Decentralized Bootstrapping of P2P Systems: A Practical View

Jochen Dinger and Oliver P. Waldhorst

Institute of Telematics, Universität Karlsruhe (TH)
76128 Karlsruhe, Germany
{dinger,waldhorst}@tm.uni-karlsruhe.de

Abstract. So far, bootstrapping constitutes the only centralized task in otherwise decentralized peer-to-peer (P2P) systems. As a contribution to the development of generally applicable decentralized bootstrap mechanisms, in this paper we analyze two decentralized approaches from a practical point of view. We consider *local host caches* and *random address probing* for bootstrapping into the BitTorrent DHT as an example for a widely deployed P2P system. Based on the results of an extensive measurement study we show that local host caches allow rejoining the P2P system quickly after short times of disconnection, but are impracticable for infrequent or first-time users. Furthermore, random address probing is feasible using a direct Internet connection with high bandwidth, but is subject to practical issues raised by typical NAT routers and the distribution of ports used by BitTorrent clients. We propose two mechanisms for increasing the performance of random address probing: (1) probing multiple ports per host and (2) hash-based filter-resistant port selection, making distributed bootstrapping feasible even from a practical point of view.

Keywords: Internet measurement, peer-to-peer, decentralized bootstrapping

1 Introduction

Since the deployment of the first peer-to-peer (P2P) systems at the end of the 1990s, P2P has become a mature technology for building distributed applications. Many examples for currently widely-deployed P2P systems exist, such as the file sharing systems KaZaA [1], Gnutella [2], BitTorrent [3], and eMule [4], the IP-telephony system Skype [5], or the video distribution system Joost [6]. In all these systems, one issue is still solved in a centralized way: the bootstrap process, i.e., the task of finding an entry point to the P2P system in order to integrate a new peer. It typically relies on dedicated bootstrap servers that are reachable under well-known addresses. This aspect is crucial, e.g., if the bootstrap server has become unavailable due to a system failure.

In this paper, we analyze approaches for decentralized bootstrapping of P2P systems from a practical point of view. Using a widely-deployed P2P system, the

BitTorrent DHT, as an example, we analyze two mechanisms for decentralized bootstrapping: First, we consider a *local host cache* to store addresses of several recently active BitTorrent hosts as entry points for re-entering the BitTorrent DHT. Second, we analyze *random address probing*, i.e., scanning of randomly generated IP addresses, for discovering active BitTorrent peers as entry points.

We analyze the performance of both mechanisms by performing an extensive measurement study that has lasted several weeks and discovered several million of BitTorrent peers. We found that the lifetimes of the peers, i.e., the time they are connected to the Internet using the same IP address, are sufficient to make local host caches an effective tool for re-entering the BitTorrent DHT after short periods of disconnection. However, reconnecting infrequent users may take several minutes or even fail at all, as it does for first-time users of the P2P system. Furthermore, we show that finding an active peer by random address probing, again, takes a few minutes, even in a laboratory setting with a powerful Internet connection. Furthermore, we show that this time is significantly increased by practical issues like the performance of typical NAT[1] routers as well as by the number of BitTorrent peers using non-standard ports.

To significantly speed up decentralized bootstrapping, we propose two mechanisms for increasing the performance of random address probing: (1) probing multiple ports per peer, increasing the probability to discover a client by exploiting knowledge on the port distribution, and (2) selecting a port by hashing the IP address of the client, making it predicable but difficult to filter by an Internet Service Provider (ISP). These mechanisms make successful random address probing feasible in reasonable time.

The remainder of this paper is organized as follows. Section 2 provides an overview of P2P bootstrap mechanisms as well as related measurement studies. In Section 3 we discuss how local host caches and random address probing can be used for distributed bootstrapping into the BitTorrent DHT. In Sections 4 and 5, respectively, we show by extensive measurements the shortcomings of both mechanisms. Subsequently, we propose mechanisms for speeding up bootstrapping in Section 6. Finally, concluding remarks are given.

2 Background and Related Work

2.1 Bootstrapping P2P Systems

Bootstrapping basically describes the process of integrating a new node into a P2P system. As an anchor point for the new peer at least the address (more precisely the IP address and port) of one active peer, i.e., a peer that currently participates at the desired P2P system, has to be discovered [8]. Several approaches for the discovery of active peers in P2P systems have been proposed (cp. [9]):

Out-of-band mechanisms. When Gnutella was launched in 2000, the address of active peers was exchanged through IRC [10]. Moreover, websites became popular as out-of-band mechanism to discover active peers [11,8].

