Distributed Bootstrapping of Peer-to-Peer Networks

Ryan C. Doyle[#]

Centre of ICT, Box Hill Institute
465 Elgar Road, Box Hill, Melbourne, Victoria, Australia
ryan@doylenet.net[#]

Abstract

A completely distributed, fault-tolerant peer-to-peer network also requires a bootstrapping mechanism that is equally distributed and fault-tolerant. This paper aims to investigate alternative bootstrapping mechanisms than those seen in common peer-to-peer networks of the current day. Existing and proposed bootstrapping mechanisms have been studied and compared against five requirements that indicate its suitability. The method proposed is a variation of the Local Random Probing mechanism for bootstrapping called Reverse Local Random Probing. Sampling part of the Internet indicating the potential size of the popular Gnutella network, a mathematical model is proposed showing the normal distribution demonstrating the amount of probes required to connect into the network. This is compared with the Reverse Local Random Probing mechanism, demonstrating the speedup gained. It is shown that there is a generous speedup that is gained from the Reverse Local Random Probing that makes is a possible candidate for bootstrapping of next generation peer-to-peer networks.

Motivation

I started researching alternative ways for bootstrapping peer-to-peer networks out of personal interest and a perceived lack of depth in research on the topic. I feel bootstrapping implementations used on current peer-to-peer networks are not clean enough to provide long-term solutions to the problem of completely distributing the bootstrapping process. The papers I have read about possible implementations for bootstrapping peer-to-peer networks have been useful resources and are referenced heavily throughout this paper. With that in mind, I have only discovered three papers published that deal directly with the issue of bootstrapping. It appears that the problem of how to coordinate a new peer to join a peer-to-peer network has been pushed aside for more interesting topics such as routing and searching in a peer-to-peer network.

This paper aims to present a new, distributed peer-to-peer bootstrapping method. It is designed to be as simple as possible and as clean as possible. I hope that it demonstrates the need and likely requirements for better methods of bootstrapping peer-to-peer networks.

1 Introduction

Peer-to-peer networks provide a distributed and fault-tolerant overlay network within a conventional inter-network. Most commonly the existing network is the Internet; but by its pure definition this can be any existing network. Ideally, there are no single points of failure in the network which would disrupt the operation of any other peers connected. Before a peer can connect to the peer-to-peer network, it needs to know how to locate the network. This process of discovering other peers and introducing itself into the peer-to-peer network is known as bootstrapping.

Using a central server as a meeting point contradicts the ideals of a peer-to-peer network and is not suitable for a pure, distributed peer-to-peer bootstrapping mechanism. The limitations of such a system were realised as early as 2002 when Napster was shutdown. Napster relied on centralised bootstrapping which was its ultimate pitfall. The Napster network was used primarily for sharing copyrighted material. File transfers were in no way routed through Napster-owned servers, but the facilitation of illegal transfers was adequate for the cessation of network operations.

The problems associated with bootstrapping peer-to-peer networks are often overlooked. Topics such as peer routing and search algorithms are often favoured. This paper aims to focus specifically on the bootstrapping problem and does not intend to go into detail of what happens *after* the connecting node discovers a peer in the network. All the issues of peer routing algorithms are not relevant if a peer cannot connect to the overlay network.

2 Requirements of a Distributed Bootstrap Protocol

Knoll, Wacker, Schiele and Weis propose five requirements for a bootstrap protocol in their paper, *Decentralised Bootstrapping in Pervasive Applications*[1]*.* These are summarised below. I propose a sixth requirement in addition to the existing five. It is felt that the existing five requirements do no go into detail about how clean the bootstrapping process should be and if it should to legal and ethical. The sixth requirement, *Lawfulness and ethical use* was proposed to counter illegal and unethical solutions that would only address the short term.

2.1 Bootstrapping Protocol Requirements

2.1.1 Robustness against Failure

This requirement is at the cornerstone of the bootstrapping protocol. The protocol should have no single point of failure and as a consequence should not rely on any form of central "bootstrapping server" as was the case with Napster. As outlined in the paper[1], a centralised model also discloses personal information about the IP addresses of peers that have used and are using the network. This requirement is still seen as very important.

2.1.2 Robustness against Security Appliances

This requirement mainly concerns home users with consumer-grade NAT firewall/routers. This requirement deals with the fact that NAT firewalls will not allow connections into the home network without port forwarding or Universal Plug-and-Play[2]. This requirement slightly contradictory as bootstrapping methods used by Gnutella require *some* peers to be unprotected from firewalls. Also peer-to-peer networks often require ports to be forwarded regardless of whether the client is bootstrapping or exchanging files. With the emergence of UPnP, the software can automatically setup a “hole” in the firewall. Therefore, this requirement is seen only to become exceedingly less apparent in the next generations of peer-to-peer applications.

2.1.3 Robustness against External Interference

This requirement is linked with the Robustness against Failure requirement. The bootstrapping protocol should be distributed sufficiently so that any external interference, insofar that no single entity would be able to shut the network down. This requirement is automatically fulfilled by ensuring that the bootstrapping process is entirely distributed. Possible groups that may want a peer-to-peer network shut down would be those dealing with copyright infringement if the network was used to share copyrighted material.

2.1.4 Efficiency

The bootstrapping process should be efficient so it minimalises the impact it has on the network. This requirement is fairly generalised so it is hard to gauge if the requirement is targeted towards the overall data exchanged in the bootstrapping process, the verbosity of the bootstrap or the CPU cycles used in addition to many other factors.

2.1.5 Scalability

The bootstrapping method proposed must also scale in terms of the number of peers in the overlay network as well as scale with the size of the initial communications network. The protocol should scale with larger and fewer numbers in both cases.

2.2 Proposed Extra Requirement

The original five requirements for a bootstrapping protocol had to be met to consider the bootstrapping method feasible. I propose that not all can be met with the same level of equality, but that they should be met as adequately as possible. There should also be an order of importance. This order of importance will change depending on *many* factors such as the estimated size of the overlay network, if the network will be used for illegal activity and if anonymity is of a prime concern; just to name a few. As well as these modifications with the terms of use, I also propose a sixth requirement.

2.2.1 Lawfulness and Ethical Use

The bootstrapping protocol should not break any laws or Acceptable Use Policies. Hijacking existing technology and modifying it to work as a bootstrapping protocol cannot be considered. The protocol should also be ethical. It should not impede on a users Internet experience or others. The extra requirement was conceived to avoid short term or otherwise *dirty* solutions to the bootstrapping problem.

