2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises

Bootstrapping Peer-to-Peer Systems using IRC

Mirko Knoll, Matthias Helling, Arno Wacker, Sebastian Holzapfel, and Torben Weis

University of Duisburg-Essen
47057 Duisburg, Germany
firstname.lastname@vs.uni-due.de

Abstract

Research in the area of peer-to-peer systems is mainly focused on structuring the overlay network. Little attention is paid to the process of setting up and joining a peer-to-peer overlay network, i.e. the bootstrapping of peer-to-peer networks. The major challenge is to get hold of one peer that is already in the overlay. Otherwise, the first peer must be able to detect that the overlay is currently empty. Successful P2P applications either provide a centralized server for this task (Skype) or they simply put the burden on the user (eMule). We propose an automatic solution which does not require any user intervention and does not exhibit a single point of failure. Such decentralized bootstrapping protocols are especially important for open non-commercial peer-to-peer systems which cannot provide a server infrastructure for bootstrapping. The algorithm we are proposing builds on the Internet Relay Chat (IRC), a highly available, open, and distributed network of chat servers. Our algorithm is designed to put only a very minimal load on the IRC servers. In measurements we show that our bootstrapping protocol scales very well, handles flash crowds, and does only put a constant load on the IRC system disregarding of the peer-to-peer overlay size.

1 Introduction

Peer-to-peer (P2P) systems of the third generation commonly form an overlay network to exchange data without exceeding computing capabilities of participating peers. In order to join this overlay network, a new peer needs a mechanism to contact an already integrated peer. In accordance to [1, 4] we call this mechanism *bootstrapping*. Despite the fact, that the bootstrapping service needs to be completely decentralized, it has to be available for anyone at any time. Otherwise, multiple isolated overlay networks may emerge, and thus partition the P2P system involuntarily.

[1]The presented work has been funded by DFG SPP 1183 "Organic Computing" and DFG "Emmy Noether" Program.

In this paper we present a refined solution for bootstrapping in P2P systems. We extend our earlier idea [5] and present our revised bootstrapping protocol based on the Internet Relay Chat (IRC). We describe the approach in detail, evaluate it and discuss its strengths and shortcomings.

2 System Model

Before discussing the requirements of our bootstrapping approach, we describe our system model briefly.

In our scenario computers using a common communication network may send to and receive messages from other computers reliably. We assume, that a subset of those computers runs a P2P-software, afterwards labeled as *peers*. All participating peers form an overlay network built on top of the communication network. In reality, this communication might be complicated due to the existence of firewalls or NAT-routers. However, we developed suitable NAT traversal mechanism to bypass such appliances [8]. Hence, dealing with the firewall problem is not the focus of this paper. We do not assume a specific algorithm to form the overlay (e.g. you may use Chord or Pastry).Additionally, we expect peers to join and leave the network without prior announcement. The essence of the bootstrapping problem may be described best by finding a peer in the overlay (if there actually is one) to become a peer itself.

3 Requirements

In this section we describe the requirements, that need to be met by a boostrapping mechanism. A more detailed discussion can be found in our earlier work [5].

Availability: Peers may join and leave whenever they want. Thus, bootstrapping needs to be available at any time, as well. In order to guarantee availability, the mechanism needs to be completely decentralized, as already mentioned above, to eliminate the possibility of disintegration by a single point of failure.

Automation: The bootstrapping mechanism needs to work fully automated and without user interaction.

1524-4547/09 $25.00 © 2009 IEEE
DOI 10.1109/WETICE.2009.40

IEEE computer society

Efficiency: Peers should be able to join the overlay in a reasonable amount of time, while limiting the network traffic to a minimum.

Scalability: The number of peers already joined should be irrelevant for the functionality and efficiency of the bootstrapper.

Fairness: The workload should be distributed equally between all peers.

Robustness against security appliances: For security reasons, peers may be hidden behind firewalls and routers. Nevertheless, they should be able to use the bootstrapping service.

4 Design Rationale

Before describing our approach for bootstrapping, we will motivate our major design decisions. Two well-known methods exist for bootstrapping: (1) peer-based and (2) mediator-based bootstrapping.

Peer-based bootstrapping: Peer-based bootstrapping searches for an already integrated peer and tries establishing a connection to it. A peer-cache, for instance, saves the addresses of former known and integrated peers to contact them again, when bootstrapping the next time. If the connection to a saved peer can be established, it can be used as the entry-point to the overlay network. If no connection can be established this way, the new peer is not able to join the network. Despite the fact, that this method is not completely reliable, it is efficient and simple.