[1] Network address translation (NAT), cp. [7]

Dedicated bootstrap servers. Many recent P2P systems like BitTorrent use one or more (central) bootstrap server(s) with well-known DNS names or IP addresses. A node willing to join contacts a bootstrap server, which provides addresses of active peers. Even if multiple bootstrap servers exist, they may become subject to Denial-of-Service (DoS) attacks and hence single points of failure, e.g., as became obvious when the login nodes of Skype [12] have been overloaded [13]. Additionally, the number of bootstrap servers must be adjusted to the number of users (and the churn rate) of a P2P system.

Local host cache. To speed up reconnection, e.g., after a typical forced 24h-disconnection of a DSL-line by the ISP, nodes may maintain a list with addresses of other peers [14]. This list is denoted as *(local) host cache* and is used, e.g., by Skype [12]. It can be easily maintained while keeping the P2P routing tables up-to-date. When a node wants to re-join, it can simply try to connect to nodes from the host cache without contacting a bootstrap server. However, at least one node in the host cache must be active. Thus, the average *lifetime* of the addresses, i.e., the time the peers are connected to the P2P system using the same IP address and port, is an important factor for the performance of the host cache (see Section 4).

Random Address Probing. Another mechanism for decentralized bootstrapping is based on randomly probing for the addresses of active peers. This is done by sending join-request messages to randomly selected IP addresses and default ports, assuming that with sufficient high probability a peer is located at one of the IP addresses and will send a reply. This is based on the observation that the P2P systems that are currently deployed have several millions of users, as shown, e.g., in [15,16]. This holds in particular for the DHT-extension of BitTorrent (see Section 3) and KAD, the DHT-extension of eMule [17,18]. However, the time until the first peer is discovered heavily depends on the number of currently active peers. [19] shows that the number of eMule-peers in dial-up-networks is particularly high. Thus, to this end the probability of success can be increased by restricting the probing processes to IP addresses from such networks (*local random address probing*). By extensive measurement studies we determine the probability of success that can be expected currently in the BitTorrent DHT in Section 5.1. Furthermore, the time until a success depends on the probing rate, i.e., the number of join-requests sent per second, which may be limited by the available bandwidth and by network hardware, as we show in Section 5.2. Such factors are not considered in [19]. Last, the time depends on the number of peers using the default port, since probing multiple ports per peer obviously increases discovery time. We will elaborate on this factor in Section 5.3

Network layer mechanisms and standard protocols. Apart from application layer approaches mentioned, it might also be possible to use network layer mechanisms like multicast, anycast [20] or the service location protocol (SLP) [21]. These protocols could facilitate the bootstrap process. However, the information stored at the multicast or anycast routers or central SLP directory services raises scalability and robustness questions. Besides this theoretical issues, there is little support for these protocols on a global scale.

2.2 Related Measurement Studies

Since we perform an extensive measurement study to analyze the performance of distributed bootstrapping, we briefly recall related work in this area. Current studies of the KAD system have shown that between 3 and 4.3 millions of peers are connected at the same time and have examined the lifetimes of those peers [17][18]. However, since these studies do not focus on the bootstrap problem, lifetime is measured per P2P-specific node identifier assigned to each peer by KAD. That is, the overall lifetime of a peer is measured, not the lifetime of the addresses, so that no conclusion on the performance of local host caches can be drawn.

The lifetime of peers in the P2P systems Gnutella, KAD, and BitTorrent (in Tracker-mode) is measured in [16]. Though that work does not consider the DHT-extension of BitTorrent, results confirm some results obtained for the extension presented in this paper. The BitTorrent client Azureus uses an alternative to the DHT-extension of mainline BitTorrent client. It is examined by [15], which argues that the lifetime of peers is several hours, but does not provide detailed measurements.

In [22] a measurement study is presented that analyzes the lifetime of peers in Overnet, which is also a Kademlia-based P2P network like BitTorrent. The analysis is based on IP address and port and shows among other things that 50 % of peers in Overnet have a lifetime of 4,300 seconds. We measured 4,500 seconds for BitTorrent peers (cp. Section 4). Furthermore, we observed 2.1 % of the peers running more than one day under the same address while they measured a fraction of 2.7 %. Opposed to our work, no conclusion on the performance of distributed bootstrapping is drawn from the measurement results presented in [22].