3 Existing and Proposed Technologies

To design a new bootstrapping method also requires some background knowledge on the current bootstrapping protocols in use today. There seems to be two schools of bootstrapping protocols; first the approaches that are used on peer-to-peer networks today and then those that have been presented by various papers but are still only ideas or theoretical designs that are not fully developed for use in the current Internet architecture.

3.1 Current Bootstrapping Methods

3.1.1 Peer Cache

The peer cache approach to bootstrapping in its simplest form requires each client to have a cached database of peers that were connected to the network when the client last joined. If the client has not joined before and they have just downloaded the peer-to-peer application, a list of active IP addresses at the time of distribution is often packaged with the client as well. There are variations of this such as GWebCache[3] which implements a caching system on a web server. Instead of having a stale database locally on the host computer, the host would connect to the web server and get the freshes list possible. This centralisation is what peer-to-peer technologies try to avoid, but with a large number of these web cache systems operating and not being owned by a particular entity, these crude methods can be effective.

The peer cache method passes several of the requirements for a distributed bootstrapping protocol, but the possibility for a stale peer cache grows too large to be used in any small scale peer-to-peer deployment. This method is robust against failure as we are not relying upon a centralised bootstrapping model. It is not robust against security appliances as there must be some hosts who have a port open accepting the incoming bootstrapping request. This method is also robust against external interference as even if a majority of web caches were to be taken down, the client can still use its local peer cache. Peer cache is relatively efficient depending upon the context it is used in. In a typical peer-to-peer file sharing scenario utilised on a desktop computer, storing and processing a peer cache database is not an issue. If this were to be extended to mobile devices, this method would perhaps not be as effective. This method does not scale down well for smaller networks which limits the peer cache method to larger and well established networks.

3.1.2 Broadcast/Multicast

Broadcasting technologies such as multicasting are lightweight and are used commonly in discovery services such as UPnP[2] and Zeroconf[4]. Unfortunately multicast routing is usually blocked at the service provider. It is a matter of what service providers *allow* it instead of those that don't. Multicasting is great in a private enterprise network, but outside of this scope, it is not feasible to be used on the Internet.

Multicasting passes only two original requirements and three with the sixth requirement this paper proposes. It is robust against failure as it is not relying on any central service and is also very efficient. Unfortunately it is susceptible to external interference as multicasting on the Internet is severely regulated. Also, not all service providers have multicasting capability on their routers[5]. Nor does Multicasting work for a majority of users behind consumer grade NAT devices that do not have IGMP proxy capability[6].

3.1.3 Rendezvous Servers

Any bootstrapping protocol that relies upon a central meeting point can be considered a rendezvous server bootstrapping method. This includes the original Napster service and also encompasses technologies such as BitTorrent.

This method of bootstrapping is one of the weakest as it is not robust against failure. The rendezvous server simply needs to be taken offline either through system failure, a law authority or a denial of service attack and the network can no longer function. The centralised method also means that the network is susceptible to external interference. This method does have a number of advantages though, which is why it is used by BitTorrent. It is a very efficient method, potentially scalable and works with consumer NAT appliances.

3.2 Proposed and Emerging Bootstrap Methods

3.2.1 Random IP Probing

This method of bootstrapping involves randomly probing addresses until an active host of the peer-to-peer network is found. Once this initial host is found, the peer can communicate with this host to join the rest of the network. This method is *completely* distributed. It is robust against failure and is also robust against external interference. It is scalable and improves its efficiency with a higher density of nodes in the peer-to-peer network. Unfortunately the efficiency of such a mechanism is still extremely low even with a high number of peers.

This method of bootstrapping can be fine tuned to aid the efficiency drawbacks such as those presented by GauthierDickey and Grothoff[7]. Their method involves biasing where to start scanning using the SOA information to find consumer-grade rich IP addresses. Their theory is that common peer-to-peer applications will be more prominent in the domestic setting and therefore there is a greater chance of “hitting” a peer through scanning. Their theory is correct as we see through the generous increase in efficiency as shown in their paper.

3.2.2 Internet Relay Chat

A solution was proposed by the same paper that defined the five requirements for a bootstrapping protocol[1]. In this paper, Internet Relay Chat was seen to fit all of the criteria. The paper proposed that a client would connect to a large and well-known IRC network. Once connected, the client would then join a predefined channel.

The paper does not go into detail about how the client would the connect to another peer, but it can be assumed that the client would either issue a command and then various peers would then return IP addresses and ports or peers would continuously advertise IP addresses and ports to connect to. This method was seen to fit the original five requirements but I believe that IRC is not robust against external interference, failure or scalable for several reasons.

In terms of failure, the IRC method relies that the network will continue operating. It is still a central point of failure. It is true that the design of IRC is distributed and fault tolerant, but we are *not* using the technology for that factor. Anyone could setup a distributed IRC network. It is because IRC is used commonly for legitimate purposes that one could get away with using it as a bootstrapping mechanism for a small overlay network. If some peer-to-peer group were to setup and use an IRC network specifically for bootstrapping a peer-to-peer network and that peer-to-peer network engaged in swapping copyrighted material, there is nothing that would stop the IRC network being shut down by the appropriate law or copyright infringement authorities.

Extra steps need to be taken to ensure that the IRC bootstrapping method will not be blocked by the IRC network. The channel that the peer-to-peer network is using could easily be shut down. Even if the network had some form of mechanism to distribute the channel names, what is stopping an IRC network from also joining that peer-to-peer group and shutting down the channels as each new one is created and propagated? Because of this fact, the IRC bootstrapping method cannot be seen robust against external interference.

If a network the size of Gnutella were to use an IRC bootstrapping method, the load on the IRC network would be readily identified. How IRC deals with hundreds of thousands or millions of clients introduces more problems. The more that the network would be abused by bootstrapping, the more the blame would be put on the IRC network for aiding in sharing copyrighted material.

The extra requirement of *Lawfulness and Ethical Use* was proposed to counter this type of bootstrapping mechanism. IRC would work for a majority of cases, but is in no way a clean and pure solution. It would not be long before this method would be detected and countermeasures put in place for such a bootstrapping method.