Mediator-based bootstrapping: In contrast to peer-based bootstrapping, the mediator-based bootstrapping uses a well known entry point (WKEP) to find a peer that is already integrated to the overlay network. The mediator is required to have a static address and to be available from everywhere. Often, the mediator is a server, raising the problem of posing a single point of failure. Thus, this method may not be very reliable.

In our bootstrapping approach, we combine both of these methods to create a reliable bootstrapping method. When bootstrapping, the new peer will first try to connect to a peer from his peer-cache. If this succeeds, the entry-point is found, otherwise, it will need to connect to the mediator. In our case, the IRC server takes the role of a mediator. By utilizing a common IRC network service, we also eradicate the possibility of a single point of failure, as the IRC architecture is completely decentralized.

The field of existing technologies to solve the bootstrapping problem is vast. However, none of these approaches fulfills all of our requirements. The class of peer-based approaches is either generally unusable (peer cache), does not scale (random probing) or simply is 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 *well-known entry point* (WKEP) by downloading a data file (eDonkey, Bittorrent). Apparently, neither of these approaches is able to automatically and robustly solve the bootstrapping problem. Therefore, we decided to develop our own bootstrapping protocol, which we present in the following.

5 IRC Bootstrapping

To allow for a simple and efficient bootstrapping we believe there is no other way than to use an existing technology. The Internet Relay Chat (IRC) [7] is one of the oldest decentralized services on the Internet. All participating servers are interconnected, thus allowing a client to connect to any of the servers while being able to communicate with any other client, even if the client is connected to another server. When IRC was introduced there was essentially one big IRC network. However, this network grew so fast that in 1996 the one big IRC network was split in multiple smaller networks. Today many thousands of IRC networks coexist in the Internet, while the biggest ones are QuakeNet, EFnet, IRCNet and Undernet. Nodes that connect to one of those networks talk to nodes from the same IRC network only.

With its highly decentralized, almost non-killable architecture, the IRC service is a good choice for bootstrapping a P2P network. Actually, IRC was already used in Gnutella to find out about other peers, but this approach required the user to take some manual actions. We envision a full-automatic use of the IRC service directly by the peers of the P2P network without any user interaction, such that the user does not need to care about the bootstrapping itself - it simply connects. The founder of an overlay selects one of the larger IRC networks and has new nodes connect to one of the servers of this specific network. A list of suitable IRC servers is provided with the client software or listed on the project website.

5.1 Bootstrapping Phase

In order to bootstrap, the peer checks whether there are peers in its peer cache (see Chapter 7). If the peer cache is not empty, the overlay tries to bootstrap with randomly chosen peers until a predefined maximum number of retries has been reached. If failed, or the peer cache is empty, the peer continues by connecting to the IRC server. After the peer has successfully connected to an arbitrary *IRC server* of the specified network, it calculates the current chat room, a so-called *channel*, based on the IRC server time.

This channel is grouping all participants to members of the overlay network. The peer then chooses an ID (here: nick), which comprises an identifier for the overlay (e.g. 'peer') followed by a random number. Using this nick, the peer calls the `Join` procedure and thus receives a list of all members, being currently in the channel. With this list,

we can identify the bootstrapping peers (BSPs) and how many bootstrapping peers there are. When becoming a BSP, the peers change their nicks to match another format (e.g. $'bs_peer'$). This way, we can differentiate between a 'regular' peer and a BSP. Entering a channel leads to the following two cases:

No bootstrap peer in channel: If the member list does not contain a BSP the newly arrived peer has no chance to learn about other peers from the overlay. Thus, the peer becomes a bootstrap peer itself and changes its nick from $'peer < id >'$ to $'bs_peer < id >'$. In that case the bootstrapping phase ends though the BSP is not yet aware of other peers. This situation only occurs when a new overlay is established. Once new peers enter the channel, the BSP is able to provide them with its own IP address.