Opposed to most related work mentioned so far we present a detailed study of connection lifetime based on the IP address and port and use it to draw conclusions on the performance of local host caches. Furthermore, we explore practical issues like randomly chosen ports or limits imposed by firewalls that are relevant for the performance of decentralized bootstrapping.

3 Decentralized Bootstrapping into the BitTorrent DHT

To analyze the performance of distributed bootstrap mechanisms, we use the DHT-extension of BitTorrent [23], since it is better documented than KAD, easing the experiments described in Sections 4 and 5. However, we state that most results of this paper likely hold for other systems such as KAD/eMule.

Bootstrapping into the BitTorrent DHT is based on centralized bootstrap servers with a well-known DNS name like `router.bittorrent.com`. We assume that this centralized mechanism is replaced by a combination of the two distributed bootstrap mechanisms described in Section 2.1: First, we use local host caches for fast re-entering the BitTorrent system after short periods of disconnection, ranging from, e.g., forced disconnections of DSL lines in the order of

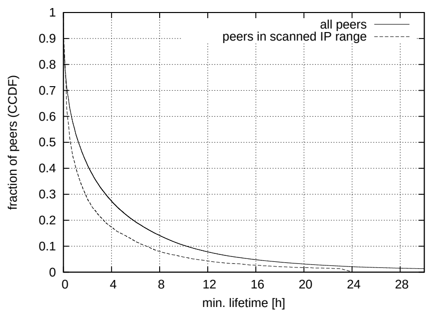

Fig. 1. CCDF of the (minimal) lifetime of the peers

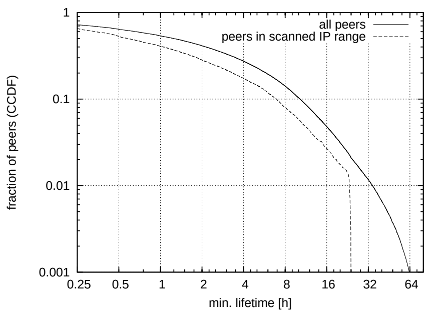

Fig. 2. CCDF of the (minimal) lifetime on log-scale

seconds to shutting down the PC when leaving for work in the order of hours. Since bootstrapping by local host caches may fail, e.g., after long periods of disconnection due to infrequent BitTorrent usage or even for first-time users, we secondly use random address probing as a fallback.

We analyze both mechanism by performing Internet measurements in the BitTorrent DHT in Sections 4 and 5. Furthermore, we consider that many typical peers use *dial-up networks* with DSL or cable modems. Typically, connections to dial-up networks use NAT routers and/or firewalls that may significantly limit the performance of random address probing as we shown in Section 5.2. Furthermore, many ISPs block the default BitTorrent UDP port 6881, forcing peers to switch to other ports with significant impact on random address probing as we show in Section 5.3.

4 Performance of Local Host Caches

To analyze the performance of local host caches we conduct a study of peer lifetimes in terms of addresses rather than P2P-identifiers as used in related work. We have measured peer lifetime by periodically scanning a certain number of IP addresses for BitTorrent peers using the ping mechanism of Kademlia. Therefore, we successively send Kademlia-`PING` packets to each selected IP address on the default port 6881 and wait for replies by BitTorrent DHT peers. Sending `PING` packets and listening for potential replies is performed asynchronously. That is, after sending one packet we continue to send `PING` packets to the next address immediately, without explicitly waiting for a reply. To all peers discovered during the periodical scan we send Kademlia-`PING` packets in intervals of three minutes. Peers that do not respond are timed out after 5 retries, i.e., 15 minutes conforming to the BitTorrent protocol specification [3]. Note that this methodology implies that the obtained results constitute lower bounds for the lifetimes, since peers may have already been connected to the BitTorrent system when they are discovered by the periodical scan. Thus, we refer to the measured lifetimes as *minimal lifetimes* as a lower bound on the performance of host caches.