3.2.3 A Generic Bootstrapping Network

Conrad and Hof[8] present the concept of a larger bootstrapping network. This bootstrapping network does not offer any services of a typical peer-to-peer network but rather is used to hand the client off to the requested peer-to-peer network. The theory behind this design leverages the fact that larger networks such and Gnutella and eDonkey will provide a bootstrapping service for smaller networks. Ideally, all peer-to-peer networks originally bootstrap off the larger bootstrap network. This would increase the overall size of the bootstrap network and means that technologies such as random probing would be more efficient.

Unfortunately the work does not propose any new methods to bootstrap. The paper recommends that clients use existing methods to bootstrap such as host caches and random probing.

4 What Popular Networks use Today

Peer-to-peer networks that are prominent today use at least one of the aforementioned bootstrapping methods. It is important to understand *what* is used to determine if the proposed bootstrapping method can be integrated into these networks. In some instances there is a severe lack of documentation about the bootstrap process of certain peer-to-peer networks. Research behind closed source protocols seems to be directed at how the protocol works and less about the specifics behind the bootstrap process.

4.1 Gnutella

Guntella was first designed by Justin Frankel and Tom Pepper of Nullsoft in 2000[9]. The protocol has grown over several revisions, currently at version 0.6. The project is an open effort and several clients exist to access the network. Fortunately, the protocol has been open from the beginning and has been analysed heavily in the scientific community.

The bootstrapping method is not defined in the official Gnutella protocol specification[12]. This document assumes that the client is able to locate and communicate with a peer. Therefore there are a few different methods of bootstrapping that Guntella uses because there is not a standard[13]. Most Gnutella clients have their own internal peer-cache. This peer-cache stores IP addresses that were active the last time the client was used. These peer lists have to be quite large to guarantee that at least one peer will be available for the client to join the network. In a network the size of Gnutella this is less of a problem, but still is quite inefficient.

Gnutella also implements a distributed rendezvous server model called, GWebCache[10]. GWebCache is a web-based caching system whereby a client can request a list of online peers from a number of GWebCaches. The lists are served through HTTP and are run by volunteers. Anyone can setup a GWebCache of their own and introduce it into the Gnutella network. This openness and flexibility allows the centralised rendezvous server model scale into a distributed rendezvous server model. As no single person operates these caches, the bootstrapping method that was centralised and weak against external interference becomes decentralised and robust against external interference.

With these advantages identified, there are also a vast number of problems. We are just shifting the problem instead of solving it. The client still requires the URL of a GWebCache in order to operate. A number of fairly active GWebCaches are usually hard-coded into the client. If these GWebCaches went down or were taken offline, any new clients that have not discovered other GWebCaches would not be able to connect. With the open attitude that anyone can run a GWebCache means the GWebCache load is not balanced correctly and servers can become overloaded[11].

4.2 FastTrack

FastTrack is the name given to the network used by the application Kazaa. It is considered a second generation peer-to-peer network protocol, first generation methods being networks such as Napster, and was created about a year after Gnutella. The FastTrack protocol is closed-source and therefore required the official Kazaa client to be used. The technical details about the bootstrap process are a bit scarce, but it appears that Kazaa also uses the peer-cache method[14]. The biggest limitation with the FastTrack network is the company that is behind the Kazaa client. Kazaa does not have any involvement with the network in terms of indexing or offering a bootstrapping service, but provides the client and is therefore facilitating illegal downloads. Due to this, the company has faced many law suits.

4.3 BitTorrent

BitTorrent[15] differs substantially in its distribution model compared to Gnutella and FastTrack. The BitTorrent protocol is merely a data distribution protocol. I will analyse how it has adopted an Internet-wide peer-to-peer context. BitTorrent separates the process of finding, bootstrapping and downloading content across several demarcation points. In order for a user to download a file, they have to search for that file that is indexed by a torrent index site. There are many of these sites available that index legal and illegal content. Once the file has been found, the user is required to download a torrent file, usually with the file extension, .torrent. This torrent file contains a hash of blocks that the data will be broken up into and also the URL of one or many trackers[16]. These trackers are the next points of contact. They coordinate the download and keep a list of peers that are active in the BitTorrent “swarm”. The BitTorrent client will then connect to peers in the swarm to begin the download.

The layered architecture is an advantage as it distributes the *responsibility*. Bootstrapping in a BitTorrent network could be considered to exist at two different points, at the indexer or the tracker. I believe that that bootstrap happens at the indexer. Without the indexer, a client does not know where to connect. This method of bootstrapping is still considered a rendezvous server. In the case of BitTorrent, the rendezvous point is a web paged served through HTTP. A user requires prior knowledge of where the website is located to be able to download the .torrent files. This method is not robust against failure as it only requires a tracker or index site to be unavailable for BitTorrent to be unusable.

In comparing BitTorrent to a network such as Gnutella, it has to be noted that the size of the overlay network in BitTorrent is much smaller. Small peer-to-peer networks are created for each torrent. There is no “single” BitTorrent network that could be taken offline.

4.4 eDonkey

eDonkey was originally run by a now defunct company called MetaMachine[18]. The client and protocol were free but closed source. Since its inception, the protocol has been reverse engineered. The official eDonkey servers have mainly been replaced by eserver[17], a free but still closed source eDonkey server; and the client replaced by eMule. The architecture of eDonkey’s network is more like a torrent tracker. You need to know the address of eDonkey servers to be able to connect to the network. These are usually available within online communities. It can be concluded that eDonkey uses a rendezvous server approach to bootstrapping. This is usually one of the weakest forms of bootstrapping due to the centrality of the service, but seems to be effective enough as there are many eDonkey servers distributed around the world.

4.5 Skype

Skype is a peer-to-peer voice over IP program and protocol. The protocol and client are both closed source, but the client is provided for free. The Skype network was designed by Kazaa in 2003 to be a decentralised peer-to-peer network[19]. It uses many concepts of the FastTrack network that was also designed by Kazaa such as super nodes. Apart from the login server that is used for accounting, the network is fully decentralised. It uses the peer-cache method of bootstrapping to locate a known super node in the network. Baset and Schulzrinne demonstrate that the number of super nodes cached is somewhere around 200[19].

5 Proposed Bootstrapping Method

The proposed bootstrapping method is a variant of Local Random Probing. Peers that have already joined the network will continue to scan for potential peers that are attempting to join. The aggressiveness of the scan can change once a client connects to the peer-to-peer network. This rate can be determined by the bandwidth available, the size of the network, and the amount of peers within the local range of the network as well as any other vectors that can be introduced.

