Final Project

Biased Random Periodic Switching in Direct Connect

Author	Yoran Heling
Supervisor	Djoerd Hiemstra
Second Supervisor	Georgios Karagiannis
Third Supervisor	Robin Aly

University of Twente
Master of Science in Embedded Systems
Faculty of Electrical Engineering, Mathematics and Computer Science

June 10, 2014

Preface

This report is written as part of my final thesis during the Embedded Systems master at the University of Twente, and represents the research performed on the topic of peer selection in Direct Connect.

Daily supervision was in the hands of Djoerd Hiemstra of the Databases group and Georgios Karagiannis of the Design and Analysis of Communication Systems (DACS) group, who both provided valuable feedback during the project. Robin Aly offered valuable feedback on the final draft of this report.

I would also like to thank all the people who actively contributed to the Direct Connect protocol and its implementations. This project would not have been possible without the continued volunteer efforts behind Direct Connect.

Contents

Abstract

In a distributed Peer-to-peer (P2P) system such as Direct Connect, files are often distributed over multiple source peers. It is up to the downloading peer to decide from how many and from which source peers to download the particular file of interest. Biased Random Period Switching (BRPS) is an algorithm, implemented at the downloading peer, that determines at what point to download from which source peer. The number of source peers that a downloading peer downloads from at a certain point is called the Degree of Parallelism (DoP). This research focussed on implementing BRPS in an existing Direct Connect client and comparing the downloading performance against an unmodified client.

Two implementations of BRPS in Direct Connect have been made. A *simple* implementation that follows the original BRPS algorithm as closely as possible, with minor modifications that were required to ensure that the downloading process would not get stuck on an unavailable source peer. An *improved* implementation has also been made with slight modifications to the original BRPS algorithm. The improved implementation incorporates two improvements to ensure that the DoP does not drop below its desired value in the face of unavailable source peers.

The original client and the two BRPS implementations have been evaluated in a controlled Direct Connect network with 50 downloading peers and a variable number of source peers. The source peers have been configured to throttle their available bandwidth to an average of 500 KB/s, and following a realistic bandwidth distribution based on measurements from the Tor P2P network. The experiments consisted of all downloading peers downloading the same file at the same time, and taking measurements on the side of these downloading peers.

Four experiments have been performed, with one varying parameter in each experiment. The size of the file being downloaded was varied between 100 MB and 1024 MB in the first experiment, the second experiment varied the DoP between 1 and 15. The number of source peers was varied between 10 and 100 in the third experiment, and in the last experiment between 0% and 80% unavailable source peers were added to the network.

In all experiments, both BRPS implementations performed close to the optimal average download time, and were consistently faster than the original client by a factor of 2 to 5. In the last experiment, the improved BRPS implementation did keep the measured DoP closer to its desired value than the simple implementation, but this has not resulted in a significant difference in the measured download times.

Chapter 1

Introduction

Peer-to-peer (P2P) systems have seen an increase of attention since the early 2000s. Starting with the popular MP3 downloading application Napster in 1999, many others have followed and improved on its design. P2P applications that are widely used today include BitTorrent [21], Skype [15], Gnutella [40] and Direct Connect [3].

A P2P system is a distributed network of peers, working together to provide a set of services to the user, such as scalable file distribution, instant messaging, music streaming and/or searching.

1.1 Direct Connect

This research focuses on the Direct Connect file sharing network. It is a hybrid Peer-to-Peer file sharing application developed in the late 1990s, and provides the following features to its users:

User management: Everyone can obtain a listing of other users on the network. This list includes some basic information such as a nick name, upload speed, a user-provided description and email address.

Instant messaging: Chat messages can be broadcasted over the network, forming a large chat channel in the same spirit as in Internet Relay Chat [37]. Private messaging between users is also possible.

File sharing: Users can select one or more directories on their local hard drive to share with other users on the network. All files are idenfitied by a cryptographic hash function (Tiger Tree Hash [29]) over their contents.

File searching and browsing: Users can search the network for files shared by others. It is also possible to browse through the shared directories of a specific user.

This report focuses on the file sharing aspects of Direct Connect.

The architecture of a Direct Connect network is conceptually simple, and consists of three main entities: *Clients*, *Hubs* and *Hublists* (Figure 1.1). A Hub is a TCP (Transmission Control Protocol) server that clients can connect to, and it provides services to allow clients to discover and communicate with each other.

A single Client can be connected to multiple hubs simultaneously. A Hublist is a central entity, commonly hosted on a web server and accessed over HTTP (Hypertext Transfer Protocol), that provides a list of publicly available hub addresses. A Hublist is not required for the functioning of the Direct Connect network, they merely help users with finding new hubs to connect to.

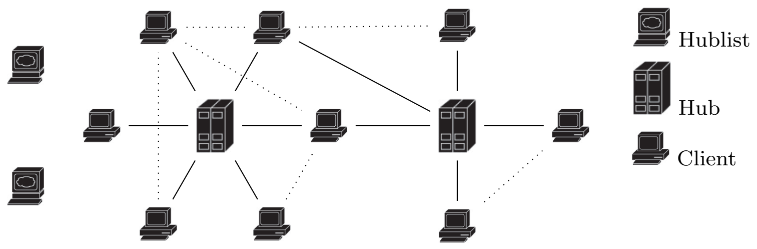

Figure 1.1: Architecture of a small Direct Connect network with two hubs and two Hublists. Two clients are connected to both hubs. Some clients that share the same hub have a direct connection with each over, indicated with a dotted line.

Clients in the Direct Connect network can download and upload files from or to other users in the network. In this report, the term *Downloading Peer* is used to denote a client that is downloading one or more files, and the term *Source Peer* is used for a client that is offering one or more files to other users in the network. Some clients are configured to perform the role of either a downloading peer or a source peer, but the majority of clients perform both roles simultaneously.

The underlying protocol used by Direct Connect is called Advanced Direct Connect (ADC) and is specified in [44]. The protocol is designed to be extensible, and common extensions to the protocol are defined and specified in [46].

1.2 File Downloading and Peer Selection

A single file is often available from multiple source peers. This gives the downloading peer the opportunity to decrease the file download time by selecting faster source peers and/or by downloading from multiple source peers simultaneously (parallel downloading).

The downloading peer has the freedom to download any file part (chunk) from any source peer at any time. It is thus the responsibility of the downloading peer to decide *when* to download from *which* source peers, and *what* chunks to request from the source peers.

The number of source peers from which a downloading peer will download at one time is called the *Degree of Parallelism* (DoP). Intuitively one might expect that downloading from all available source peers at the same time will result in the shortest download time, but doing so will negatively impact the scalability of the network [49], and further increasing the DoP above a certain threshold will not result in a significant improvement in the download time [17]. The maximum DoP used by downloading peers is therefore limited.

Given this constraint, it is important that the downloading peer selects the right source peers to download from in order to minimize the download time. This selection is complicated by the distributed nature of P2P systems, because the downloading peer does not have a holistic view of the network and does not know in advance how much bandwidth each source peer can provide.

A variety of peer selection mechanisms have been proposed in literature. This research will focus on the peer selection mechanism proposed by Chiu [17, 19], which will be referred to in this report as *Biased Random Periodic Switching* (BRPS). BRPS differentiates itself from other peer selection mechanisms with time-based switching and probability-based source peer selection.

1.3 Contributions

The contribution of this report is to evaluate whether and how BRPS can be integrated into the Direct Connect network, and to compare it against the peer selection mechanism currently employed in Direct Connect. The main research question is as follows. **Can the BRPS algorithm help reduce the download time in Direct Connect?**

This report attempts to answer this question by dividing the problem into the following sub-questions.

1. How can the BRPS algorithm be integrated in Direct Connect?
2. Which characteristics can improve the BRPS performance?
3. How can a Direct Connect client that uses the BRPS algorithm be analyzed and evaluated?
4. Does BRPS reduce the download time compared to the currently implemented peer selection mechanisms?

The open source ncdc [7] Direct Connect client is used in this research to evaluate BRPS.

This report is organized as follows. Chapter 2 gives an overview of existing work in peer selection and follows with a more detailed description of Direct Connect. Chapter 3 describes the challenges of integrating BRPS into a Direct Connect client, and the solutions are offered in Chapter 4. The experiments and results are described in Chapter 5, followed by the conclusions and recommendations for future work in Chapter 6.

Chapter 2

Preliminaries and Related Work

This chapter provides a quick overview of the existing peer selection mechanisms discussed in literature and offers a more detailed description of the BRPS algorithm. The second part of this chapter describes a few relevant functions of the Direct Connect protocol in more detail and explains how existing clients implement peer selection.

2.1 Peer Selection Mechanisms

Peer selection mechanisms can be categorized based on the kind of information they utilize to select source peers. The information that is used in the peer selection mechanism determines the variables that each peer selection mechanism attempts to optimize and also determines whether a specialized infrastructure is needed to supply this information. Three categories of peer selection mechanisms are discussed in the sections below.

2.1.1 Topology Estimation

The goal of topology estimation techniques is to make the application aware of the underlying network topology. This allows the downloading peer to sort its list of source peers based on an approximate cost function, where the cost being measured may be the number of required routing hops to reach the destination or the geographical distance with each source peer.

Aggarwal et al. [13, 43] propose to implement topology estimation at the side of the Internet Service Provider (ISP), since the ISP is already aware of the underlying network routes. A downloading peer can then send its list of source peer addresses to the ISP and the ISP will respond with an ordered list of source peers to download from. The IETF Application-Layer Traffic Optimization (ALTO) Working Group [2] has focussed its research on this peer selection mechanism and has developed a protocol standard for the communication between P2P applications and ISPs [39]. A criticism of this technique is offered by Paitek et al. [38], who argue that the incentives of the ISP do not necessarily align with the goal of offering shorter or faster routes.

Since infrastructure for peer selection in ISPs is not ubiquitous, Ono [20] proposes to make use of the existing infrastructure of Content Delivery Networks (CDN) to aid in peer selection. A large-scale CDN such as Akamai [1] or Limelight [6] has servers all around the globe, and distribute the load over these servers by giving a different Domain Name System (DNS) response to each client based on its location. The goal of this DNS-based redirection mechanism is to ensure that each client is directed to a server with close proximity to the client. This existing infrastructure can be re-used by letting all peers in the P2P network query one or more CDNs and have them exchange the responses they receive. If the CDN gives a similar response to two peers, then these peers are likely in close proximity to each other, and the downloading peer can prioritize downloading from these closer source peers.

Both of the aforementioned proposals rely on the existence of one or more centralized services. Two proposals for a distributed alternative have been proposed as Global Network Positioning (GNP, [24]) and Vivaldi [22]. These proposals work by introducing a coordinate system, where every source peer is assigned an absolute coordinate in a fixed geometric space. The distance between the coordinates of two peers then represents the approximate network distance between these peers. Each peer maintains and adjusts its own coordinate based on Round-Trip Time (RTT) measurements with other peers in the network (Figure 2.1). Some problems with this system have been discussed by Ledlie et al. [32], and include slow reaction time for newly connected peers and inaccuracies caused by drift and triangle inequality violation [30].