Figure 1 plots the complementary cumulative distribution function (CCDF) of minimal lifetimes, i.e., the probability of a peer being still connected after a given period, as a function of the time. The figure shows two curves. The curve labeled "peers in scanned IP range" plots only peers within the IP range 84.128.x.x - 84.144.x.x. This range was scanned periodically with a scan rate of 300 PING packets per second (pkt/s) over a time of 27 days in February 2008. A scan of the complete IP range took about 58 minutes, i.e., the minimal lifetime for peers discovered in the second or later scans is under-estimated by at most 58 minutes. Peers that are online for less than 58 minutes may remain undiscovered by this approach. However, the curve shows that more than 8 % of the discovered peers have minimal lifetimes longer than 8 hours with a significant drop at about 24 hours. This is due to the fact that most ISP force a disconnection after this time span. The log-scale plot shown in Figure 2 illustrates this fact more clearly.

The second curve labeled "all peers" additionally considers nodes from outside the scanned IP range. Although we did not actively send Kademlia-`PING` packets to these nodes, Kademlia nodes exchange information about known peers, propagating the IP address and port of our measurement peer in the network. Within the 27 days we discovered 6,449 nodes within the scanned IP address range, compared to a total number of 238,628 nodes. Note that we likely underestimate the lifetime of nodes outside the scanned IP address range by more than 58 minutes. Nevertheless, Figure 1 shows that minimal lifetime for all peers exceeds minimal lifetime of peers in the scanned IP address range. In particular, more than 2% of the peers are reachable using the same address for more than 24 hours, some even for more than 7 days, implying that they are not located in dial-up networks.

In summary, the experiments show that 1.3 % of the peers in the scanned IP range had a lifetime of at least 23 hours. 0,1 % of all nodes had a lifetime of at least 64 h. Assuming a disconnection time of 64 h and a host cache size of 10,000 peers, the probability for successfully bootstrapping is almost 1. Due to the Geometric Distribution the expected number of peers that have to be probed until finding the first active peer is 1,000. We conclude that successful bootstrapping by local host caches is still feasible after a disconnection of several days. Nevertheless, bootstrapping is infeasible for low-frequency or first-time users. Thus, the next section explores whether local host caches can be complemented by random address probing.

5 Performance of Random Address Probing

To analyze the feasibility of random access probing approach, we investigate the expected latency using a direct Internet connection in Section 5.1. The limitations of typical DSL hardware that result in a significantly increased latency are discussed in Section 5.2. Furthermore, the distribution of the ports used by the BitTorrent clients further increases latency as shown in Section 5.3

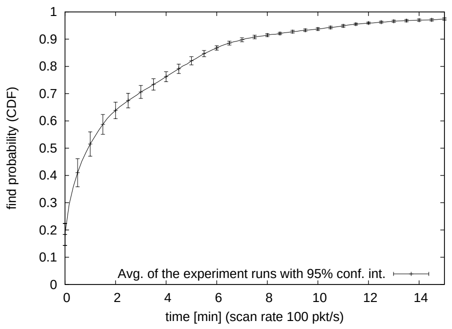

Fig. 3. Latency for finding a BitTorrent DHT peer by local random address probing

5.1 Latency of Random Address Probing

In a first experiment, we apply local random address probing by scanning a list of 29.014 Class-C dial-up networks (cp. [19]) using the Kademlia-PING mechanism as described above. We use a direct Internet connection from the network of our university. In the experiments we have completely scanned all 29.014 Class-C networks, i.e., more than 7 millions IP addresses. The experiment has been conducted in September 2007. We used the default UDP port 6881 of the BitTorrent DHT. The addresses have been scanned sequentially and the time elapsed between receiving two response messages has been logged. The scan rate has been adjusted to 100 pkt/s.

Recall that it is sufficient to discover a single BitTorrent DHT peer to join the system. Figure 3 shows the results as cumulative distribution function (CDF), i.e., it plots the probability of finding at least one peer in a particular time interval. We conducted 12 experiment runs that led to more than 5,500 values and plot the mean values together with the 95% confidence intervals. The figure shows that a peer can be located within 10 minutes with a probability of $\geq 94\%$. However, we expect that in the future the results will change to the worse to some extend, since one of the most popular BitTorrent clients $\mu Torrent$ does not use the default UDP port 6881 anymore, but a randomly selected port. Thus, the success probability of finding a peer at the default port will decrease. The current port distribution is outlined in Section 5.3.