The Gnutella network is used as a working example to demonstrate the advantages that this method may have. Gnutella is a good example to draw parallels with as it is quite a large network and better documented that any other peer-to-peer network.

5.1 Shift to Large-Scale Peer-to-Peer Networks

The original research proposal suggested that the bootstrapping protocol would be used for small-to-medium scale peer-to-peer networks. There is no fixed definition for what is considered small or what is considered medium sized, but my original intention of bootstrapping was a network of about 100 – 10,000 nodes. Through further research of current peer-to-peer networks and how they operate, it was concluded that there was less of a need to find a bootstrapping protocol that was fully fault tolerant for these smaller network sizes.

BitTorrent is a perfect example of such a small-to-medium scale peer-to-peer network. The amount of peers in a BitTorrent swarm can vary *dramatically* but is usually somewhere under the 10,000 mark per torrent. BitTorrent was identified to be a rendezvous server model, but it stands up where it would usually fail in a large deployment. Since BitTorrent doesn’t require the node to go to the same rendezvous server each time, regardless of the torrent file being downloaded, the rendezvous server model can be used. Only the peers downloading that torrent are affected by an outage, not everyone using the BitTorrent protocol.

5.2 Analysis of Gnutella

5.2.1 *Density*

The Gnutella network was used to get some real-world figures to the potential size of common peer-to-peer networks. The most important factor in determining the size of Gnutella is the *density*. The density is determined by finding out how many live Gnutella nodes exist in an overlay network.

$$n = \frac{N}{V}$$
$$where$$
$$N = number\ of\ Gnutella\ nodes$$
$$V = size\ of\ overlay\ network$$

Fig 1 – Density function for the Gnutella network

Instead of trying to determine the size of the entire Gnutella network, several samples were taken to determine the density. The samples were taken over several days from 28/10/2008 - 4/11/2008 at random times. Two different /16 IP blocks were scanned at these times to get a bigger sample size. Nmap[20] was used as the preferred port scanner. Nmap was invoked with the following command.

```
nmap -sT -P0 -T4 -n -p 6346 10.10.0.0/16 --open -oN 10.10.0.0_16_scan_6346_5_11_2008.txt
```

This forces Nmap to perform a TCP CONNECT scan (`-sT`), without sending a ping request (`-P0`), at an aggressive rate (`-T4`), without reverse DNS lookups (`-n`), on port 6346 (`-p 6346`) for the network 10.10.0.0/16. Actual consumer-grade DSL and cable IP ranges were used instead of 10.10.0.0/16. Each scan probed a total of 65536 potential Gnutella peers. Once the scans were complete, the output files would be searched for the amount of “open” text strings. This would determine how many hosts were active established a TCP session.

	IP Range 1	**IP Range 2**
29/10/2008 @ 8PM	15	22
1/11/2008 @ 12PM	10	15
3/11/2008 @ 2PM	9	13
5/11/2008 @ 11AM	6	15
Avg.	10	16.25
TOTAL AVG.	**13.125**	

Table 1 – Dates of probes and the number of hosts found for each probe

The reason we see a higher number of hosts on the 29th is most likely due to the time the scan was done. For the scan on the 29th, it was done at 8PM. It is assumed that more residential computers would be on and connected at night time, and used for the recreation that Gnutella can provide. What this means to the bootstrapping protocol is that it could potentially take longer to connect to the network during the day time.

$$n = \frac{13.125}{65563}$$
$$n = 0.0002$$

Fig 2 – Density function applied to Gnutella scan

This average can then be used to determine the density of the two /16 networks. Note that the density *will* be quite low. This is to be expected. Once we have the density, this number can be used to determine how many probes would be required in order to find 1 peer out of the 13.125.

5.2.2 *Probes required to find a peer*

The density is also the probability of hitting 1 out of the 13.125 hosts in the network. All we need is to connect to a single peer in order to connect to the peer-to-peer network. This can be graphed using the normal distribution function.

$$f(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}$$

Fig 3 – Standard normal probability density function

As well as this function, the mean and standard deviation are needed as well

$$\mu = np$$
$$\sigma = \sqrt{npq}$$

Fig 4 – Functions to find the mean and standard deviation

As noted before, the density is also interchanged with the probability. We can therefore determine the mean number probes required, *n*, in order to hit a single node

$$1 = n(.0002)$$
$$n = 5000$$

Fig 5 – Mean number of probes to hit a single node

This means that it will take on average, 5000 probes to find the first peer in a density of .0002. This can then be substituted for *u* as out aim is plot the distribution of the number of probes required for a single hit.

$$\mu = np$$
$$therefore$$
$$n = \frac{\mu}{p}$$

Fig 6 – Rearranging the formula lets us calculate *n*

We can then substitute *n* into the standard deviation formula and calculate it accordingly.

$$\sigma = \sqrt{npq}$$
$$\sigma = \sqrt{\frac{\mu p(1-p)}{p}}$$
$$\sigma = \sqrt{\frac{5000(.0002)(1-.0002)}{.0002}}$$
$$\sigma = 70.7$$