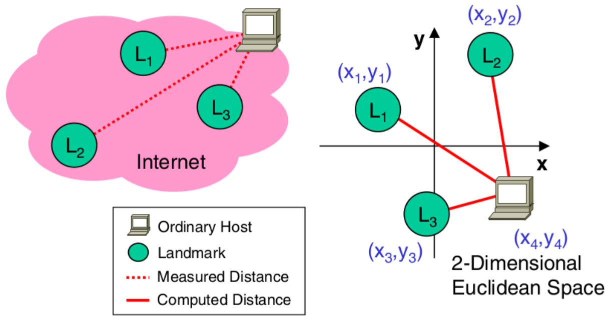

Figure 2.1: A Global Network Coordinate system using landmarks to determine a peers' coordinate. Copied from [24].

All of the above mechanisms optimize for close network proximity, but there is no guarantee that a close source peer will offer more bandwidth than one that is far away. It is possible that better downloading performance can be achieved when downloading from source peers that are further away from the downloading peer.

2.1.2 Incentive-Based Peer Selection

The problem that incentive-based peer selection mechanisms attempt to solve is the problem that, in a P2P network, individual peers may not have a strong *incentive* to offer their bandwidth to the rest of the network. Yet it is the basic foundation of P2P systems that enough peers share their resources with other peers. The goal of incentive-based peer selection mechanisms is thus to reward peers that share their bandwidth and, in effect, to punish peers that do not contribute to the network.

A possible solution to incentivise uploading is to adjust the peer selection process to favour peers that regularly share their resources with others. By doing so, these peers will get priority when they themselves request something of the network. The most widely known example of incentive-based peer selection is BitTorrent [21], but many alternative algorithms have been described in literature [45, 12, 48, 42, 47, 35].

In a Direct Connect network, it is the responsibility of the Hub administrator to set the policies for all peers connected to that hub and to monitor these policies. The policies implemented in a Hub include rewarding peers that contribute to the network and to punishing peers that do not. Since incentives are already part of the Direct Connect network, incentive-based peer selection mechanisms have not been considered in this research.

2.1.3 Performance Estimation

Performance estimation techniques perform network measurements to estimate source peer bandwidth and use that information to influence the peer selection. Unlike the previously mentioned peer selection mechanisms, performance estimation techniques attempt to optimize bandwidth utilization and download time more directly, and are generally oblivious to network proximity or incentives.

Many P2P networks, including Direct Connect, already provide some information that may predict the performance of source peers. This information includes fields such as the current load of the source peer, whether the peer is behind a firewall or not, the number of shared files and a user-provided indication of the maximum bandwidth. Not all of these fields offer any useful insight into the performance and some fields may be inaccurate. In order to still be able to make use of this information, a machine learning approach is proposed as Adaptive Peer Selection [16]. The downloading peer gathers as much information about the source peer as possible and uses this information in combination with a decision tree to rank source peers. A learning process is necessary to generate a good decision tree, so new download peers will not benefit from this solution until enough learning data is available. Even when enough learning data is available, it may turn out that none of the available fields offer a good prediction of the source peer bandwidth.

Zeitoun et al. [49] propose to use the network round-trip time to decide which source peers to download from. In their solution, the downloading peer measures the RTT of each source peer and selects only the source peers that reply the fastest. In large-scale networks, however, RTT measurements may not be a reliable estimation of bandwidth availability and may vary widely over the course of a few seconds.

A similar approach is proposed by Li [34], but instead of RTT measurements the downloading peer downloads a small fixed-size chunk from each source peer in order to measure the actual bandwidth.

Chui et al. [18, 33] have shown that the actual bandwidth received from source peers on the internet can vary widely even over short time spans. Even if a source peer provides high bandwidth at one point, no guarantee is provided that the remainder of the download will continue at that same speed. It is therefore likely that the bandwidth estimates provided by previous solutions could still result in suboptimal downloading performance. As a solution, the authors propose to use *Random Periodic Switching*. In this scheme, the downloading peer will make a uniformly random selection from the source peers and keep downloading from those source peers for a short time period. After that period, a new set of source peers is randomly selected again and a new period starts. Random Periodic Switching guarantees that the downloading process will not get stuck indefinitely on a slow source peer.

Random Periodic Switching, however, does not necessarily minimize the download time. Biased Random Periodic Switching (BRPS) is an improvement to the algorithm proposed by the same author [17, 19]. In this scheme, each source peer is assigned a different connection probability based on the bandwidth received in earlier periods and the number of competing downloading peers in the network.

2.1.4 The BRPS Algorithm

The BRPS algorithm, as proposed in [17] and [19], works as follows. The downloading process is divided into time periods of a fixed length, say, one minute. At the start of each period, the downloading peer selects, in a random manner, a number of source peers and then keeps downloading from those source peers for the entire duration of the period. When the period ends and there is still file data remaining to be downloaded, a new random selection is made and the next period starts.

The downloading peer maintains a list of all source peers, S, that have file data of interest. A connection probability p_j is assigned to each source peer $j \in S$, which indicates the probability that the downloading peer will connect to that source peer in the following period. These connection probabilities are normalized to L, the configured DoP (Equation 2.2). This way, the actual number of source peers that is selected in each period is random, with an average of L. L is bounded by the number of available source peers, i.e. $L \leq |S|$.

The distribution of p_j is determined by the optimization problem in Equation 2.1, where $|D|$ is the number of competing download peers in the network and c_j the total upload bandwidth provided by source peer j. The equation describes the optimal values of p_j for which the sum is the maximum. The sum has the form of $\sum_{j \in S} \alpha_j c_j$, where $0 \leq \alpha_j \leq 1$ is the variable to be determined. The maximum of the sum is achieved when all α_j for the highest corresponding c_j are assigned a higher value. If α_j were *equivalent* to p_j, then *only* the source peers with the highest c_j would be assigned a nonzero probability. But we want slower source peers to have a connection probability, too, especially if there is much competition (i.e. when $|D|$ is higher). Defining $\alpha_j = 1-(1-p_j)^{|D|}$ ensures this. When $|D|$ is small, p_j is close to α_j and only source peers with a high c_j are assigned a nonzero p_j. As $|D|$ increases, the influence of p_j decreases and source peers with a lower c_j are assigned a nonzero p_j, too.

$$\underset{p_j}{\operatorname{argmax}} \sum_{j \in S} [1 - (1 - p_j)^{|D|}] c_j \tag{2.1}$$

$$\text{s.t.} \sum_{j \in S} p_j = L, 0 \leq p_j \leq 1 \tag{2.2}$$

When viewed from a higher level perspective, the goal of this equation is to distribute the connection probabilities in such a way that source peers with more bandwidth (a higher c_j) are assigned a higher p_j than source peers with less bandwidth. However, if there is much competition in the network from other downloading peers, i.e. with larger $|D|$, it is likely that faster source peers are already busy with serving other downloading peers. In that case, it may be beneficial to give source peers with lower bandwidth a better chance to be selected, in order to better distribute the bandwidth among the available source peers.

This behaviour is visualised in Figure 2.2. For smaller values of $|D|$, the connection probabilities are chosen to strongly favour source peers with higher bandwidth. But as $|D|$ increases, p_j converges to a uniform distribution in order to give slower source peers a better chance to get selected.

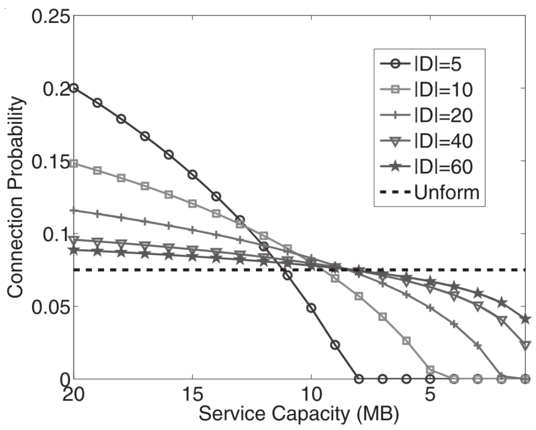

Figure 2.2: Distribution of the connection probabilities (p_j) on the Y axis among different source peer bandwidths (c_j) on the x axis. Plotted for different values of $|D|$. Copied from [17].

Note that, in the above equations, it is still possible for a source peer to be assigned a zero probability of being selected. In practice it is desirable to still give every source peer a small chance to be selected so that it may be re-evaluated again in the future. To achieve this, all values of p_j are multiplied by a factor $(L - 0.5)/L$, and $0.5/|S|$ is then added to each p_j. This essentially reserves half of a DoP to uniform selection, while ensuring that the invariants of Equation 2.2 still hold.

The exact values of $|D|$ and c_j are not known to the downloading peer, but these values can be estimated. When connected with a source peer, the downloading peer needs to know how many other peers the source peer is uploading to (X_j). Additionally, the downloading peer measures the received bandwidth

(R_j) from each connected source peer throughout the downloading period. In order to obtain the initial estimates, the downloading peer will connect to at most L random source peers which it has not downloaded from yet in the first number of periods. The value of X_j can be used to estimate $|D|$, and X_j and R_j together can then be used to estimate c_j. The full algorithm is listed in Algorithm 1.

Algorithm 1 The BRPS algorithm as implemented in a downloading peer. Based on [17].

```
1: $S_j \leftarrow 0$            ▷ Number of times we have downloaded from source peer $j$
2: $c_j \leftarrow 0$                  ▷ Average estimated bandwidth of source peer $j$
3: $\chi_j \leftarrow 0$                      ▷ Average number of peers connected to $j$
4: while there is something to download do
5:     if $\exists S_j = 0$ then
6:         Select, uniformly random, at most $L$ source peers for which $S_j = 0$
7:     else
8:         $|\hat{D}| \leftarrow \frac{1}{L} \sum \chi_j$
9:         Calculate $p_j$ according to Equation 2.1 and 2.2, using $|\hat{D}|$ as $|D|$.
10:        Select source peer $j$ with probability $p_j$
11:    end if
12:    Download from the selected source peers for the duration of one period
13:    for all selected source peers do
14:        $S_j \leftarrow S_j + 1$
15:        $X_j \leftarrow$ number of peers connected to source peer $j$
16:        $R_j \leftarrow$ observed download speed from source peer $j$
17:        $c_j \leftarrow \frac{(S_j-1)c_j + R_j * X_j}{S_j}$
18:        $\chi_j \leftarrow \frac{(S_j-1)\chi_j + X_j}{S_j}$
19:    end for
20: end while
```

2.2 Direct Connect

A quick overview of the architecture of the Direct Connect network has been given in Section 1.1. A few protocol and implementation details that will be useful in the remainder of this report are explained in more detail in this section.

2.2.1 High-level overview

The overall network topology of a Direct Connect network has been displayed in Figure 1.1 in the introduction. Because clients are all interconnected with each other through hubs, the hub is responsible for coordinating a number of network-level tasks.

User authentication: Users log in to the hub with a user name — often called a *nick*. If the user is registered to the hub, it needs to provide a password as well. The hub is responsible for ensuring that the nicks are unique within the context of the hub and that the same user does not connect to the hub multiple times.

User list synchronisation: The list of all users connected to the hub is synchronised to all connected clients. The following information is commonly distributed: Nick, IP address, e-mail address, description, upload speed, number of slots (described in Section 2.2.2), number of other hubs the client is connected to, and the total size of the users' shared files. Many of these fields are provided by the user and are therefore not always reliable. Some hubs remove fields or do not keep all information updated in order to save bandwidth.