5.2 Limits imposed by Standard DSL Hardware

The measurements in Section 5.1 have been conducted using a scan rate of 100 pkt/s. With a packet size of 107 Bytes, this requires an upstream bandwidth of about 86 kbit/s. Thus, a typical DSL-line with an upstream bandwidth of about 200 kbit/s should be able to handle the scan traffic easily. However, when we repeated the experiment using DSL-lines, we found that the upstream bandwidth is not the limiting factor. In fact, scanning performance is dramatically

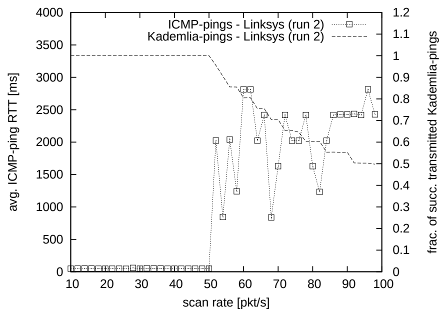

Fig. 4. Relation of ICMP ECHO delays and losses of Kademlia-packets

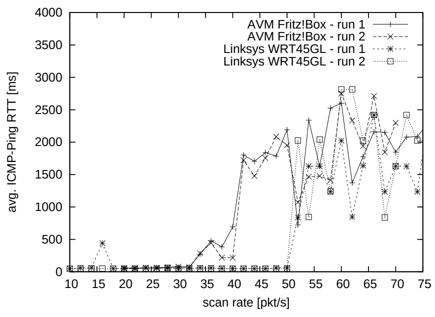

Fig. 5. Relation of ICMP ECHO delays and scan rate

limited by standard NAT-boxes with integrated firewalls. These boxes discard packets from hosts that have not previously been contacted from inside the home network by using *connection tables.* These tables may overflow even at low scan rates.

We found that the performance of some NAT boxes is heavily degraded, while others offer constant performance by overwriting connection table entries and, thus, discard valid responses from BitTorrent peers (Kademlia-PINGs) that do not arrive in time. The Netgear TA612V is an example of the second class. Our experiments have shown that it uses about 4,000 entries in the connection table. With a rate of 100 pkt/s, an Kademlia-`PING` reply has to arrive within 40 seconds to be accepted. Thus, such implementation does not impose a practical limit on the scan rate.

In contrast, overloading the router due to a connection table overflow does clearly limit the scan rate up to a complete loss of functionality as we observed for the AVM FRITZ!Box 7050, which is very popular in Germany, and the Linksys WRTL45GL, which is popular on a global scale.

To gain insight into this behavior, we traced the Kademlia-`PING` packets send on the WAN port of the Linksys NAT box. We found that discarding of Kademlia-`PING` packets is strongly correlated to the discarding of ICMP ECHO packets. Thus, we send ICMP ECHO REQUEST packets in parallel to the random address probing. Measuring the average ICMP-based round trip time (RTT) is a much more convenient indicator to detect the overload than tracing all Kademlia-`PING` packets, which might even not be possible for all deployment scenarios. To illustrate this effect, for the experiments shown in Figures 4 and 5, we scanned with different scan rates for 300 seconds. Subsequently, we stopped scanning for another 300 seconds to let the NAT box recover its normal functionality, before scanning with the next rate. The available upstream bandwidth was more than 200 kbit/s. The timeout for ICMP ECHO REPLY messages was set to 4 seconds. As shown in Figure 4, the fraction of successfully transmitted Kademlia-`PING` packets decreases dramatically with increasing scan rate. For a scan rate of 100 pkt/s only 50 % of the packets are successfully transmitted. At

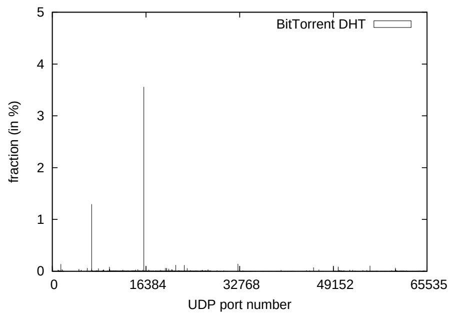

Fig. 6. Fraction of BitTorrent DHT peers per UDP port

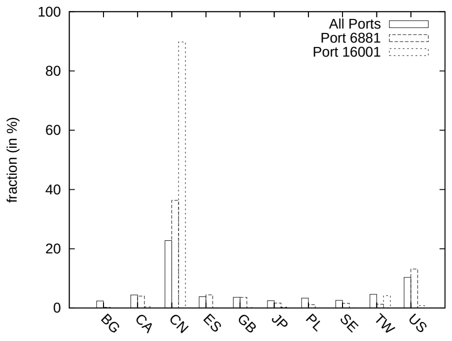

Fig. 7. Geographic distribution of BitTorrent DHT peers (top 15)