Fig 7 – Standard deviation is 70.7 when *u* is 5000

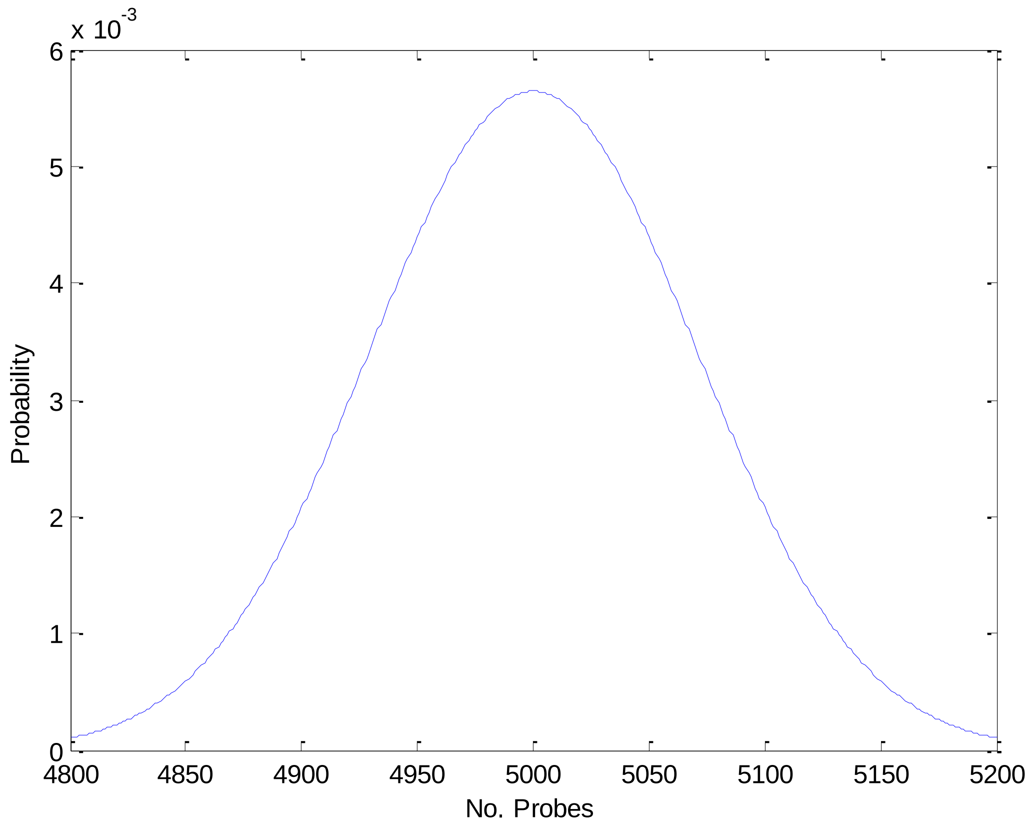

Fig 8 – Normal distribution plot.

From the graph, we can then calculate the maximum number of probes required to achieve a 95% confidence interval. This is calculated to 5116.

$$\Pr(.95) = 5116$$

Fig 9 – Maximum number of probes required to hit a single node at a 95% confidence interval

This maximum number is important as it determines the effective maximum number of probes a client will have to send in order to connect to the first node. The same graph can be plotted using the standard normal cumulative function. The standard normal cumulative function displays the same results as a normal distribution, but displays the probability as a running cumulative value

$$f(x) = \frac{1}{2}\left(1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right)$$

Fig 10 – Standard normal cumulative distribution function

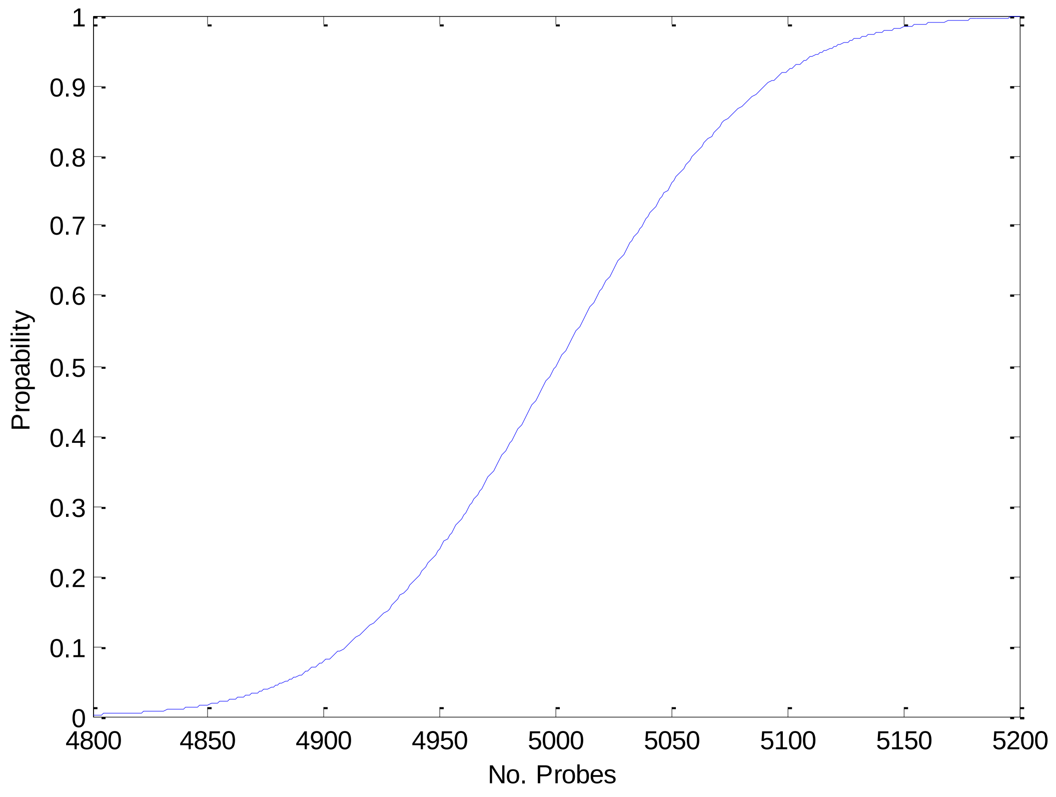

Fig 10 – Number of probes plotted against the probability

5.3 Reverse Local Random Probing

The Reverse Local Random Probing method this paper presents uses the same concept as local random probing, but allows for peers that are already connected to the network to continue scanning for possible new nodes. Before Reverse Local Random Probing can be fully explained, the concepts of Local Random Probing need to be covered

5.3.1 Local Random Probing

Local Random Probing assumes that the client wanting to join the peer-to-peer network will be a consumer-grade DSL or cable connection. If the client is on one of these connections, it is very likely that the address block they are on will also contain other DSL and cable users. Therefore, if the client were to use their IP address as some kind of starting point, there is a much greater chance of random probing succeeding.

The advantage with Local Random Probing is the greater possibility for a dense network, some of the traffic could be free if it is contained to the ISP's network and less waste of bandwidth as the amount of hops a packet will need to go is far less. This method is not foolproof and does not accommodate for users that are on IP ranges whose densities are extremely low.