Routing chat messaging: Chat messages can either be broadcasted to all connected clients — which is often considered the "main chat" of a hub — or they can be sent to a specific user to allow for private messaging[1].

Routing search queries: Search queries for specific files are routed through the hub.

Facilitating connections: The hub relays messages between two clients in order to initiate a direct connection between them. This is discussed in more detail in Section 2.2.3.

Hubs differentiate themselves in many ways. Many require that a user shares a certain minimum amount of data before they are accepted. Some are specifically targeted at users in a geographical location or with specific interests, e.g. certain types of files or a specific musical genre. Many hub operators continuously monitor the behaviour and shared files of their users, either manually or automatically, in order to discourage free-riding [36, 14] and to ensure that everyone only shares allowed files according to rules imposed by the hub. Not all hubs are public: Direct Connect is also commonly used in local area networks, where users communicate and exchange files using a hub running on the local network itself.

Direct Connect has been around since 1999 and the protocols have been revised many times. There have been many different client implementations for DC, but only a handful have been updated to keep up with the backwards-incompatible protocol changes that have been made over the years. As such, only a few clients are still in active use today, these include DC++ [3] and its various derivatives such StrongDC++ [8] and EiskaltDC++ [4]. Jucy [5] and ncdc [7] are two modern clients that are not based on DC++.

2.2.2 Source Peer Behaviour

Each source peer has a set of local files which it makes available to (*shares with*, in Direct Connect terminology) the other peers in the network. Only when a direct TCP connection between a downloading peer and a source peer exists (see Section 2.2.3), can the downloading peer download files from the source peer. This downloading happens with a 'GET' command, which takes three arguments: A file identifier, the byte offset to start downloading from, and the number of bytes to download. This allows the downloading peer to request any byte range in any file shared by the source peer.

[1]Private messages are, however, not completely private. They are still routed through the hub in plain text.

The ADC protocol specifies that the connection between two peers is in a special state while it is used for a file upload. In this state, the source peer does not accept a new 'GET' request until the current file transfer has been completed. In DC++ (0.802) and ncdc (1.19), the client still reads new incoming messages from the connection, but ignores (DC++) or disconnects (ncdc) the other peer upon receiving a new request while a file transfer is active. As a result, downloading peers can not request multiple file parts simultaneously, and have to wait for a previous chunk to complete downloading before requesting a new chunk. The pipelining technique proposed by Rodriguez et al. [41] will not work in current implementations of Direct Connect.

The number of peers that can simultaneously download from a source peer is restricted by the uploading peer. In Direct Connect terminology, this is called the *slot count*, where a *slot* represents one upload transfer. This number can be configured by the user, and typically varies between 1 and 50. If a downloading peer sends a 'GET' request when all slots in the source peer are in use, the source peer will reply with an error.

2.2.3 Connection initiation

As mentioned in the previous section, a direct TCP connection between the source peer and downloading peer has to be established in order to transfer file data. A distinction is made between two client modes: *active* and *passive*. A peer capable of accepting incoming TCP connections from the outside is considered active, whereas a peer behind a firewall or Network Address Translator (NAT) is passive.

Depending on whether the source and/or downloading peer are active or passive, there are three possible scenarios when attempting to establish a connection between two peers. The hub plays a central role in coordinating connection establishment in all three cases.

1. **Downloading peer is active.** (Figure 2.3a) In this scenario, the downloading peer can accept incoming TCP connections from the outside world. In order to establish a connection with a source peer, the downloading peer can, through the hub, send a 'Connect To Me' message to the source peer (steps 1 and 2). This message includes the IP address and TCP port of the downloading peer, which the source peer can use to establish a connection (step 3).

2. **Source peer is active.** (Figure 2.3b) If the downloading peer is passive but the source peer is active, the downloading peer can send a 'Reverse Connect To Me' message through the hub to the source peer (steps 1 and 2). Upon receiving this message, the source peer will reply with a 'Connect To Me' message (steps 3 and 4) as in scenario #1, and the downloading peer opens a direct TCP connection to the source peer (step 5).

3. **Source peer and downloading peer are passive.** The above two scenarios do not work if both peers are passive. A NAT traversal technique based on TCP Hole Punching [25] has been implemented as the NATT [46] extension for the ADC protocol. This extension can allow for two passive peers to still create a direct connection with each other, but does not work for all NATs and does not solve the problem of restrictive firewalls.

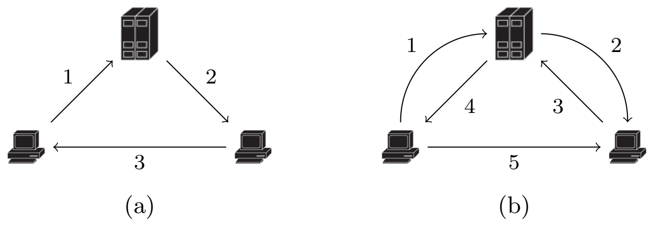

Figure 2.3: Connection initiation between two peers. The initiating peer is active in (a), passive in (b).

In most Direct Connect client implementations, configuring active mode is the responsibility of the user. It may happen that the user has configured active mode, but has not updated their firewall or NAT to actually allow the incoming connection, causing scenario #1 and #2 to fail. In addition to the fact that NAT traversal does not always work, this means that it is possible that two peers are unable to establish a direct connection with each other.

2.2.4 Peer Selection in Ncdc

Since this research will be comparing BRPS against an unmodified version of ncdc, an explanation of how peer selection currently works in ncdc (1.19) is beneficial.

All source peers known to ncdc are in one of the following states.

Idle Not connected. All source peers are in this state at the start of the downloading process.

Connecting Trying to open a connection. If opening a connection fails within a certain time, the source peer automatically moves into the Idle state.

Connected A direct connection exists with this source peer, but this connection is not yet used for downloading. If a source peer remains in the Connected state for a certain time, it is automatically disconnected and moved to the Idle state.

Downloading A direct connection exists with this source peer and is being used to download file data.

The *actual DoP* refers to the number of source peers in the Downloading state. The peer selection in ncdc is straightforward, and its primary goal is to ensure that the actual DoP equals the configured DoP. Every 500 ms, ncdc looks at the state of all source peers and performs the following two operations.

1. If the actual DoP is lower than the desired DoP and there are one or more source peers in the Connected state, start downloading from these source peers. If there are more source peers in the Connected state than necessary to achieve the desired DoP, the source peers are chosen by pseudo-random selection.

2. If the actual DoP is lower than the desired DoP and there are no more source peers in the Connected state, try to open a connection with all source peers that are in the Idle state.

The pseudo-random selection in the first operation is a consequence of the use of a hash table as the container for all source peers, and the peer selection algorithm walks through the source peers in hash table order. A unique client identifier is used as part of the key in the hash table, which ensures that different downloading peers will have a different hash table order. However, the order of the list of source peers may be exactly the same at different points in time in a single downloading peer, so the same 'random' selection may be chosen multiple times.

It is also worth noting that, once the actual DoP equals the desired DoP, no changes are made to the peer selection. The effect of this is that, as soon as an ncdc downloading peer has started downloading from a selection of source peers, it will keep using this selection for the entire remainder of the downloading process.

Chapter 3

Challenges in Integrating BRPS in Direct Connect

This chapter provides an overview of the challenges in integrating the BRPS algorithm in a Direct Connect client. Solutions to these challenges will be presented in Chapter 4. The challenges are described in sections 3.1 through 3.4, and summarized in Section 3.5.

3.1 Number of Downloading Peers

The BRPS algorithm, as described in Section 2.1.4, requires information about how many other downloading peers a particular source peer is uploading to. This information is needed to estimate the competition ($|D|$) in the network, which is used to obtain good a connection probability distribion. When integrating BRPS into a Direct Connect client, the client must thus be modified to obtain this information while it is downloading from each source peer.

3.2 Chunk Allocation

The BRPS algorithm determines *when* to download from *which* peers, but it does not say anything about *what* should be downloaded. A separate chunk allocation algorithm is necessary to *allocate* byte ranges (chunks) within the requested files and assign those chunks to connected source peers for downloading. After this assignment, the downloading peer can use the 'GET' request described in Section 2.2.2 to download the file data.

A chunk allocation algorithm has two primary goals that are in conflict with each other. As soon as one chunk has been downloaded, a new 'GET' request has to be sent to the source peer before the download of the next chunk is started. Such a request causes the download progress to be interrupted for the duration of a single network round-trip, and can cause the overall downloading performance to degrade significantly when performed too often [41]. As such, a chunk allocation algorithm should request large chunks in order to minimize the time that is lost in waiting for the 'GET' requests.

On the other hand, a chunk allocation algorithm should also ensure that the same file data is only downloaded at most once, and as long as there is still file data to be downloaded from a selected source peer, the download can continue to make progress.

As an example, suppose that the downloading peer has one file in its download queue, and there are two peers from which it can download this file. In order to minimize the number of 'GET' requests, it could request the entire file as a single chunk from one source peer. With that strategy, the second source peer can not make any downloading progress because everything has already been requested from the first source peer. An alternative strategy is to split the file in two equally-sized chunks and download the chunks from both source peers. But it is likely that one source peer is faster than the other, and the download of one chunk completes much faster than the other chunk. In that case, because the other half of the file has already been allocated to the slower source peer, there is no more unallocated file data for the faster source peer and the download is again stuck on a single source peer.

3.3 Source Peer Unavailability

The BRPS algorithm as described in Section 2.1.4 has an implicit assumption that all known and online source peers can be connected to and downloaded from at all times. In particular, if it is possible that one of the known source peers cannot be downloaded from at all, then the branch on line 5 of Algorithm 1 in Section 2.1.4 will be taken at every iteration, and no downloading progress will be made at all.

Unfortunately, the assumption that all source peers can be downloaded from does not hold in a real-world deployed Direct Connect network. Two major factors that break this assumption are described below.

1. All upload slots in the source peer can be in use, as described in Section 2.2.2. For very popular source peers, or for source peers with an unfair slot allocation scheme, this waiting time can be in the order of a few days or even weeks.

2. As described in Section 2.2.3, it is possible that no direct connection can be established at all with a certain source peer, making downloading from that source peer impossible.

Neither of the above problems can be detected up front in a reliable fashion. The only reliable way to know whether it is possible to download from a particular source peer is to open a connection and start downloading a file. Doing this for all possible source peers in the network is not feasible, so another solution needs to be implemented.

3.4 New Source Peers

When one or more new source peers are added to the list of known source peers while a download is in progress, the BRPS algorithm will select a number (limited by the DoP) of these source peers for downloading in the next period, as per line 5 and 6 of Algorithm 1 in Section 2.1.4. If, in that next period, the number of new source peers is lower than the DoP, then these new source peers are the *only* source peers being selected in that period.

More generally, as long as there exists at least one source peer that has not been downloaded from (i.e. $\exists S_j = 0$), then the random selection stage of BRPS will not occur, and the maximum DoP in that period is limited by the number of source peers for which $S_j = 0$. This may cause the overall average DoP to drop below its desired value.

This problem will not prevent download progress from being made at any time, but it may negatively affect the downloading performance. A Direct Connect client that implements BRPS could thus be improved by implementing a solution that ensures that the average DoP will approximate the desired DoP.

3.5 Conclusions

There are four challenges that need to be addressed when implementing BRPS in a Direct Connect client.