the same time the average ICMP-based RTT increases significantly. Figure 5 plots the average ICMP-based RTT as a function of the scan rate for both NAT routers. It shows that packet losses with the AVM FRITZ!Box 7050 occur at a scan rate of about 35 to 40 pkt/s. With a Linksys WRTL45GL this effect occurred at about 45 to 50 pkt/s.

The observed behavior of NAT boxes clearly puts a limit on the scan rate, increasing the time required to successfully detect a BitTorrent DHT peer reported in Section 5.1. Unfortunately, the time will further increase due to the changing usage of ports as we show in the next section.

5.3 Port Distribution of BitTorrent DHT Peers

The mainline client of BitTorrent used the UDP port 6881 as default port for the BitTorrent DHT until version 5.2.0. With the shift to version 6.0 the basis of the mainline client changed to *µTorrent* and also the default port is not used anymore. In fact the port is chosen randomly to avoid port filtering by ISPs. To quantify the impact of the port distribution, we conducted an additional experiment: experiment setup was as follows: First, we bootstrapped in the BitTorrent DHT with our test client. Second, we sent `FIND_NODE` packets for randomly chosen keys with an rate of 1 keys/s. Over a period of 8 days in April 2008 we recorded about 5.2 million addresses (IP address and port) from peers that sent a response to our `FIND_NODE` packets.

The analysis of the captured data shows that each of the 65,536 possible ports is occupied by some peers of the BitTorrent DHT with an average of 78 peers per port and a standard deviation of 9. Figure 6 shows the distribution as histogram. Besides the small peaks that are distributed across all port numbers, we see two major peaks at 6881 and 16001. The first peak corresponds to the former default port, but the reason for the second peak is not obvious. Our assumption is that the second peak is caused by a client that uses the port 16001 as default port. We were not able to identify the client exactly, since it does not transmit the client type in the exchanged messages. However, we were able to trace its geographical origin, as Figure 7 shows by the geographic distribution of peers from the top 15 countries for all ports, port 16001 and port 6881 respectively. The figure shows that 23 % of all peers are located in China and about 10 % in the US. Moreover, we can conclude that the questionable port 16001 is used by a client that is popular in Chinese speaking countries, because about 90 % of the peers are located in China about 4 % in Taiwan, and about 1 % in Hong Kong.

To summarize the findings on the port distribution, we calculate that its entropy is $H_Q = 15.17$. Due to the 2^{16} possible Ports, the maximum entropy is $H_{QMax} = 16$. Assuming there is only *one* default port, the resulting entropy would be $H_Q = 0$. Thus, the search space w.r.t. random address probing is drastically expanded by the port distribution. Hence, we try to find optimizing strategies and analyze therefore the interrelation between the used ports and IP addresses in more detail in the next section.

6 Improving Distributed Bootstrapping

In this section, we propose two mechanisms that can improve the performance of distributed bootstrapping building upon the observations on the distribution. The first mechanism improves performance of random address probing by exploiting the fact that peers use more ports than just one "standard port". This optimizing mechanism works without changing the port selection mechanism, i.e., is optimal for the current situation. With the same rational, we proposes a second mechanism for selecting "quasi"-random ports that can be applied to future peer implementations.

6.1 Using More than One Port

If the peers use more than one port as observed in 5.3, the strategy for random address probing can be potentially optimized. To illustrate that, assume the port distribution is known. Let $P_A(a)$ be the probability that a peer is run using a specified IP address a and $P_P(i)$ the probability that a Peer uses Port i. W.l.o.g. we assume $P_P(i) \geq P_P(j)$ for $0 \leq i < j \leq 65535$.