This may seem like a waste of bandwidth and resources, but the traffic associated with a TCP SYN is extremely low. A single TCP SYN IP packet only contains 56 bytes. Using the Gnutella example, a mean amount of data used to bootstrap can be calculated easily.

$$bytes = 56(5000)$$
$$bytes = 280{,}000$$

Fig 11 – 280KB is required for a mean of 5000 probes

A slight modification to local random probing is proposed that could allow for a faster and more efficient bootstrap. Instead of using the clients IP address as a starting point and scanning up or down from that number, I propose that the client scans either side of their starting IP address. For example, if a client were to have an IP address of 10.0.0.120, the client should start scanning 10.0.0.121, then 10.0.0.119, then 10.0.0.122 and 10.0.0.118 and so on. The closer a client scans to its original IP address, the better. The closer the probe is, the greater the possibility that the traffic will be contained to the ISP network and hence quick and more efficient bootstrap. There is no need to scan on a per subnet basis.

5.3.2 Reverse Probing

The concept behind reverse probing is simple. Once a peer has connected and joined the peer-to-peer network, that peer continues to probe for more potential peers. The potential peers require that the peer-to-peer application is running and the required port is also available. After the peer has joined, the peer will then be intelligent towards the distribution of pervious scans, and which scans need to happen next in the local area.

5.3.3 Broadcast Domains and Reverse Intelligent Probing

The idea of broadcast domains comes into play when a client has successfully bootstrapped and is now performing the reverse probing function. When a client initially bootstraps, they will use their IP address as a starting point and scan both up and down their original IP address. Broadcast domains in the context of peer-to-peer networks attempt to apply concepts of broadcast domains found in layer 2 Ethernet switching. A client will be given a domain in which to probe. Usually this domain's middle point should float if not on, but near the clients IP address. The size of this domain should be inversely proportional to the density of the peer-to-peer network. This is when intelligence is required in the network that will allow it to gauge some sort of density and adjust the size of the reverse probing domain. This is where the terminology Reverse Intelligent Probing (RIP) is derived from.

I will again use the example of the Gnutella network to demonstrate how this concept works. The density of Gnutella was measured at .0002. This equates to a mean of 5000 probes to find 1 peer. Each peer that is connected to a peer-to-peer network will be given a domain of 5000 IP's to look for other potential peers. Ideally the middle point will be the peers IP address, but this is not a necessity. The peer will continue to probe the range of 5000 IP addresses over and over. If a client wants to connect to the peer-to-peer network that is in the range of one of these domains, the peer controlling this domain will probe for the client and then introduce it into the peer-to-peer network.

Broadcast domains can overlap if needed. A broadcast domain should always be greater than the inverse of the density. The important note to take is that intelligence will be built into the peer-to-peer network. Density could perhaps be determined on a per autonomous system, IP range or country basis.

Reverse Intelligent Probing conceptualises that the peer-to-peer network would be intelligent. It would be aware of the bootstrapping process and would store and distribute lists of broadcast domains to peers. There is the possibility with such an aware system that a client could inform what ISP that user was on and add the IP range of that ISP to the Reverse Intelligent Probing database. This avoids Reverse Random Probing from wasting probes on unused networks. A client may have to do a lengthy random probing scan the first time the peer-to-peer application used, but when that client connects and is introduced as a peer, it will tell the network to add its IP range to the list of IP ranges to scan. All these situations are possible if the peer-to-peer network is intelligent and actively involved in the bootstrapping process.

5.3.4 *Determining the Random Reverse Probing Efficiency Gains.*

When determining the speedup gained from reverse probing, it is being determined in the context of a single node, herein referred to as *node*$_A$ with all other nodes known as *nodes*$_X$. The efficiency gained is only measured by that single node and the amount of probes required. Also in determining the efficiency gained, it is being calculated from a pure random perspective. It is assumed that the network is not working *intelligently* as was suggested in the previous section. Finally, it is assumed that all nodes are probing at the same rate.

Firstly, the probability needs to be determined. This is calculated as shown below

$$\begin{aligned} p = {} & (node_A\ succeeds)(nodes_X\ don't\ succeed) + \\ & (node_A\ doesn't\ succeed)(nodes_X\ succseed) + \\ & (node_A\ succeeds)(nodes_X\ succeed) \end{aligned}$$

Fig 12 – Determining the probability of reverse random probing

$$p = \left(\frac{13.125}{65536}\right)\left(1-\frac{13.125}{65536}\right) + \left(1-\frac{13.125}{65536}\right)\left(\frac{13.125}{65536}\right) + \left(\frac{13.125}{65536}\right)\left(\frac{13.125}{65536}\right)$$

$$p = .0004005$$

Fig 13 – Probability of node$_A$ hitting a nodes$_X$ or getting hit by nodes$_X$.

With this probability, the mean number of probes can be determined.

$$\begin{aligned} \mu &= np \\ 1 &= n(.0004) \\ n &\approx 2497 \end{aligned}$$

Fig 14 – Mean number of probes required for node$_A$ to bootstrap

This is then used to find *n* when *u=2497* and hence find the standard deviation

$$\begin{aligned} \sigma &= \sqrt{npq} \\ \sigma &= \sqrt{\frac{2497(.0004)(1-.0004)}{.0004}} \\ \sigma &= 49.96 \end{aligned}$$