1. The downloading peer needs to be able to obtain information about the number of other downloading peers that the source peer is serving.
2. A chunk allocation algorithm is necessary to decide which file parts to request from source peers.
3. It is possible that some source peers are not available when they have been selected for downloading, and BRPS needs to be modified to deal with this in order guarantee that downloading progress is always being made.
4. The BRPS algorithm may not reach the desired DoP when new source peers are discovered during the downloading process.

These challenges are addressed in the next chapter.

Chapter 4

Solutions to Integrating BRPS in Direct Connect

This chapter looks at the challenges described in Chapter 3 and proposes a solution for each challenge. For some challenges it is possible to use multiple solutions. In these cases, the solutions proposed in this chapter are categorized as either *simple* or *improved*. The solutions marked as *simple* offer a minimal solution in order to ensure that BRPS can work within the context of Direct Connect. The solutions marked as *improved* go beyond that to avoid certain performance pitfalls when using the simple solution.

This chapter follows a similar organization to Chapter 3. Each challenge is handled in its own section in the same order as Chapter 3, followed by a summary in Section 4.5.

4.1 Number of Downloading Peers

The number of downloading peers that each source peer is uploading to can be obtained by observing the slot counts of each source peer. The number of *active slots* is equivalent to the number of downloading peers that the source is uploading to, but the ADC protocol does not provide a mechanism to obtain this number directly. Instead, two other numbers are obtained. One of them is the total slot count, which is distributed to all connected clients through the hub, and usually does not change over the course of a single download session. The other number that can be obtained through the ADC protocol is the number of *free slots*. That is, the number of upload slots that are not being used. This number can be obtained in the following three ways.

Direct search The ADC protocol offers each peer the ability to route a search query to peers connected to the hub. A response to such a query includes the number of free slots of the responding peer. The downloading peer can thus perform a search for a particular file that the source peer has, and in addition to that, extract the free slot count from the search results.

Such a search query can be targeted to all peers connected to the hub (flooding search), or can be specifically targeted towards a single peer (direct search). A flooding search is a relatively expensive operation for the hub, and hub implementations therefore impose strict limits on the number of flooding searches that each peer may perform within a certain time frame.

A direct search requires much less resources on part of the hub, but the currently available hub implementations do not distinguish between a flooding search and a direct search when limiting the number of search queries that a peer may perform. This restriction makes it impractical to use a direct search to query the free slot count for each selected source peer.

Distributed The free slot count of every source peer may be broadcasted to all connected clients using the *Free Slots* (FS) protocol extension [46]. Using this extension, the downloading peer can obtain the free slot counts in the same way as the total slot count — without querying the network. Unlike the total slot count, however, the number of free slots is more dynamic and may change over the course of a few minutes. Broadcasting every change of this number to all connected clients requires significant bandwidth on part of the hub, and is therefore not always done. Furthermore, not all client implementations support this extension, so the information may not be available for all source peers even if the hub does distribute it.

Get File Information When a direct connection has been established with a source peer, the downloading peer can communicate with the source peer without involvement of the hub. The downloading peer can then send a "Get File Information" (GFI) request. The response to a GFI request is equivalent to that of a regular search, and thus includes the number of free slots. Because the hub is not involved in this communication, there are no strict limits to how often this information can be requested.

The number of active slots can be calculated by taking the total slot count and subtracting the number of free slots. Since, following Algorithm 1 in Section 2.1.4, this number is only required of source peers that are being downloaded from, a connection with the source peer is already open and using the GFI mechanism is then the most suitable.

4.2 Chunk Allocation

The problem of chunk allocation as described in Section 3.2 is not specific to a download peer that implements BRPS. Every application that can download a single file from multiple source peers will have to implement a chunk allocation algorithm, but which strategy works best is dependent on properties of the files to be downloaded, the network, and the peer selection mechanism being used. As such, there is no general chunk allocation strategy that is optimal for all downloading applications.

The existing chunk allocation algorithms that are used in ncdc 1.19 [7] and DC++ 0.831 [3] are described below, and a suitable algorithm is selected for use with BRPS.

4.2.1 DC++

The chunk allocation algorithm used in DC++ 0.831 works as follows. Each file to be downloaded is divided into a number of equally-sized chunks. The size of these chunks is chosen such that it is a power of two larger than or equal to 64 KB, the minimum chunk size, and large enough to ensure that a single file consists of at most 1024 chunks. For example, a 8 MB file is divided into 128 chunks of 64 KB each, and a 150 MB file is divided into 600 chunks of 256 KB each.

When downloading from a source peer, one or more contiguous chunks are downloaded in a single 'GET' request. The number of chunks to download in the next request (n_{next}) is dependent on the number of chunks used in previous request (n) and the time it took to download those chunks (t). The formula used to calculate the next number of chunks is listed in Equation 4.1, where r is the ratio between t and a hard-coded setpoint of two minutes. If nothing has been downloaded from the source peer before, the initial number of chunks is chosen to result in a download request of 1 MB.

$$n_{next} = \begin{cases} n * 2 & \text{if } 0 \leq r < \frac{1}{4} \\ n + 1 & \text{if } \frac{1}{4} \leq r < \frac{4}{5} \\ n & \text{if } \frac{4}{5} \leq r < \frac{5}{4} \\ \min(1, n-1) & \text{if } \frac{5}{4} \leq r < 4 \\ \min(1, n/2) & \text{if } 4 \leq r \end{cases} \tag{4.1}$$

Chunks are allocated within the files using a first-find algorithm. The first found chunk in the file that has not been downloaded yet and that is not being downloaded is allocated to the source peer. If there are less than n_{next} free contiguous chunks after the first free chunk, then the number of chunks that is requested from the source peers is limited by the number of free contiguous chunks.

4.2.2 Ncdc

Ncdc 1.19 implements the following chunk allocation algorithm. Each part of a file is in one of the following states: *Unallocated* (initial state), *Allocated* or *Downloaded*. In the Unallocated state, the file data has not been downloaded yet and has not yet been requested for download. In the Allocated state, the file data has been requested and is being downloaded. When the file data has been received it is considered to be in the Downloaded state.

Ncdc uses the concept of *threads* to group byte ranges that have not been downloaded yet. File data to be downloaded is allocated within threads with the following algorithm:

1. Initially, a single thread is created to cover the entire byte range of the file (Figure 4.1a).
2. When a source peer is selected for downloading and there exists a fully-Unallocated thread, the first chunk of the largest such thread is requested from the source peer and is moved to the Allocated state (Figure 4.1b).
3. When another source peer is selected for downloading and there are no fully-Unallocated threads available anymore, the thread with the largest Unallocated part is chosen, and its Unallocated part is split in two halves.

A new thread is created from the second half (Figure 4.1c), and the first chunk of this thread is requested for downloading (Figure 4.1d).

4. When Allocated data has finished downloading, the starting offset of the thread is increased to exclude the downloaded data (Figure 4.1e). The thread then becomes available again for allocation in step 2.

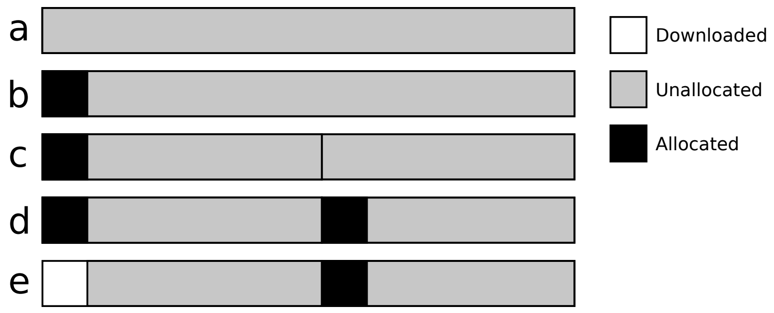

Figure 4.1: Example chunk allocation with the algorithm used in ncdc 1.19.

It is possible that an active download is interrupted before the Allocated chunk has finished downloading. In such a case, the data that has been received before the interrupt is removed from the thread following step 4, and the remaining chunk is available again as Unallocated file data.

The size of the chunk that is requested for downloading in step 2 and 3 is dynamic, and is dependent on the observed transfer speed received from the source peer. The chunk size is chosen in such a way to approximate a download time of 5 minutes for each requested chunk. When the download speed of the source peer is not known, a chunk of 128 KB is chosen in order to get an initial estimate.

4.2.3 Evaluation

Both the DC++ and ncdc chunk allocation algorithms make a trade-off between minimizing the overhead of issuing multiple 'GET' requests and ensuring that a number of source peers can download the same file data simultaneously.

Both algorithms take the speed of the source peers into account. DC++ does this by updating the size of the request by at most a factor two between consecutive requests, whereas ncdc does not limit the rate of change between requests. The thread-based allocation algorithm used in ncdc guarantees that, if a large enough free contiguous chunk is available, that chunk is allocated to the source peer. The first-find allocation scheme used by DC++ does not guarantee this, and may result in smaller request sizes than preferred.

Since neither chunk allocation algorithm relies on any specific peer selection mechanism, both are suitable for use with BRPS. While it may be possible to design an improved algorithm specific to BRPS, the primary goal of this research is to compare different *peer selection* mechanisms with each other. Because ncdc is used to evaluate BRPS in this research, it makes sense to re-use its chunk allocation algorithm.

4.3 Source Peer Unavailability

The problem of unavailable source peers described in Section 3.3 has two implications on downloading performance. If the problem is not handled at all, then the downloading progress may get stuck indefinitely if one of the source peers is unavailable. But even if a solution is implemented to ensure that downloading progress can be made, unavailable source peers can still degrade downloading performance by lowering the average DoP.

A simple solution to ensure that progress is being made is proposed in Section 4.3.1. An improved solution to ensure that the downloading performance does not degrade is proposed in Section 4.3.2.

4.3.1 Simple Solution

Because it is not always known in advance which source peers can be downloaded from and which source peers are unavailable, an implementation has to accept that some source peers selected in a period may be unavailable for downloading.

A simple solution to minimize the number of times that unavailable source peers will be selected is by adding a *backoff timer* [11]. When a source peer has been selected but could not be downloaded from, it will be temporarily removed from the list of known source peers. After a certain timeout, the source peer is re-added so that a new connection can be attempted again in a next downloading period.

If a constant timeout value is chosen, then it is possible that still no progress can be made when the number of unavailable source peers grows beyond a certain number. Exponential backoff can be used to avoid this problem. With exponential backoff, the time that a source peer is removed from the list is doubled after each failed download attempt. For example, when a selected source peer is unavailable after the first attempt, it will be removed from the list of known source peers for the duration of one period. When the source peer has been re-added and the second connection download attempt has failed as well, it will be removed from the list for the duration of two periods. After a third failed attempt, the timeout is increased to four periods, and so on, up to at most 64 periods (32 minutes). This ensures that source peers that are continuously unavailable will not hinder the selection of other source peers, thus allowing downloading progress to be made.

4.3.2 Improved Solution

The simple solution proposed in the previous section ensures that downloading progress can be made, but the presence of unavailable source peers can still degrade the downloading performance. This is possible because, when one or more unavailable source peers are selected in a period, the number of peers that are actually downloaded from in that period will be lower than the number of selected peers. The end result is that the overall average DoP will be lower than desired.