Algorithm 1 IRC Bootstrapping in general

```
procedure Bootstrap(IRCServer)
Connect(IRCServer)
Join(CalculateChannel())
while WaitForActivePeerList(Δ(t_WBQ)+Random(ε)) is empty do
    SendChannel("PeerQuery")
end while
if JoinOverlay(ActivePeerList) then
    if GetBSPCount() < MaximumBSPs then
        BecomeBSP()
    else
        Leave(Channel)
        Disconnect()
    end if
end if
```

Bootstrap Peer available: In case the peer has found a bootstrap peer in the current member list, it waits for the time t_{WBQ} plus a random offset before querying the bootstrap peer (`SendChannel`). This allows for flash crowds to enter the channel simultaneously without increasing the load. One peer will request the peer list and one BSP will answer with his bootstrapper list. Note that the peers contained in this list are not necessarily the same that are acting as bootstrapping peers in the IRC channel. This way, we can distribute the tasks efficiently to different peers: the ones listening on the IRC channel, and the ones integrating new peers into the overlay. As the IRC implements an application multicast, the reply (the query as well) is heard by all peers in the channel. Thus, all other peers waiting for an address are served concurrently and therefore do not have to send queries separately. This approach is not only efficient, but as well necessary as most IRC servers have a strict limit on how many concurrent messages they allow within a certain time interval. A pseudo code example for this bootstrapping scheme is given in Algorithm 1.

Regarding security appliances, it may happen that the joining peer is not able to connect to a designated bootstrapper that is behind a firewall or a router. As we assume that some peers are "hidden" behind a firewall, we use our NAT traversal mechanisms for the connection establishment, as we presented in [8].

5.2 Joining the Overlay

After the bootstrapping phase, the peer enters the overlay using the IP address of a BSP it was given before (`JoinOverlay`). Therefore, it then sends a request to the BSP outside the channel using direct communication. Doing so does not impose any load on the IRC service provider. The BSP then sends all information necessary to enter the overlay. Once the peer successfully connects with the overlay it leaves the IRC channel shortly after. As we have to keep the communication in the channel minimal, only BSPs remain in the channel. In case there are enough BSPs new peers will leave the channel as soon as they have joined the overlay. This is the reason, why we waive to use regular peers for bootstrapping new nodes. We want to keep the communication traffic to a minimum and thus only keep a reasonable amount of bootstrapping nodes in the channel. If a peer cannot connect to the overlay network within a given time interval, it chooses another bootstrapping peer and reruns the bootstrapping phase until it eventually succeeds.

5.3 At Runtime

Bootstrap peers have a special task, as they have to guarantee a successful bootstrap at all times. Through the `SelfStabilizing` procedure they discover (see `GetBSPCount`) when the amount of bootstrap peers drops under a specified *BSPThreshold*. In this event, the peer will wait a random timespan until it will recheck whether the situation has improved through actions of another BSP. If not, the peer commands the overlay network to search for new bootstrappers. Random delays before sending search commands prevent the channel from growing unnecessarily. Peers that receive a search command connect to the IRC network, enter the appropriate channel and assign themselves a bootstrapping nick.

To fulfill the fairness requirement, we check a BSP's uptime on a regular basis using `BSPTimeElapsed()`. If the following two requirements are met, a BSP may leave the IRC channel and stop acting as a BSP. First, its uptime exceeds the threshold $max_{BSPuptime}$, and second the amount of remaining BSPs in the channel is greater than *BSPThreshold*. This feature disburdens a single peer from acting as a BSP for a longer time. Instead, over time we distribute the role of a BSP to all participating peers using round-robin. To minimize the chance of multiple BSPs leaving at the same time because their uptime exceeds the maximum, we introduce random delays before quitting.

The next peer that successfully joined the overlay network, will now take this leaver's place and act as BSP. A further improvement to achieve a better workload distribution balance is to split the task of bootstrapping and distribute the two remaining tasks to different peers. Peers without firewalls may now act as so-called super-peers, that integrate new peers into the overlay network, while firewalled peers are listening to the IRC channel and deploy the addresses of those super-peers. This way, new peers also do not try to connect to symmetric-firewalled peers, requiring much more time to become integrated into the overlay network.

5.4 Leaving

Peers can either leave the IRC channel *actively* or *passively*. Leaving passively implies that connections or the application are closed forcefully. As already mentioned, when peers leave the IRC server without announcement other BSPs are requesting new bootstrappers using the overlay network, if there are less than c_{min} BSPs in the channel.