We assume that the peers will change during the process of random address probing, because the number of potential IP addresses is huge compared to the possible scan rate. Thus, the probability for finding a peer using an IP address a, $P_A(a)$, is independent of the history of probed IP addresses. In contrast we argue that the ports can be checked in a short period of time and therefore $\sum_{i=0}^{65535} P_P(i) = 1$. Using these assumptions we formulate an optimization problem that is given in Formula 1. The goal is to maximize the probability to find a peer after i tries and therefore determining the optimal number of ports m that have to be probed per IP address. The key idea behind this is to look for the probability that a peer was not found until the i-th try. The quotient in the exponent results from the fact that if multiple ports are probed, less IP addresses can be probed.

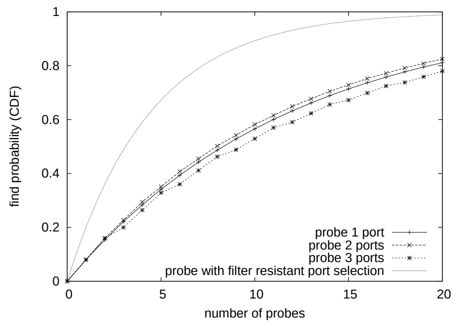

Fig. 8. Exemplary probability distribution for probing peers with multiple ports (see Section 6.1) and the probability distribution that results from applying the filter resistant port selection via hashing (see Section 6.2)

$$max\left\{1-\left(1-\left(P_a(a)\cdot\sum_{i=1}^{m}P_P(i)\right)\right)^{\frac{1}{m}}\right\} \qquad (1)$$

The optimal value for m can be easily determined by iterating over all possible values. To see that the solution of the optimization problem is not always trivial, consider the following exemplary probabilities $P_P(1) = 0.4$, $P_P(2) = 0.4$, $P_P(3) = 0.2$, $P(i) = 0$ for $i \geq 4$. and $P(A) = 0.2$. Figure 8 illustrates the resulting probability distributions for probing one, two, and three ports per IP address (For comparison reasons the filter resistant mechanism of Section 6.2 is also included.). Here, the optimal strategy is to probe two ports, i.e., $m = 2$.

Assuming that the ports are (exactly) uniformly distributed, we were able to prove that it is more efficient to probe all ports before using a different IP address. Due to space limitations the proof is omitted here, it can be found in [24].

The optimization problem can be used to determine the best strategy in currently deployed P2P networks with a given port distribution. In future networks random address probing can be assisted by an filter resistant port selection mechanism that is presented in the next section.

6.2 Filter Resistant Port Selection via Hashing

To reduce the entropy of the port distribution we develop a filter resistant port selection mechanism that is optimized for random address probing and, nevertheless, cannot be trivially filtered by an ISP. The mechanism is based on the fact that the ternary content addressable memory (TCAM) in routers is used by the ISPs for establishing access control lists. TCAM is a limited resource [25]. Thus, the ISPs filter ports on the basis of IP address ranges rather than individual IP addresses.

Based on this observation, we propose the following mechanism: The IP address of a peer and a so called virtual port ($port_{virt}$) are used to calculate the actual port ($port_{real}$) that is used by the peer:

$$port_{real} := h(\ ipAddr\ \otimes\ port_{virt}\ )\ mod\ 2^{16} \tag{2}$$

$h(x)$ is a consistent hash function like SHA-1. $port_{virt}$ can be arbitrarily selected, but has to be fixed for all peers.

The characteristics of this approach are on the one hand that the ISPs can not filter peers on large scale, because each peer uses a quasi random port that depends on the IP address. On the other hand w.r.t. to random address probing the IP address is known in advance and therefore the entropy of the port distribution is $H_Q = 0$ for random address probing. Compared to the optimum that can be achieved for randomly chosen ports using the Formula 1 from above, the impact is huge as it is shown in Figure 8.

Furthermore, this mechanism can be seamlessly integrated in existing P2P networks, because current implementations can already handle arbitrary ports. Thus, the existing P2P protocol does not have to be changed and new peers using filter resistant port selection are fully compatible with existing ones.

Conclusion

In this paper, we showed the difficulties of decentralized bootstrapping into peer-to-peer systems by extensive Internet measurements. We identified two critical issues, i.e., limitations by NAT routers and the port distribution used by the clients. Hence, we proposed two mechanisms to increase the performance of decentralized bootstrapping.