The solution to this problem is to ensure that the number of source peers that is actually downloaded from, equals the number of selected peers in each period. An approach to ensure this is by implementing gradual source peer takeover. When a new period starts, the downloading peer will keep downloading from the source peers that were selected in the previous period. Then, the downloading peer will attempt to request a download from each of the newly selected source peers. If such a request succeeds, the download from a source peer of the previous period is stopped. If, on the other hand, the request fails, then an existing downloading connection that has been selected in the previous period will keep running. In this way, a gradual source peer takeover procedure is implemented where the existing download connections from the previous period will be replaced gradually with the selection of the next period.

Since the number of source peers that is selected in each period is random, it will happen that there is a difference between the number of selected source peers and the number of active downloads from the previous period. If there are more active downloads than there are selected source peers, then a number of active downloads will be stopped immediately at the start of the period. Likewise, if there are less active downloads than desired, then no active downloads will be stopped until enough new download connections have been started.

4.4 New Source Peers

The problem of new source peers, as described in Section 3.4, does not prevent download progress from being made. As such, a simple BRPS implementation does not need to implement a solution in order to function. However, the problem of new source peers may degrade downloading performance and an improved BRPS implementation may benefit from implementing the following solution.

The problem of the decreasing average DoP can be trivially avoided by ensuring that the random peer selection is performed even if there are still source peers that have not been downloaded from. If the number of source peers that have not been downloaded from is lower than the desired DoP, then the random selection can be performed using the *remaining DoP* as the new value of L in the calculation of the connection probabilities p_j. This solution is illustrated in Algorithm 2, which serves as the replacement to line 5–11 of Algorithm 1 in Section 2.1.4.

Algorithm 2 Fall back on random selection as long as the remaining DoP is larger than 0.

```
 1: $N \leftarrow L$                    ▷ Remaining DoP, initialized to the desired DoP
 2: if $\exists S_j = 0$ then
 3:     Select, uniformly random, at most $L$ source peers for which $S_j = 0$
 4:     $N \leftarrow N -$ number of selected source peers
 5: end if
 6: if $N > 0$ then
 7:     $|\hat{D}| \leftarrow \frac{1}{L} \sum \chi_j$
 8:     Calculate $p_j$ according to Equation 2.1 and 2.2,
 9:       substituting $|D|$ for $|\hat{D}|$ and $L$ for $N$.
10:     Select source peer $j$ with probability $p_j$
11: end if
```

4.5 Conclusions

A number of solutions have been proposed to the challenges posed in Chapter 3. The solutions are organized in two categories, simple and improved. This also results in two implementations of BRPS in Direct Connect, a simple implementation that implements BRPS with little modifications to the core algorithm, and an improved implementation that attempts to avoid some of the performance pitfalls that may arise in a real network.

The simple solutions are as follows:

- The *number of competing downloading peers* in the network can be obtained with the “Get File Information” command when connected with a source peer.
- The *chunk allocation algorithm* that is already used in ncdc is suitable for use with BRPS, and can thus be re-used.
- *Unavailable source peers* that were selected in a certain period but could not be downloaded from within the same period are temporarily removed from the list of source peers using exponential back-off. This ensures that available source peers will get a better chance to be selected in the next period.

In addition to the simple solutions, the improved implementation will implement the following solutions:

- *Unavailable source peers:* To avoid that unavailable source peers will cause the average DoP to drop below its desired value, download connections from source peers selected in the previous period are gradually taken over after the connections with the newly selected source peers are successful.
- *New source peers:* If new source peers are selected in a period but there were less new source peers than the desired DoP, the random selection stage is performed with the remaining DoP.

Chapter 5

Experiment Setup and Results

This chapter attempts to answer question 3 of Section 1.3: How can a Direct Connect client that uses the BRPS algorithm be analyzed and evaluated? The experiments experiments used to evaluate BRPS in Direct Connect are explained below.

This chapter is organized as follows. Section 5.1 introduces the goals of the experiment, Section 5.2 describes the topology used for the experiments and the interesting performance metrics are described in Section 5.3. Sections 5.4, 5.5, 5.6, and 5.7 explain various experiments where one parameter of interest is being varied.

5.1 Experiment Goals

The goal of the experiment is to evaluate the performance of three peer selection implementations:

ncdc The existing algorithm used in ncdc 1.19 [7].

BRPS-simple A modified version of ncdc 1.19 with the simple BRPS algorithm, as described in Section 4.5.

BRPS-improved A modified version of ncdc 1.19 using the improved BRPS algorithm, as described in Section 4.5.

The experiments aim to answer the following sub-questions:

1. How do the different implementations perform in a realistic network situation?
2. What is the effect of different file sizes on the download time?
3. What is the effect of a different configured DoP on the downloading time and the number of chunk requests?
4. What is the effect of the number of source peers on the downloading time?
5. What is the effect of unavailable source peers on the actual DoP?

5.2 Topology

The experiments are performed on a controlled Direct Connect network that is setup to represent a realistic network scenario. All Direct Connect clients in this network are evenly distributed over 10 computers. Each of these computers runs 64-bit Ubuntu 12.04 on an Intel Xeon E3310 CPU with 8 GB of memory and a 4 TB hard disk. These computers are connected with each other in a star network with a Gigabit Ethernet switch.

All clients in the network connect to a Direct Connect hub running on a separate computer. This hub runs a default configuration of uhub 0.4.1 [10], an open source hub implementation.

5.2.1 Source Peers

A number of clients in the network will be configured to function as source peer. These clients will all share the same files. Since the peer selection algorithm does not affect the behaviour of the source peers, it does not matter which Direct Connect implementation is being used for these clients. In the experiments, all source peers will run an unmodified ncdc 1.19.

Based on measurements performed in [27], popular files in a large Direct Connect hub are often available from 20 to 200 source peers. As a common middle ground, 50 source peers will be used in most experiments. Other values are tried in the experiments described in Section 5.6.

Although the number of upload slots of each source peer is often limited in real networks, the source peers in the experiments will be configured to have enough upload slots to serve all downloading peers simultaneously. In this way, no source peer will refuse service during the experiments, giving all downloading peers the freedom to choose source peers as they wish. The reason for removing the upload limit restriction is to move the responsibility of a good bandwidth distribution on the peer selection mechanism implemented in the downloading peers.

Of course, source peers that refuse service do exist in real networks and it is important that a peer selection algorithm takes this into account. This is tested in a separate experiment described in Section 5.7.

Bandwidth distribution

Because it is highly uncommon for all source peers in a P2P network to be able to offer a full Gbit/s upload bandwidth, the maximum upload speed of each source peer will be throttled in the experiments. To improve the realism of the experiments, the bandwidth distribution of the source peers will be based on measurements performed on an existing P2P network. No existing measurements for Direct Connect could be found, so measurements from another P2P system will be used as an approximation.

Existing measurements on the Gnutella P2P network [40] have been published in [26], but this study was performed in 2002 and is unlikely to be representative today.

The bandwidth measurements of all active peers in the Tor P2P network [23] is available on the TorStatus website [9]. These measurements are updated in real-time to reflect the actual network status and are therefore representative of the bandwidth distribution in a modern P2P network. A snapshot of these measurements made on January the 13th of 2014 will be used to derive the bandwidth distribution of the source peers in the experiments. The average bandwidth has been normalized to 500 KB/s. The Cumulative Distribution Function (CDF) is displayed in Figure 5.1.

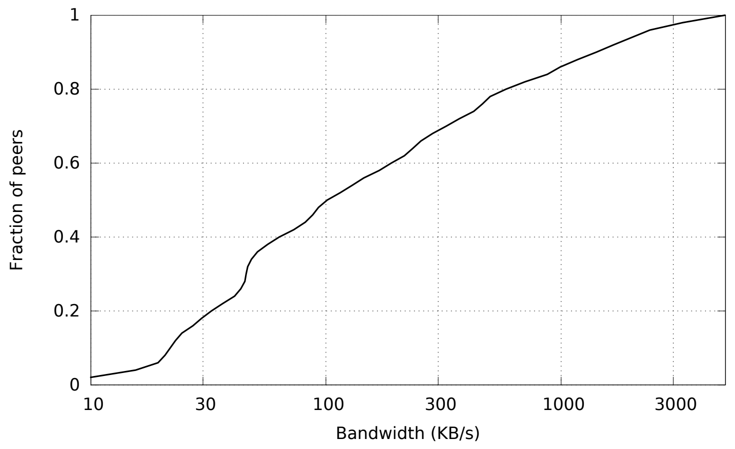

Figure 5.1: CDF of the normalized bandwidth distribution of source peers in the Tor network, retreived on January the 13th, 2014.

5.2.2 Downloading Peers

During the experiment, a number of clients will be configured to perform the role of the downloading peer. All downloading peers in an experiment will use the same client implementation, and the experiments are repeated for each implementation mentioned in Section 5.1.

In order to emulate competition between downloading peers, and to ensure that the experiments are performed on a network that is under actual load, each experiment will be performed with 50 downloading peers in the network. This results in a 1:1 ratio between downloading peers and source peers[1].

The downloading peers will download with a configured DoP of 4, which is the default value for ncdc 1.19. Other values are tested in the experiment described in Section 5.5.

The bandwidth that is available to the downloading peers is already limited by the source peers, and no further bandwidth limiting will be performed at the downloading peers.

In each experiment, all download peers will simultaneously download the same file from all of the available source peers. A 500 MB file will be used in most experiments. This is roughly as large as a (small) Linux installation image or an hour long compressed DVD video file. With the average source peer bandwidth of 500 KB/s, each experiment will take at least 17 minutes to complete according to Equation 5.1 described in Section 5.3.1. Other file sizes are tested in the experiment described in Section 5.4.

[1]But the number of source peers is varied in Section 5.6, so this ratio is not fixed.

For the clients that implement the BRPS algorithm, the length of each period needs to be chosen. In [17] a value of one minute is suggested, but no rationale is given. The first number of periods in BRPS are used to download from every source peer in sequence in order to obtain initial stats, per line 5–6 in Algorithm 1. Only after this initialization has finished are the probability calculations used for selecting source peers. The number of periods spent in this initialization phase is calculated by the ratio of the number of source peers and the DoP, i.e. $\lceil \frac{|S|}{L} \rceil$. With 50 source peers and a DoP of 4, this results in 13 periods used for initialization. If a period length of one minute is chosen, then the probability-based peer selection starts only after 13 minutes — more than halfway into the experiment. In order to take better advantage of the BRPS probability-based peer selection algorithm, a period length of 30 seconds is chosen instead. With this, the initialization phase only takes 6 minutes and 30 seconds.

5.3 Performance Metrics

Three performance metrics are of interest during the experiments. The download time, the average DoP and the number of chunk requests. Each of these is described in detail in the sections below.

5.3.1 Download time

The download time for a single downloading peer is defined as the time interval between the start and the end of the downloading process. When a downloading peer is instructed to start downloading, it records the system time (T_{start}) in a log file. As soon as the last byte of the file has been received, it records the system time (T_{end}) again. The download time for this downloading peer is $T_{end} - T_{start}$.

This download time is measured from all downloading peers simultaneously in each experiment. The average from these measurements is taken to represent the download time of a single experiment.

In addition to the averge download time for all downloading peers, a Cumulative Distrubion Function (CDF) is provided for some experiment results as well. This CDF displays the distribution of the download times among all downloading peers.