Fig 15 – Standard deviation for reverse random probing

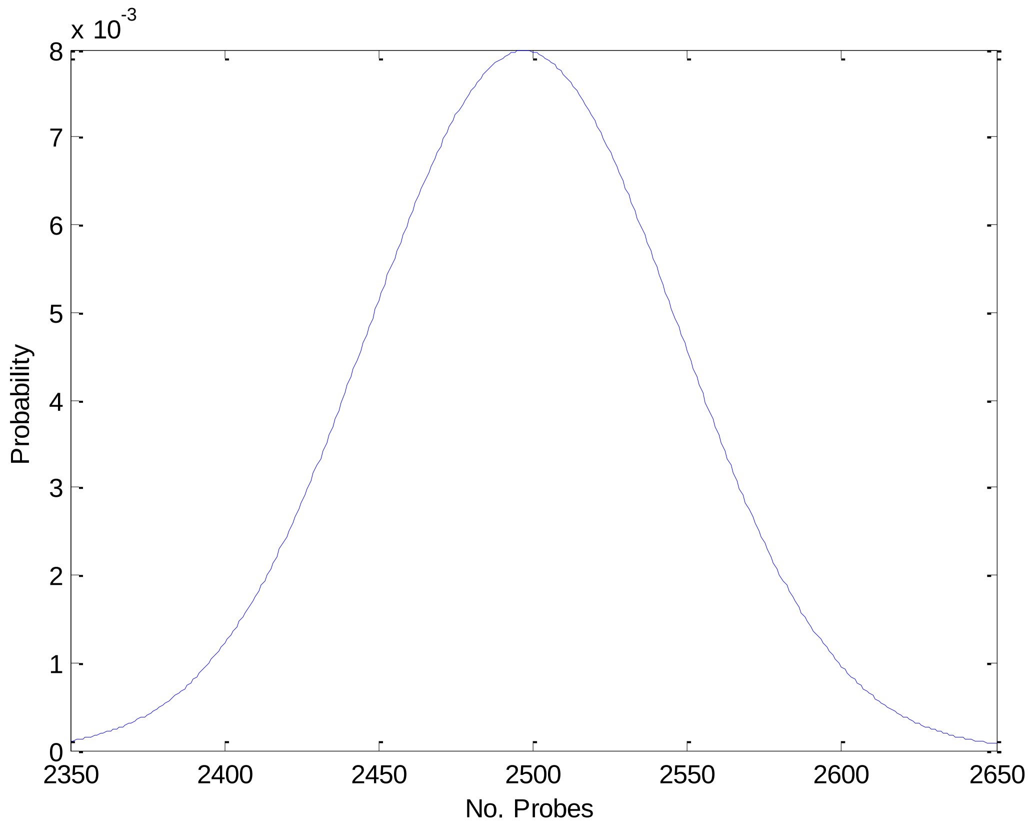

Fig 16 – Standard normal probability density function of reverse random probing

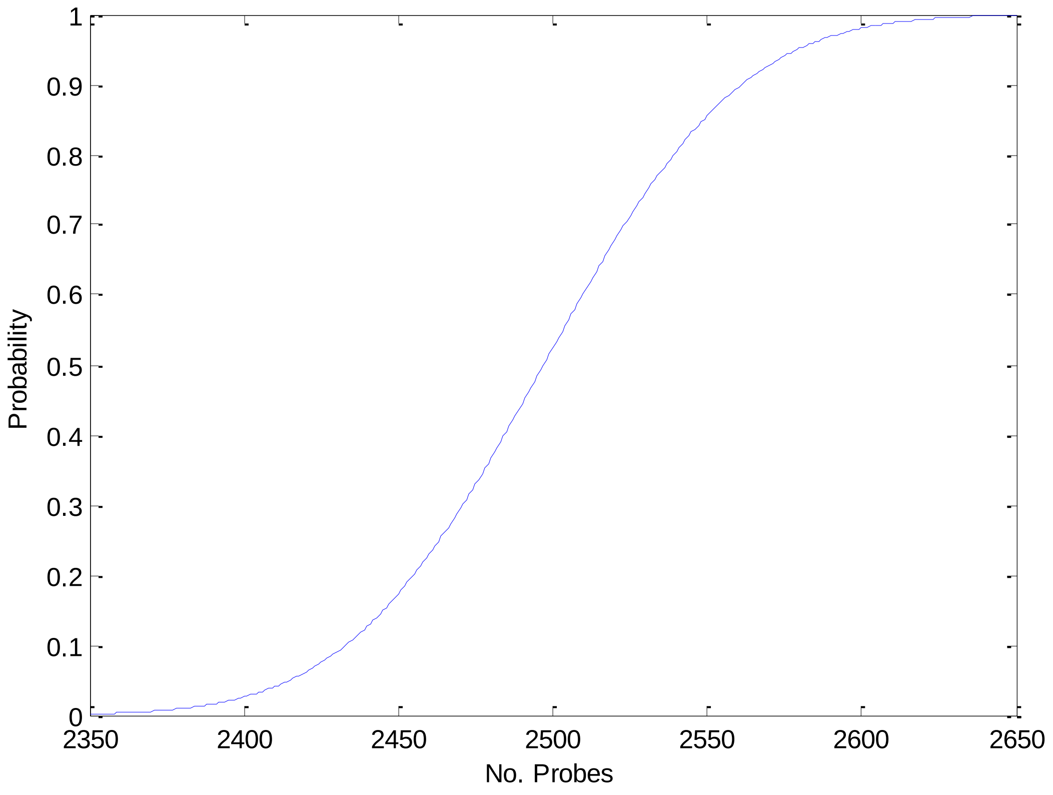

Fig 17 – Standard normal cumulative distribution of reverse random probing

$$\Pr(.95) = 2579$$

Fig 18 – It takes 2579 probes with a 95% confidence interval

The overall speedup with reverse random probing is about 100%. It will take twice the number of probes to bootstrap if nodes$_X$ are not probing back. This speedup is generous, but the extra bandwidth used by nodes$_X$ is substantial.

5.3.5 Bootstrapping From Scratch or Complete Failure

Many bootstrapping protocols require that the network is already well established. This section discusses theoretical speed up gained if the reverse random probing method were to be used to start a peer-to-peer network from scratch or after complete failure. The reverse random probing method could be used in a scenario where all other bootstrapping methods would fail. The mathematics involved is beyond the scope of this paper, but samples and possible solutions are proposed.

The network is considered converged when all nodes can communicate with each other, either directly or routed via another node. As a starting point, the birthday problem[21] can be used to determine the amount of probes required for any one node to probe any other. The birthday problem can be used in the context of a collision problem, whereby a collision of any 13.125 nodes in the 65536 address space is considered a success.

$$p(n;d) \approx 1 - e^{-(n(n-1))/2d}$$
$$p(n;d) \approx 1 - e^{-(13.125(13.125-1)/2(65536)}$$
$$p(n;d) \approx .00121$$

Fig 19 – Generalisation of the birthday problem

$$\mu = np$$
$$1 = n(.00121)$$
$$n = 826$$

Fig 20 – A mean of 826 probes will be required for any two nodes to find each other

This generalisation only applied for the first two nodes to find each other. To find the probability for the entire network to converge requires the use of multivariable statistics which are also beyond the scope of this paper. It is also assumed if we are using Reverse Intelligent Probing, when the nodes connect to each other, they will also share what address ranges they have scanned so that there is no need for overlap. These details make calculating the convergence time extremely difficult.