Peers are leaving actively if they announce their intention of leaving. The active leaving actions depends on the peer's role. In common, peers need to check whether they are needed as a BSP immediately before leaving the IRC server. In case a non-BSP wants to leave, it waits some random time first. Then, it checks whether there still are enough BSP in the channel. If so, it leaves the server. Otherwise, it becomes a BSP. In case a BSP wants to leave, another strategy is needed to retain consistency. If the number of BSPs is greater than c_{min}, the peer initiates a search-for-bootstrapper query, waits some random time and leaves the server. If the number of BSPs is less or equal to c_{min}, the peer initiates the query too, but instead waits for a successor.

The active leaving process is comparably simple. A regular peer may simply leave the overlay without any further procedures. BSPs have to check whether the number of their kind is still above the *BSPThreshold*. If that's the case, the peer may leave the channel and the overlay. In the other case, the bootstrap peer has to invite a node out of its own peer cache to take its place. Here as well, the node keeps sending invitations until the condition is satisfied.

6 Evaluation

For this performance evaluation we focus on the messages being used for the bootstrapping process solely. Therefore, we omit messages sent by peers inside the P2P overlays to get a clean result. To perform this evaluation we prototypically implemented a simulator application. Using this simulator, we examined the traffic necessary for our bootstrapping approach. Note, that to fulfill the strict regulations of IRC servers, only a limited amount of peers resides in the IRC channel. Using the IRC for simulations requires the setting of few parameters which we describe in the following. In general, we only consider messages being sent to the IRC server. Messages being sent from the server are not taken into account as we cannot influence them.

Server Messages contains messages for registering in the channel and setting the name. Lifeness checks use `Ping` messages until the leave the channel. *BSP Messages* contain the query for a bootstrap peer. In future versions this message type will also include the data required for a certificate exchange. *Total Messages* sum up server and BSP messages as this combined value must not exceed standard IRC boundaries (around 10-20 messages per second). If a peer sends more messages it will be kicked from the channel or eventually banned entirely from the IRC server. The *online time* specifies the time a non-bootstrap peer stays in the IRC channel. The longer a node stays in the IRC channel, the higher the probability, that this peer will eventually become a BSP. We could also invite peers from the overlay back into the IRC channel to overtake a BSP's role. However, this would increase the message load on the channel and might lead to banishment.The *lifetime* denotes how long a peer stays in the overlay network. This parameter further limits the online time, as a peer will not stay in the IRC channel once it left the P2P network. The *BSP min/max* value is used to compensate the effects of a higher churn rate. To guarantee a successful bootstrap we aim to keep a constant number of BSPs in the channel at all times. To suppress reactions on every BSP entering or leaving the channel, we introduced two constants, namely *max* and *min*. Nodes entering the IRC channel check whether the current amount of BSPs is smaller than *max* and while BSPs leaving the channel check whether the amount is larger than *min*.

6.1 Scenarios

We used three different models to cover a wide range of possible situations. For the node generation process we used Markov chains. Thus, each node "birth" is independent from the former one.

Steady Rise: In this scenario we want to maximize the amounts of participants. Thus, we set the online time for peers to infinity, such that they will not leave the IRC channel after a specified time. Thereby, the amount of peers in the channel will steadily increase until the maximum amount of peers in the channel is reached. This limit is set by the IRC server and usually circles around 100 nodes per channel.

Short Lifetime: To maximize the effect of a high churn rate, we set the lifetime of the peers very low (between 150 and 200 seconds). To further stress the the IRC server, we reduced the online time to 1 minute as well. In doing so, we can analyze the amount of messages being sent into the channel in times where the nodes come an go very quickly. The amount of BSPs concurrently in the channel is set lower to stay within 5 and 8 to enforce a high churn of BSPs also.

Regular Runtime: To monitor the usual behavior in the

average case, we set the parameters according to models from literature. [2] has analyzed that the session time for a peer in the overlay network (e.g. in Gnutella) averages around 80 minutes. The online time we retained at 1 minute, thus simulating peers that leave the channel soon after they have successfully bootstrapped. The values for the number of BSPs in the channel in is set between 5 and 15.