Optimal download time

The optimal download time for all downloading peers is, in these experiments, limited only by the bandwidth available from the source peers. Ignoring any signalling overhead and following the fluid model described in [31], the optimal average download time is then described by Equation 5.1. Here, F denotes the file size, $|D|$ the number of downloading peers and u the sum of all upload bandwidth provided by the source peers.

$$T_{opt} = \frac{F * |D|}{u} \tag{5.1}$$

This optimal average download time will be displayed in the resulting graphs for comparison against the theoretically optimal peer selection.

5.3.2 Average DoP

The desired DoP can be configured in the downloading client, but the actual number of source peers that a downloading peer is downloading from may vary over time. This variation can be the result of unavailable source peers (Section 3.3), because no more unallocated file chunks are available (Section 4.2), or by the time spent in signalling overhead.

As a performance metric, the average DoP for a single downloading peer is defined in Equation 5.2. Here t represents the time spent downloading the complete file, as described in Section 5.3.1, and t_j represents the time spent downloading from source peer $j \in S$. According to this equation, the maximum DoP that can be achieved is when downloading from all known source peers simultaneously for the entire duration t, in which case the DoP becomes $|S|$.

$$\text{DoP} = \frac{\sum_{j \in S} t_j}{t} \tag{5.2}$$

The average DoP is measured from all downloading peers simultaneously during each experiment, and the average from these measurements is taken to represent the average DoP of a single experiment.

5.3.3 Number of chunk requests

During the downloading process each downloading peer will send requests for file chunks to source peers. The number of these chunk requests is measured for each downloading peer in each experiment, and the average from all downloading peers is taken to represent the number of chunk requests of a single experiment.

As explained in Section 3.2, using few chunk requests to download a file is better than using many. However, a good peer selection will have to switch between several source peers to get better performance, and will thus necessarily increase the number of chunk requests.

Whether, and to what degree, a high number of chunk requests influences the download time is dependent on several factors. The most important factor being the network latency. If a network round-trip can take up to a second to complete, then each chunk request might cause the download time to increase by a second. In practice the time overhead of chunk requests can often be mitigated by implementing the pipelining technique (Section 2.2.2), and by not disconnecting old peers before a chunk request with a new peer has succeeded. The latter technique is already implemented in the improved BRPS method, as described in 4.3.2.

Since the experiments are performed on a low-latency network, having many chunk requests during the experiments is unlikely to have a measurable influence on the download time. To still be able to offer some insight into how the different peer selection mechanisms affect the number of chunk requests, this information is measured and displayed separately.

5.3.4 Confidence Intervals

For each of the above performance metrics, the 95% confidence interval is calculated according to [28, ch 13.2] and displayed along with the results. For the calculation of these confidence intervals, the Student's t-distribution is used.

In order to get accurate measurements, all experiments will be repeated multiple times in order to ensure that the confidence intervals stay within 5% of the measured average. To ensure that any randomness in the algorithms is different at each experimental run, the random seed is changed before each run.

5.4 File Size

In the first experiment the file size is varied among five different levels: 100 MB, 250 MB, 500 MB, 750 MB and 1024 MB. Everything else will remain as described in Section 5.2.

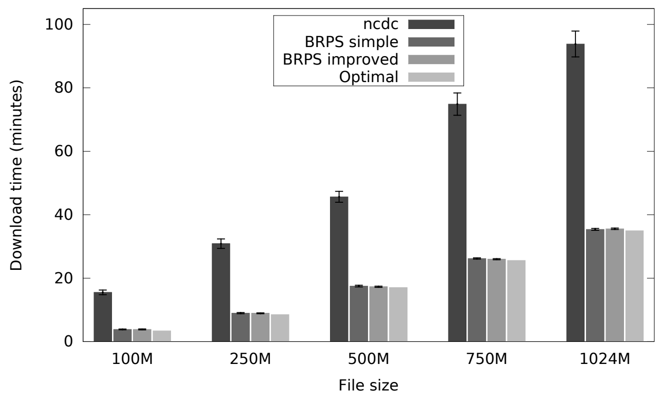

Figure 5.2: Comparison of the download time of the three implementations with different file sizes. The error bars display the 95% confidence intervals.

The measured download time for each of the implementations along with the optimal download time is displayed in Figure 5.2. Both BRPS implementations are very close to the optimal download time, whereas ncdc takes roughly three times as long to complete.

It is interesting to note that with a 100 MB file all BRPS peers finish downloading *before* the 6 minute initialization phase completes. This suggests that uniform random peer selection is actually a good peer selection strategy in this experiment.

The CDF for the experiment with 500 MB is displayed in Figure 5.3. What is interesting to see here is that not only are the BRPS implementations close to the optimal average download time, they also result in an almost even distribution of download times among the downloading peers. This is clearly not the case with ncdc. About 10% of the ncdc peers complete the download before the average optimal download time, but this is compensated for by the long tail of peers that take much longer to complete the download. The slowest ncdc peers take approximately 9.7 times longer than optimal, whereas the slowest BRPS peers are close to 1.1 times optimal.

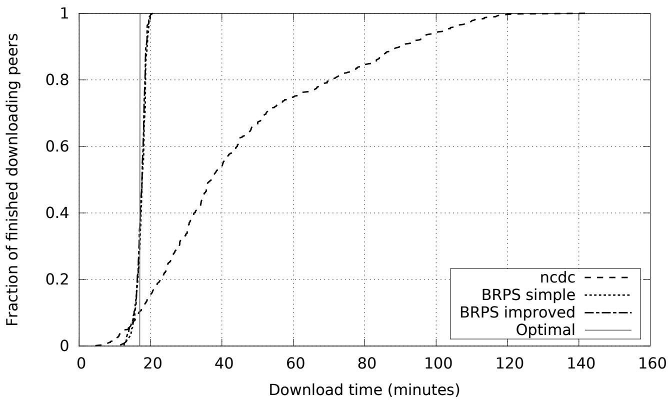

Figure 5.3: CDF of the of the download time of the three implementations while downloading a 500 MB file. The two BRPS implementations overlap.

5.5 Degree of Parallelism

In this experiment the DoP configured in each downloading peer is varied between the values 1, 4, 6, 10 and 15. A DoP of 4 is the default value in ncdc, and 6 is the default of DC++. The DoP is user-configurable and some users increase this value in the hopes of getting better performance, so a higher DoP of 10 and 15 is also included in this experiment. All other parameters remain as described in Section 5.2.

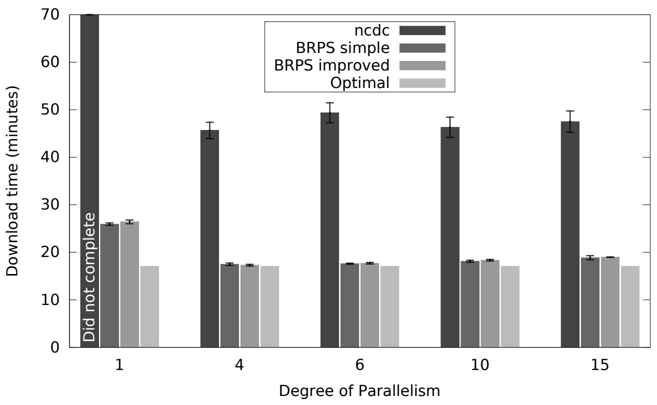

Figure 5.4: Comparison of the download time of the three implementations with a different DoP.

A comparison of the download times for this experiment is displayed in Figure 5.4. Ncdc did not complete the downloading progress within 12 hours at a DoP of 1 and has been marked as such. The reason for this suboptimal performance becomes clear when looking at the source peer bandwidth distribution shown in Section 5.2.1: The slowest source peer has an upload bandwidth of 10 KB/s. If even a single downloading peer chooses that source peer as its *only* source, then that peer will take at least 14 hours to download the 500 MB file.

If another downloading peer happens to choose that same source peer, then the download time will increase even further.

The BRPS implementations also do not perform optimally at a DoP of 1. At this DoP it takes 25 minutes[2] for BRPS to finish its initialization phase, taking up the vast majority of the download time. Clearly, uniform random selection is not an optimal peer selection mechanism at such a low DoP.

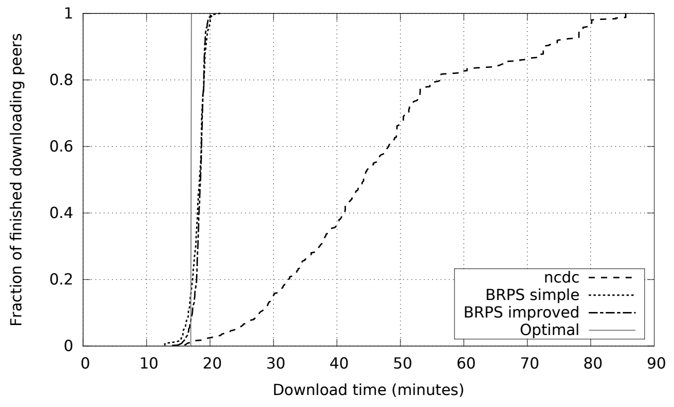

Figure 5.5: CDF of the of the download time of the three implementations at a DoP of 10.

While there is no significant change in the average download times at a higher DoP, the distribution of the download times does change for ncdc. As seen in the previous section, the slowest ncdc downloading peers would take up to 120 minutes at a DoP of 4. At a DoP of 10 (Figure 5.5) the slowest ncdc peers now finish within 90 minutes. This is expected, because the probability of selecting at least one fast(er) source peer increases as more source peers are used in the selection, thus making it less likely that a downloading peer gets stuck on a very slow source peer. This, however, has not improved the average download time, because the 'faster' peers have slowed down instead. With a DoP of 4, more 55% of the download peers finished before 40 minutes, with a DoP of 10 this is lowered to 40%. A higher DoP does not affect the download time distribution for the BRPS implementations.

The number of chunk requests required to download the file is graphed in Figure 5.6. As expected, the BRPS implementations perform significantly more chunk requests, approximately 10 times as many ncdc. Equally unsurprising, the number of chunk requests increases as the DoP increases. For BRPS this is because each new period requires sending out chunk requests to all selected source peers, and the DoP determines how many source peers there are.

The reason that the number of chunk requests also increases for ncdc can be explained with the chunk allocation algorithm described in 4.2. Chunk sizes are chosen to approximate a download time of 5 minutes, so a new chunk request is sent to each selected source peer *at least* every 5 minutes.

[2]At a 30 second period time and with one source peer per period. That's 2 source peers per minute, so 25 minutes for 50 source peers.

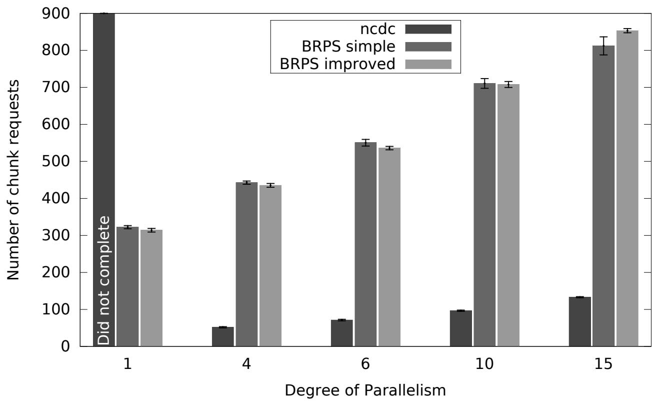

Figure 5.6: Comparison of the number of chunk requests of the three implementations with a different DoP.