We could assume that convergence would be achieved in a maximum probes of n<826(13.125)

$$n(\max) \le 826(13.125)$$
$$n(\max) \le 10{,}842$$

Fig 21 – Possible maximum convergence time

5.4 Benefits of Reverse Random Probing

Reverse Random Probing and Reverse Intelligent Probing are ideal candidates for a fully distributed, fault-tolerant bootstrapping protocol. All other known bootstrapping methods in use today are not completely distributed. It is believed that random probing is the *only* pure method for bootstrapping peer-to-peer networks. It is not relying on a common point of contact to discover other peers. It does not rely on any external services. It does not rely on the peer-to-peer network to be fully converged. For example, half of the network could be unavailable and a new peer could still successfully bootstrap; the only effect is that it would take twice as long to bootstrap. There would be no "fatal" number of unavailable peers that would stop the bootstrap process.

5.5 Limitations of Reverse Random Probing

Reverse random probing and its variants do have some limitations. The major concern is the amount of probes required for smaller networks. This number grows at the inverse of the density. For a Gnutella sized network, reverse random probing is a possibility. For a network of 2000 nodes, the number of probes required is simply far too great. Another drawback is that the bootstrapping protocol has to use a static port. When probing for peers, the client needs to probe on a certain port. It is possible that an ISP may choose to block this port. For an end-user to see the benefits associated with reverse random probing, the bootstrap port needs to be forwarded to the running application. This is less of a concern with the emergence of UPnP, but still a consideration in analysing the effectiveness of reverse random probing.

6 Results

The results gained from reverse random probing show a speedup of about 100%. The extra bandwidth used has to be considered to determine if this method of bootstrapping would be appropriate which would be different depending on the peer-to-peer network.

	Local Random Probing	**Reverse Random Probing**
Mean probes required	5000	2496
Standard Deviation	70.7	49.96
95% Confidence Interval	5116	2579

Table 2 – Comparison of local random probing compared with reverse random probing

It was explained that the greatest speedup of Reverse Random Probing or Reverse Intelligent Probing would be to bootstrap a peer-to-peer network after complete failure or if the peer-to-peer network is new and not well established. Both situations have the same requirements.

7 Conclusion

Reverse Random Probing and its variant Reverse Intelligent Probing provide some theoretical solutions to the problem of bootstrapping in a peer-to-peer network. They solve this issue by not depending on *any* hard-code value which is a problem with many other methods.

Peer cache methods require many IP addresses or URLs to be hard-coded into the application upon distribution. Rendezvous server models also require a known entry point. Multicast and broadcast technologies seen in the LAN environment are not suitable for deployment as a bootstrapping protocol on the Internet. Bootstrapping using IRC also requires a well known entry point that needs to be hard coded into the client

Reverse random probing solves these issues by not storing any information about the state of the network. Modifications could be made to the bootstrap process to include a caching system as well. This would mean that a client attempted to connect using the last IP addresses that it knew were active. If the client could not connect using these IP addresses, then it would perform a local random probe, if not already picked up by a peer performing a reverse probe over that domain.

After analysis of the performance gains, such a bootstrapping method could be implemented to the scale of the Internet, but I feel that Random Reverse Probing and Random Intelligent Probing are still ideas that require some maturing before we can consider them to replace current bootstrapping technologies.

It is no denying that current techniques do work. Gnutella is a prime example displaying how the combination of peer-cache and rendezvous servers work, but it is still not without its disadvantages. BitTorrent is another good example of how distributing the responsibility over several demarcation points improves the resilience against external interference. With peer-to-peer networks seen as a favoured software distribution model, it will be interesting to see how next generation peer-to-peer networks solve the continuing problem of bootstrapping.

Acknowledgments

A special thank you to Sudipto Mitra for his help with the probability density functions. Thank you to Umesh Patel for supervising this research paper.

References

1. M. Knoll, A. Wacker, G. Schiele, T. Weis, "Decentralised bootstrapping in pervasive applications" in *Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, New York, United States, March 2007.
2. UPnP Forum, http://www.upnp.org/, 2008
3. Hanuke Dämpfling, http://www.gnucleus.com/gwebcache/newgwc.html, 2008
4. Zero Configuration Networking, http://www.zeroconf.org/, 2008
5. M. Handley, J. Crowcroft, "Internet Multicast Today". [Online]. Available: http://www.cisco.com/web/about/ac123/ac147/ac174/ac198/about_cisco_ipj_archive_article09186a00800c851e.html
6. D. Wing, T. Eckert, "IP multicast requirements for a Network Address Translator(NAT) and a Network Address Port Translator", Request For Comment 5135, February 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5135.txt
7. C. GauthierDickey, C. Grothoff, "Bootstrapping of Peer-to-Peer Networks" in *The Third International Workshop on Dependable and Sustainable Peer-to-Peer Systems*, Turku, Finland, July 2008.
8. M. Conrad, H-J. Hof, "A generic, self organising and distributed bootstrap service for peer to peer networks" in *Second International Workshop of Self-Organizing Systems*, The Lake District, United Kingdom, September 2007.
9. Gnutella, http://en.wikipedia.org/wiki/Gnutella, October 2008
10. Hanuke Dämpfling, http://www.gnucleus.com/gwebcache, October 2008

11. P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. Riley, E. Zegura, “Bootstrapping in Gnutella: A Measurement Study” in the *5th Passive & Active Measurement Workshop*, Antibes Juan-es-Pins, France, April 2004.
12. Patrick Kirk, RFC-Gnutella, http://rfc-gnutella.sourceforge.net/index.html, October 2008
13. Bootstrapping – Limewire, http://wiki.limewire.org/index.php?title=Bootstrapping, October 2008
14. Stephanie Watson, “How Kazaa Works”, October 2008. [Online]. Available: http://computer.howstuffworks.com/kazaa3.htm
15. BitTorrent, http://www.bittorrent.com/, October 2008
16. BitTorrent tracker, http://en.wikipedia.org/wiki/BitTorrent_tracker, October 2008
17. eD2K-Serverboard, http://www.ed2k-serverboard.de/index.php, October 2008
18. Metamachine.com, http://www.metamachine.com/, October 2008
19. S. A. Baset, H. Schulzrinne, “An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol”. Submitted paper, September 2008. [Online]. Available: http://arxiv.org/ftp/cs/papers/0412/0412017.pdf
20. Nmap, http://nmap.org/, October 2008
21. Eric Weisstein, “Birthday Problem”, November 2008. [Online]. Available http://mathworld.wolfram.com/BirthdayProblem.html