Gathering real-world data by extensive measurement over several weeks, we analyze a combination of two mechanisms, local host caches and random address probing. For local host caches, we found that they enable successful bootstrapping after short periods of disconnection. After longer disconnections, however, one must resort to other bootstrapping mechanisms. Unfortunately, random address probing does not help to this end, since practical problems imposed by typical NAT hardware and the usage of non-standard ports by the peers limit its performance. To cope with these problems, we proposed two mechanisms, probing multiple peers per host and filter resistant port selection.

References

1. Sharman Networks Ltd.: Website of KaZaA (2008) `http://www.kazaa.com/`.
2. Gnutella Developer Forum: Gnutella: A Protocol for a Revolution. (WWW-Publication) (2003) `http://rfc-gnutella.sourceforge.net`.
3. BitTorrent, Inc.: BitTorrent Website (2008) `http://www.bittorrent.com/`.
4. n.a.: eMule Website (2008) `http://www.emule-project.net`.

5. Skype, Ltd.: Website of Skype (2008) http://www.skype.com/.
6. Joost N.V.: Website of Joost (2008) http://www.joost.com/.
7. Srisuresh, P., Holdrege, M.: IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (Informational) (1999)
8. Karbhari, P., Ammar, M., Dhamdhere, A., Raj, H., Riley, G., Zegura, E.: Bootstrapping in gnutella: A measurement study. In: Proc. 5th Int. Workshop on Passive and Active Network Measurement (PAM 2004). (2004) 22–32
9. Cramer, C., Kutzner, K., Fuhrmann, T.: Bootstrapping locality-aware p2p networks. In: Proc. 12th IEEE Int. Conf. on Networks (ICON 2004). (2004) 357–361
10. Kan, G.: Chapter 8: Gnutella. In: Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. O'Reilly (2001)
11. Dämpfling, H.: Website of the Gnutella Web Caching System (GWebCache) (2008) http://www.gnucleus.com/gwebcache/.
12. Baset, S.A., Schulzrinne, H.G.: An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In: Proc. IEEE INFOCOM 2006. (2006) 2695–2706
13. Arak, V.: What happened on August 16. (WWW-Publication) (2007) http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html.
14. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like P2P systems scalable. In: Proc. ACM SIGCOMM 2003. (2003) 407–418
15. Falkner, J., Piatek, M., John, J., Krishnamurthy, A., Anderson, T.: Profiling a million user DHT. In: Proc. 7th ACM SIGCOMM Internet Measurement Conference (IMC 2007). (2007) 129–134
16. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: Proc. 6th ACM SIGCOMM Internet Measurement Conference (IMC 2006). (2006) 189–202
17. Stutzbach, D., Rejaie, R.: Improving Lookup Performance Over a Widely-Deployed DHT. In: Proc. IEEE INFOCOM 2006, IEEE (2006) 2884–2895
18. Steiner, M., En-Najjary, T., Biersack, E.: A global view of kad. In: Proc. 7th ACM SIGCOMM Internet Measurement Conf. (IMC 2007). (2007) 117–122
19. Conrad, M., Hof, H.J.: A Generic, Self-Organizing, and Distributed Bootstrap Service for Peer-to-Peer Networks. In: Proc. 2nd Int. Workshop on Self-Organizing Systems (IWSOS 2007). (2007) 59–72
20. Hinden, R., Deering, S.: IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard) (2006)
21. Guttman, E.: Service location protocol: Automatic discovery of ip network services. IEEE Internet Computing **3**(4) (1999) 71–80
22. Qiao, Y., Bustamante, F.: Structured and unstructured overlays under the microscope: a measurement-based view of two p2p systems that people use. In: Proc. USENIX Annual Technical Conference (ATEC '06). (2006)
23. Loewenstern, A.: BitTorrent Protocol Specification: BitTorrent Enhancement Proposal DHT Protocol. (WWW-Publication) (2008) http://www.bittorrent.org/beps/bep_0005.html.
24. Dinger, J.: Das Potential von Peer-to-Peer-Netzen und -Systemen: Architekturen, Robustheit und rechtliche Verortung. PhD thesis, Universität Karlsruhe (TH) (2009) ISBN: 978-3-86644-327-3.
25. Lakshminarayanan, K., Rangarajan, A., Venkatachary, S.: Algorithms for advanced packet classification with ternary cams. In: Proc. ACM SIGCOMM 2005, ACM (2005) 193–204