5.6 Number of Source Peers

In this experiment the number of source peers is varied, while the number of download peers and the total available bandwidth stays the same. The number of source peers is varied between the following values: 10, 20, 50, 70 and 100.

In changing the number of source peers, it is unavoidable that something changes to the bandwidth distribution as well. Following Equation 5.1, the optimal download time is independent of the number of source peers, but does depend on the total available bandwidth. If the average upload speed of each source peer remains unchanged, then the total bandwidth available to the network, together with the optimal download time, will change as the number of source peers is varied.

The goal of this experiment is to allow comparison of the download performance at a different number of source peers while keeping all other parameters the same. In order to achieve this, the average bandwidth of each source peer is scaled to provide the same total bandwidth in each experiment. In this way the optimal download time is equivalent in each experiment, making for easy comparison. The average bandwidth per source peer and total network bandwidth for each experiment is listed in Table 5.1. The bandwidth distribution still follows the same shape as Figure 5.1 in every experiment.

#SP	u_i (KB/s)	u_t (KB/s)
10	2,500	25,000
20	1,250	25,000
50	500	25,000
70	357	25,000
100	250	25,000

Table 5.1: Mean upload speed for each source peer (u_i) and total bandwidth (u_t) at a different number of source peers.

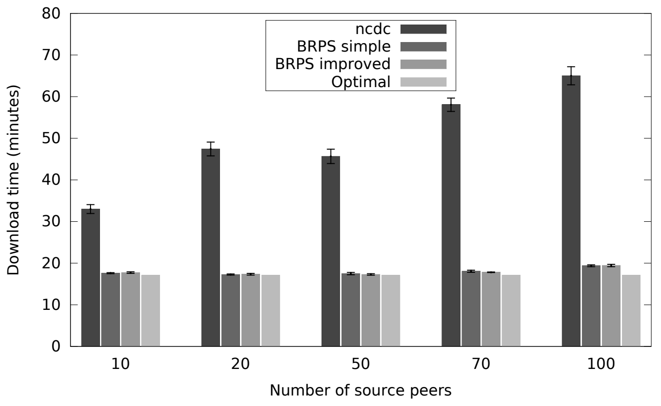

Figure 5.7: Comparison of the download time of the three implementations with a different number of source peers.

The resulting download times in this experiment are displayed in Figure 5.7. When the number of source peers to choose from is closer to the configured DoP, the peer selection mechanism being used has less influence on the download time. This is evident in the experiment with 10 source peers, where ncdc performs much better than in all previous experiments. The opposite is also true, a good peer selection is increasingly more important as the number of source peers to choose from increases. This explains why ncdc performance decreases at a higher number of source peers, while the two BRPS implementations remain roughly the same.

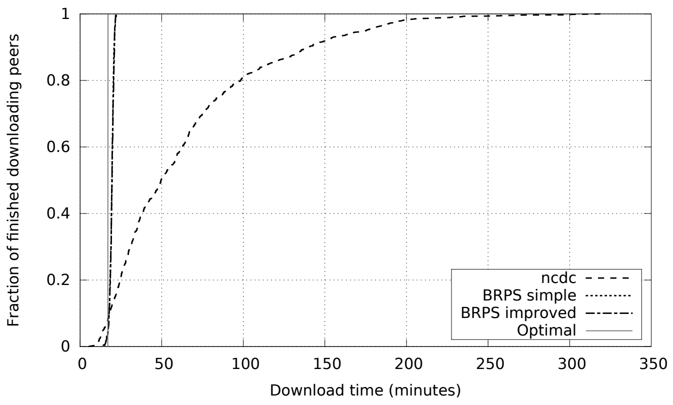

Figure 5.8: CDF of the of the download time of the three implementations with 100 source peers.

The distribution of the download times with 100 source peers is displayed in Figure 5.8. This distribution is roughly equivalent to those given in the previous sections. The main difference here is that ncdc is significantly slower.

5.7 Unavailable Source Peers

In this experiment the number of unavailable source peers is varied. An unavailable source peer appears as a normal source peer in the list of source peers that each downloading peer is given, but when a download peer tries to actually download from an unavailable source peer, the source peer will reply with an error and refuse to upload the file.

The optimal download time is only dependent on the bandwidth provided by the *available* source peers. The effect of changing the number of available source peers has already been studied in Section 5.6, and in order to focus solely on the effect of unavailable source peers, the number of available source peers will remain fixed in this experiment. The number of source peers used in each experiment is summarized in Table 5.2.

Available	Unavailable	Total
20	0 (0%)	20
20	5 (20%)	25
20	13 (40%)	33
20	30 (60%)	50
20	80 (80%)	100

Table 5.2: Number of available and unavailable source peers in each experiment.

The measured download times are displayed in Figure 5.9. What is surprising here is that the download times for ncdc do not remain constant. In fact, it appears that ncdc performs better as more unavailable source peers are added. In this experiment ncdc had similar behaviour as it had in other experiments, it did not switch source peers any more during the downloading process and there was no significant change in the DoP. The only difference at 80% unavailable source peers is that ncdc had selected faster source peers than in the other experiments. This seemingly random improvement might be caused by the fact that the peer selection in ncdc is determined by the order of a hash table implementation, as explained in Section 2.2.4. The number of source peers in this experiment is changed, and that likely causes the source peers in the hash table to be listed in a different order as well. Note that the order of the hash table is deterministic, so it does not change with different runs of the same experiment. It could have been the case that in this specific situation, the hash table order turned out to result in a better peer selection. More research would be needed in order to confirm this conjecture.

The number of unavailable source peers does not seem to have an effect on the performance of either BRPS implementation. Even the simple implementation does not degrade much at 80% unavailable source peers.

A difference in the two BRPS implementations can be observed when looking at the actual DoP, displayed in Figure 5.10. The average DoP remains constant for ncdc — as expected — and degrades only slightly for the improved BRPS implementation. The simple BRPS implementation, on the other hand, struggles to keep the DoP close to the optimal as the number of unavailable source peers increases. This difference between the two implementations can be attributed to the different handling of unavailable source peers, previously explained in Section 4.3.2. As seen in Section 5.5, BRPS performs well even at lower DoP values, which explains why the lower DoP of the simple implementation did not result in a significant increase of the download time.

The distribution of the download times at 80% unavailable source peers is displayed in Figure 5.11. This distribution is similar to the other experiments, the main difference being that ncdc performs better.

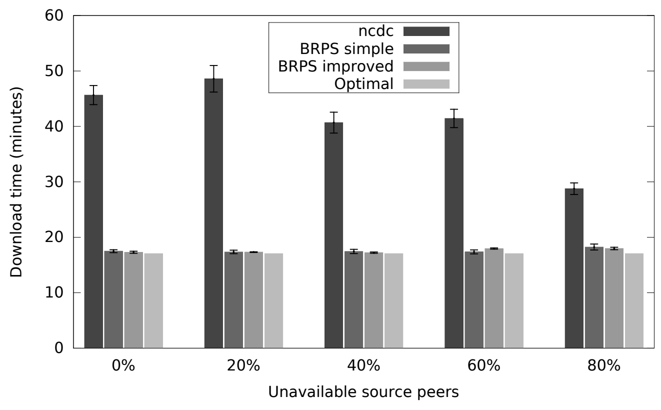

Figure 5.9: Comparison of the download time of the three implementations with a different number of unavailable source peers.

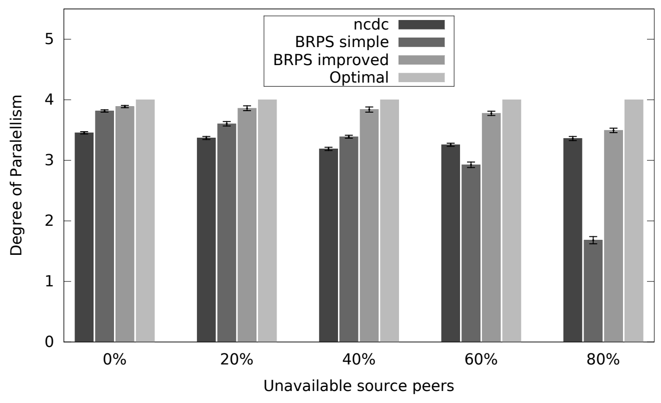

Figure 5.10: Comparison of the actual DoP of the three implementations with a different number of unavailable source peers.

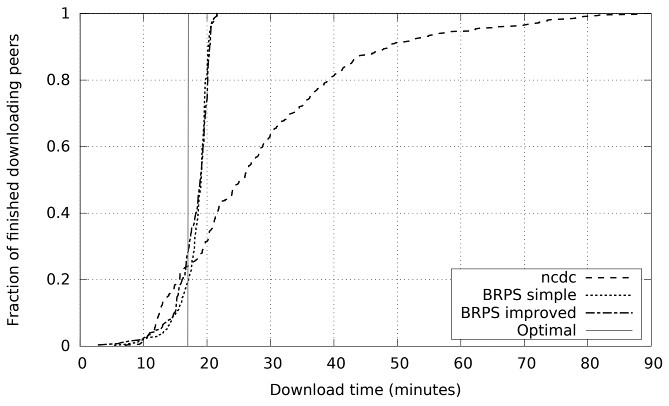

Figure 5.11: CDF of the of the download time of the three implementations with 80% unavailable source peers.

Chapter 6

Conclusions and Future work

This chapter finalizes this report with a short summary concluding the research, followed by recommendations for future research.

6.1 Conclusions

This research has focussed on integrating BRPS into an exiting Direct Connect client and to compare the downloading performance of an implementation with and without BRPS. The following research questions have been answered.

1. **How can the BRPS algorithm be integrated in Direct Connect?**

   No protocol changes were necessary to implement BRPS into Direct Connect, but a few challenges remained.

   The BRPS algorithm requires knowledge of the number of peers that are downloading from a particular source peer. This information is obtained by the slot counts that the source peer publishes.

   The BRPS algorithm does not dictate which file chunks are being requested from the source peers, so a separate chunk allocation algorithm is needed to do this. The existing chunk allocation scheme in ncdc has been re-used in the BRPS implementation.

   In an actual Direct Connect network, source peers may not always be available. In order to ensure that downloading progress can be made at all times, an exponential back-off timer has been added to temporarily exclude unavailable source peers from being selected.

2. **Which characteristics can improve the BRPS performance?**

   There are two scenarios where the average DoP can drop below the desired average DoP. An improved BRPS implementation has been designed to deal with these scenarios.

   When a source peer that has been selected for downloading turns out to be unavailable, the actual DoP will drop below what has been selected for that period. In order to avoid this, download connections from source peers selected in the previous period are gradually taken over after the connections with the newly selected source peers are successful.

When there exists a source peer that the downloading peer has not downloaded from yet, BRPS will always select that source peer. If the number of such new source peers is lower than the DoP, then the DoP in that period will be lower than desired. To avoid this problem, the remaining DoP is used for the regular random selection stage.

3. **How can a Direct Connect client that uses the BRPS algorithm be analyzed and evaluated?**

   The simple and improved BRPS algorithms have been implemented in ncdc 1.19, and the performance has been evaluated using set of experiments. The experiments have also been run with an unmodified version of ncdc for comparison.

   The experiments were performed on a controlled Direct Connect network that has been setup to represent a realistic network scenario. This setup consists of 50 downloading peers and a variable number of source peers. The source peers have been configured to throttle their upload speed to an average of 500 KB/s, and following a realistic bandwidth distribution based on measurements on the Tor P2P network.