6.2 Results

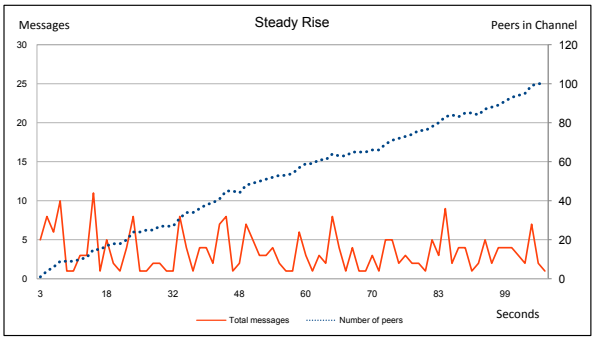
As we use several bootstrap peers in the channel the load on each one stays low as there are no periodic messages, that have to be sent. In general, we can state that the amount of messages in the channel depends on the node arrival frequency. The more nodes arrive within a short time interval the more messages are sent. The time between joining a channel and sending a request for an IP plus the fact that channel messages are broadcasted allows for constantly low message rates. Even in the Steady Rise scenario (Figure 1(a)) with a hundred nodes in the channel, the overall message rate does not exceed 12 messages / second. This is due to the fact, that we can keep a steadily constant amount of bootstrap peers in the channel, serving incoming requests. Even with a high churn rate, the amount of bootstrap peers and total messages stays within tolerant boundaries (Figure 1(b)). As we have already shown to provide the ability to successfully bootstrap without putting to much load on the IRC provider in rather stressing scenarios, we also examined the behavior with standard parameters. Figure 6.2 shows a typical IRC channel with peers coming and going while the message rate stays extremely low (around 1-3 messages/second). Thus, we are able to show that our protocol fulfills our requirements for an efficient bootstrapping protocol.

6.3 Open Issues

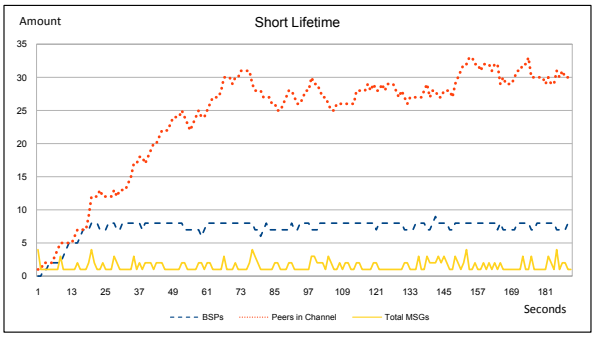
In this paper we focus more on the technical feasibility than on security aspects. Nonetheless the presented IRC-based approach is prone to attacks from third persons. A malicious user may, e.g. log in to the IRC server, join the channel and deploy a list of addresses, that do not exist, or are integrated in another, malicious network. Thus, the attacker can create partitions and disturb the bootstrapping methods with minimal effort. We have examined several approaches to hinder malicious users from disturbing the bootstrapping method:

Password-protected channels: The IRC protocol allows for protection of channels by a password. Everytime the bootstrapper creates a new channel, it sets a password, that is calculated using the server time. The downside of this approach is, that malicious users can easily find out the scheme after that the password is generated.

Certificate exchange: One possibility is, that users in the IRC channel exchange certificates before communicating.



(a) IRC Load under a steady rise



(b) IRC Behavior with nodes having a short lifetime

Figure 1. Evaluations on IRC

We use certificates to identify and authorize users for our overlay network. At the first glance, this may seem plausible, but malicious users may have valid certificates, too. Thus, this approach just makes things more complex but does not solve the problem.

Reversed connection: Another possibility is to reverse the connection establishment. This way, an integrated peer would contact the new one to let him enter the overlay. Unfortunately, this approach does not work either, since the new peer may be equipped with a firewall, blocking incoming connections.

Changing the channel periodically: Changing the channel periodically is a method we use to aggravate the possibility to disturb the system. Every hour a new channel is calculated with the IRC server time. The bootstrapping peers join this channel some time before the hour elapses and leave the old one after the hour elapsed. Unfortunately, this method only makes disturbing the system more difficult, but does not prevent this. Malicious users only need to change the channel, too. Therefore, they need to know how the channel name and its password is to be calculated. We will evaluate this method in detail in an upcoming paper.

