The IRC-based approach is very robust against all kinds of interferences. To interfere with bootstrapping, an ordinary attacker would have to shut down potentially thousands of IRC servers. An IRC operator could ban users or protect a channel to hinder bootstrapping. To counter this, additional measures are necessary, e.g. random user names and ”hot swapping” the channel at runtime. In the presence of NAT and/or firewalls the IRC channel can be seen as a backup communication channel between peers, as an outgoing TCP connection to an IRC server is allowed by most firewalls. Hence, this approach fulfills our *robustness* requirement. The approach works *efficiently*, since it uses only few resources on the host-service and the operation of joining the overlay is just a matter of trying nodes from the peer-cache and additionally (in the unsuccessful case) a connection to an IRC server.

VI. Conclusion and Future Work

Pervasive applications have become popular over the last years. That's why it is important to offer mechanisms, which allow a fast and reliable setup of those applications. Therefore, we need procedures enabling new nodes to find other participants and join their group. In this paper we clarified the requirements for finding the first other node to allow for a reliable bootstrapping process. We then presented popular approaches and analyzed their performance in regard to those requirements. As none of these approaches satisfies all demands, we present three ideas for different procedures. We believe that those may be suited for our purpose. Especially, the IRC approach seems to work fine and promises to be very robust. Nevertheless, there are still some aspects, which are not yet completely researched. In the future, we plan to implement our approaches and conduct some field tests, to evaluate the road capability. Furthermore, we want to investigate in long-term approaches, such as IP v6 and modified DNS records, which make the use of out-of-band communication obsolete. This will surely require massive changes in the current network structure, but as pervasive applications gain more and more importance in the everyday life, it is a necessary step.

References

[1] M. Saternus, T. Weis, M. Knoll, and F. Dürr, “Symstry: Ein P2P-System für Ortsbezogene Anwendungen,” 2006, submitted paper.

[2] D. Heutelbeck, “Distributed Space Partitionin Trees and their Application in Mobile Computing,” Ph.D. dissertation, Open University Hagen, Germany, May 2005.

[3] C. Cramer and T. Fuhrmann, “Bootstrapping chord in ad hoc networks: Not going anywhere for a while,” in *Proceedings of the 3rd IEEE International Workshop on Mobile Peer-to-Peer Computing*, Pisa, Italy, March 2006, publication. [Online]. Available: http://i30www.ira.uka.de/research/publications/p2p/

[4] C. Cramer, K. Kutzner, and T. Fuhrmann, “Bootstrapping locality-aware p2p networks,” in *Proceedings of the IEEE International Conference on Networks (ICON)*, Singapore, November 2004, publication, pp. 357–361. [Online]. Available: http://i30www.ira.uka.de/research/publications/p2p/

[5] C. Shirky, “Listening to napster,” in *Peer-to-Peer. Harnessing the Power of Disruptive Technologies*, A. Oram, Ed. Sebastopol, CA, USA: O’Reilly, 2001, pp. 21–37.

[6] G. Kan, “Gnutella,” in *Peer-to-Peer. Harnessing the Power of Disruptive Technologies*, A. Oram, Ed. Sebastopol, CA, USA: O’Reilly, 2001, pp. 94–122.

[7] P. Francis, “Yoid: Extending the internet multicast architecture,” 2000. [Online]. Available: http://www.icir.org/yoid/docs/yoidArch.ps

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content-addressable network,” in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. ACM Press, 2001, pp. 149–160.

[9] B. Cohen, “Incentives build robustness in bittorrent,” in *In Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003. [Online]. Available: http://citeseer.ist.psu.edu/cohen03incentives.html

[10] R. Khare, “Microformats: the next (small) thing on the semantic web ?” *IEEE Internet Computing*, vol. 10, pp. 68–75, Jan-Feb 2006.

[11] J. Oikarinen and D. Reed, “Internet relay chat protocol,” Internet Network Working Group RFC 1459, May 1993.