   The experiments consisted of all downloading peers downloading the same file at the same time, and taking measurements on the side of these downloading peers. The measurements of interest include the download time, average DoP and number of chunk requests. The experiments have been repeated with different file sizes, configured DoP, number of source peers, and number of unavailable source peers.

4. **Does BRPS reduce the download time compared to the currently implemented peer selection mechanisms?**

   Four experiments have been performed in order to compare the performance of the two BRPS implementations with ncdc.

   In the first experiment the file size has been varied among five different levels: 100 MB, 250 MB, 500 MB, 750 MB and 1024 MB. As expected, the average download times increased linearly with the file size for all implementations. Both the simple and improved BRPS implementations performed close to the optimal average download time and were consistently faster than ncdc by a factor of 3.

   When looking at the distribution of the download times among the individual downloading peers, the BRPS peers all finished in a short time interval. The fastest BRPS peer finished around 20% before the optimal average download time, and the slowest peer around 15% after the optimal average download time. The distribution was very different for ncdc. The fastest ncdc peers finished much earlier than the optimal average download time (around 80% faster than optimal), but the large majority of the ncdc peers took much longer. The slowest ncdc peer took 8 times as long as the optimal average download time.

   In the second experiment the configured DoP was varied between the values 1, 4, 6, 10 and 15. At a DoP of 1, ncdc did not finish the downloading process within 12 hours. The two BRPS implementations did finish, but took around 50% longer to complete than the optimal average download time. At all other values of the DoP, the results were the same: BRPS performed close to the optimal average download time, and ncdc was again slower by a factor of 3.

In the third experiment the number of source peers was varied between the values 10, 20, 50, 70 and 100. The average bandwidth of each source peer was adjusted in order to keep the total network bandwidth equivalent in all experiments. The importance of source peer selection is lower when there are less source peers to choose from, and this was visible in the download times of ncdc. With 10 source peers, ncdc performed only 70% worse than the optimal download time, but with 100 source peers this increased to 300%. The two BRPS implementations again performed close to the optimal download time for all values.

In the final experiment a number of unavailable source peers has been added to the list of source peers. The number of available source peers has been fixed at 20, and the number of unavailable source peers has been varied between 0%, 20%, 40%, 60% and 80%. A surprising result has been that ncdc performed better as more unavailable source peers were added. At 80% unavailable source peers ncdc performed only 50% slower than optimal. The reason for this performance increase could be attributed to the pseudo-random selection based on the hash table order, but more investigation is needed to confirm this.

Both BRPS implementations again performed close to the optimal download time. However, a significant difference between the two implementations could be found when looking at the actual DoP. The simple BRPS implementation only had an average DoP of 1.3 with 80% unavailable source peers, whereas the improved BRPS implementation was at 3.7, much closer to the optimal of 4.

In all experiments, BRPS performed better than ncdc. The improved BRPS implementation performed better than the simple BRPS implementation in keeping the actual DoP close to the desired DoP, but this has not resulted in a noticeable difference in the download times.

6.2 Future work

What follows is a short list of recommendations for future research.

- Only a limited number of experiments have been performed in this research. It may be interesting to see how well BRPS compares against other Direct Connect implementations such as the DC++ client, or how well it performs with different bandwidth distributions and speed fluctuations in source peers.

- Some experiment results finished before the biased random selection stage of BRPS has started, suggesting that any periodic random selection, even uniformly random selection, is close to optimal. Future experiments could be performed in order to compare BRPS against uniform random selection to give more insight into the effect of the bias in BRPS.

- The exact reason why ncdc performs better when unavailable source peers are added is not known. A more thorough investigation is necessary to explain this behaviour.
- Future research could focus on finding a better chunk allocation algorithm than the existing implementation in ncdc.
- This research has assumed that all downloading peers already have a complete list of source peers they can download from, but in an actual Direct Connect network these peers have to be discovered first. Source peer discovery in Direct Connect is a topic where many improvements to the status quo could be achieved.

Bibliography

[1] Akamai. `http://www.akamai.com/`.

[2] Application-Layer Traffic Optimization (alto) Working Group. `https://datatracker.ietf.org/wg/alto/`.

[3] DC++ client. `http://dcplusplus.sourceforge.net/`.

[4] EiskaltDC++. `https://code.google.com/p/eiskaltdc/`.

[5] Jucy Direct Connect Client. `http://jucy.eu/`.

[6] Limelight Networks. `http://www.limelight.com/`.

[7] Ncurses Direct Connect. `http://dev.yorhel.nl/ncdc`.

[8] StrongDC++. `http://dcplusplus.sourceforge.net/`.

[9] Tor Network Status. `https://torstatus.info/`.

[10] uhub — High performance ADC Hub. `https://www.uhub.org/`.

[11] IEEE Std 802.3 - 2005 Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications - Section Five. Technical report, 2005.

[12] Micah Adler, Rakesh Kumar, Keith Ross, Dan Rubenstein, Torsten Suel, and David D. Yao. Optimal Peer Selection for P2P Downloading and Streaming. In *IEEE INFOCOM*, 2005.

[13] Vinay Aggarwal, Anja Feldmann, Christian Scheideler, and Michalis Faloutsos. Can ISPs and P2P users cooperate for improved performance. *ACM SIGCOMM Computer Communication Review*, 37(3):29–40, 2007.

[14] Andreas J. Alberts and Albert S.J. Helberg. Practical Implementation of a Virtual Currency Based Incentive Mechanism in a Direct Connect Peer-to-Peer System. In *SATNAC*, 2010.

[15] Salman A. Baset and Henning Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *Computing Research Repository*, abs/cs/041, 2004.

[16] Daniel S. Bernstein, Zhengzhu Feng, Brian Neil Levine, and Shlomo Zilberstein. Adaptive Peer Selection. In *The Second International Workshop on Peer-to-Peer Systems*, pages 237–246, 2003.

[17] Yuh-Ming Chiu. *On the Performance of Peer Selection Strategies in Stochastic Peer-To-Peer Networks.* Proquest, Umi Dissertation Publishing, 2011.

[18] Yuh-Ming Chiu and Do Young Eun. Minimizing file download time in stochastic peer-to-peer networks. *IEEE/ACM Transactions on Networking*, 16:253–266, 2008.

[19] Yuh-Ming Chiu and Do Young Eun. On the performance of content delivery under competition in a stochastic unstructured peer-to-peer network. *IEEE Transactions on Parallel and Distributed Systems*, 21:1487–1500, 2010.

[20] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. *SIGCOMM Computer Communication Review*, 38(4):363–374, 2008.

[21] Bram Cohen. The BitTorrent Protocol Specification, 2009. `http://bittorrent.org/beps/bep_0003.html`.

[22] Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *ACM SIGCOMM Conference*, pages 15–26, 2004.

[23] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-generation Onion Router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[24] T. S. Eugene and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *IEEE INFOCOM*, pages 170–179, 2001.

[25] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-Peer Communication Across Network Address Translators. In *USENIX Technical Conference*, pages 179–192, 2005.

[26] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. A Measurement Study of Napster and Gnutella As Examples of Peer-to-peer File Sharing Systems. *SIGCOMM Comput. Commun. Rev.*, 32(1):82–82, jan 2002.

[27] Yoran Heling. Some Measurements on Direct Connect File Lists. January 2014. `http://dev.yorhel.nl/doc/dcstats`.

[28] Raj Jain. *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling.* John Wiley & Sons, May 1991.

[29] Gordon Mohr Justin Chapweske. Tree Hash EXchange format (THEX), March 2003.

[30] Mohamed Ali Kaafar, Franc ois Cantin, Bamba Gueye, and Guy Leduc. Detecting Triangle Inequality Violations for Internet Coordinate Systems. In *Future Networks*, June 2009.

[31] R. Kumar and K.W. Ross. Peer-Assisted File Distribution: The Minimum Distribution Time. In *Hot Topics in Web Systems and Technologies, 2006. HOTWEB '06. 1st IEEE Workshop on*, pages 1–11, 2006.

[32] Jonathan Ledlie, Paul Gardner, and Margo I. Seltzer. Network Coordinates in the Wild. In *Networked Systems Design and Implementation*, 2007.

[33] Keqin Li. Analysis of random time-based switching for file sharing in peer-to-peer networks. In *Symposium on Parallel and Distributed Processing*, pages 1–8, 2010.

[34] Keqin Li. Reducing Download Times in Peer-to-Peer File Sharing Systems with Stochastic Service Capacities. In *Symposium on Parallel and Distributed Processing*, pages 608–617, 2011.

[35] Richard T. B. Ma, Sam C. M. Lee, John C. S. Lui, and David K. Y. Yau. Incentive and Service Differentiation in P2P Networks: A Game Theoretic Approach. *IEEE/ACM Transactions on Networking*, 14(5):978–991, October 2006.

[36] Karl Molin. Measurement and Analysis of the Direct Connect Peer-to-Peer File Sharing Network. Master's thesis, University of Gothenburg, June 2009.

[37] J. Oikarinen and D. Reed. RFC 1459: Internet Relay Chat Protocol, May 1993.

[38] Michael Piatek, Harsha V. Madhyastha, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Pitfalls for ISP-friendly P2P design. In *The Eighth ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[39] Y Yang R. Alimi, R. Penno. ALTO Protocol Internet-Draft v13. Technical report, IETF, September 2012.

[40] Matei Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network. In *First International Conference on Peer-to-Peer Computing*, pages 99–100, aug 2001.

[41] Pablo Rodriguez and Ernst W. Biersack. Dynamic parallel access to replicated content in the internet. *IEEE/ACM Transactions on Networking*, 10:455–465, 2002.

[42] Anna Satsiou and Leandros Tassiulas. Reputation-Based Resource Allocation in P2P Systems of Rational Users. In *IEEE Transactions on Parallel and Distributed Systems*, volume 21, pages 466–479, April 2010.

[43] Mischa Schmidt, Jan Seedorf, Stefano Napolitano, RosarioG. Garroppo, Andrea Cavaliere, Thilo Ewald, Armin Jahanpanah, Zbigniew Kopertowski, Marcin Pilarski, and Pawel Grochocki. Experiences with large-scale operational trials of ALTO-enhanced P2P filesharing in an intra-ISP scenario. *Peer-to-Peer Networking and Applications*, pages 1–21, 2012.

[44] Jacek Sieka. *ADC Protocol version 1.0.1*, 2008. `http://adc.sourceforge.net/ADC.html`.

[45] David A. Turner and Keith W. Ross. A Lightweight Currency Paradigm for the P2P Resource Market. 2003.

[46] Fredrik Ullner. *ADC Extensions version 1.0.6*, 2010. `http://adc.sourceforge.net/ADC-EXT.html`.

[47] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gn Sirer. KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing. In *Workshop Economics of Peer-to-Peer Systems*, 2003.

[48] Yonghe Yan, Adel El-atawy, and Ehab Al-shaer. Ranking-Based Optimal Resource Allocation in Peer-to-Peer Networks. In *IEEE INFOCOM*, pages 1100–1108, 2007.

[49] Amgad Zeitoun, Hani Jamjoom, and Mohamed El-Gendy. Scalable parallel-access for mirrored servers. In *The 20th IASTED International Conference on Applied Informatics*, pages 93–98, 2002.