7 Related Work

There are several different approaches on how to discover a peer in an overlay network. The focus of our

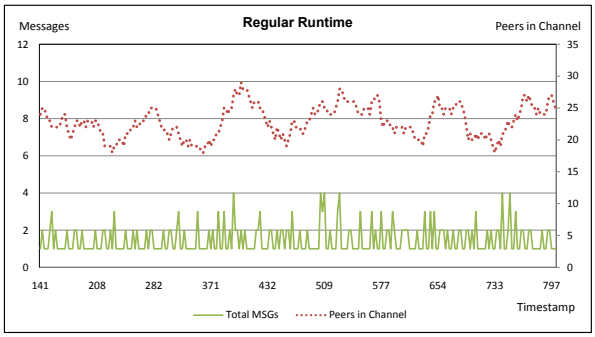


Figure 2. Regular IRC Behavior

approach lies in finding the first node of an overlay network rather than optimizing the bootstrapping process [2] or merging two rings into one [3]. In [6] we give a detailed overview on recent bootstrapping approaches relevant for our scenario. In the following, we distinguish between two different classes, either relying on a mediator or other peers.

Peer-based approaches try to detect active peers directly without a mediator. If they succeed, they receive information on how to join the overlay network. Peer caches rely on finding an active peer by maintaining a list of peers from former session. Though this approach is simple and efficient it fails when all formerly known peers are offline. Finding a node through active probing as proposed in [2] only works when there is another node in the same subnet. As those queries are not forwarded further this approach only has a chance to work in very populated networks. Searches initiated through broadcasts, multicasts or anycasts share the same problem. The current Internet infrastructure does not forward these type of packets and therefore does not allow for an efficient bootstrapping.

With mediator-based approaches, finding active peers of the overlay is achieved by the usage of a special mediator. Despite the fact, that these mediators often suffer from being a single point of failure, they are more reliable than peer-based approaches, in general. Mediators are used to retrieve an arbitrary amount of active peers. For instance, Napster and Bittorrent utilize a directory service to find active peers of the overlay, whereas in in Gnutellathe *host cache* is downloaded via a hard-coded URL. However, all of these approaches suffer from the mediators as a single point of failure.

8 Conclusion and Future Work

In this paper we presented a bootstrapping method using a peer-based and a mediator-based approach. If the peer-based approach fails, the mediator is used as a fallback. As mediator we use an IRC server, that is used if the peer cache approach was not successful. As the IRC service is completely decentralized there is no single point of failure. Thereby, our approach fulfills the *availability* requirement. Furthermore, our approach is *automated* and *efficient*, since it only poses a minimal load on the IRC server and on the bootstrapping peers, as shown in the evaluation. The evaluation also proves the *scalability* of our system, that is guaranteed by the leaving of non-bootstrapping peers. The required *fairness* is achieved by first allowing peers with higher uptime to leave the channel if possible and secondly by dividing the task of bootstrapping into two subtasks, namely *IRC bootstrapping* and *overlay bootstrapping*, as described in 5.3. *Robustness against security appliances* is not a direct task of the IRC Bootstrapper. We fulfill this requirement by using our own NAT Traversal framework presented in [8].

Thus, with our approach we have presented a solution for bootstrapping peers into a newly generated overlay network. Furthermore, this approach may also serve as a fallback solution for existing overlay networks as it helps to overcome server errors or bootstrap peers with an empty peer cache. Developers may therefore include this approach in their projects to help creating distributed applications without the need and the disadvantages of centralized authorities.

References

[1] 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.

[2] C. Cramer, K. Kutzner, and T. Fuhrmann. Bootstrapping locality-aware p2p networks. In *Proceedings of the IEEE International Conference on Networks (ICON)*, pages 357–361, Singapore, November 2004.

[3] A. Datta and K. Aberer. The challenges of merging two similar structured overlays: A tale of two networks. In *1st International Workshop on Self-Organising Systems*. Springer, 2006.

[4] W. Ding. Bootstrapping chord over manets - all roads lead to rome. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3501–3506, Kowloon, China, March 11-15 2007.

[5] M. Knoll, A. Wacker, G. Schiele, and T. Weis. Decentralized bootstrapping in pervasive applications. *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07)*, 0:589–592, 2007.

[6] M. Knoll, A. Wacker, G. Schiele, and T. Weis. Bootstrapping in peer-to-peer systems. In *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*, Melbourne, Australia, December 2008.

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

[8] A. Wacker, G. Schiele, S. Holzapfel, and T. Weis. A nat traversal mechanism for peer-to-peer networks. In *The Eight International Conference on Peer-to-Peer Computing (P2P'08)*, Aachen, Germany, September 8th-11th 2008. IEEE.