Research Project

Peer Selection in Direct Connect

Author Yoran Heling
Supervisor Georgios Karagiannis

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

April 5, 2013

Contents

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 [22], Skype [15], Gnutella [37] and Direct Connect [3].

A P2P system is a distributed network of peers, working together to provide a set of services to the user. Applications of P2P systems include:

File distribution Scalable and efficient distribution of files. BitTorrent is a popular example of a file distribution application that allows a regular internet user to share large files with other users on the internet, without having to set up and maintain a complex server infrastructure and pay expensive bandwidth bills.

Instant messaging P2P systems can be used to provide secure and/or reliable instant messaging between users. Direct Connect and Gnutella are examples of file sharing applications that include instant messaging functionality. Skype is a popular P2P application that provides scalable instant messaging, voice chat and video conferencing.

Music streaming The music streaming service Spotify [9] uses a P2P architecture to supplement their available bandwidth.

Searching Many file sharing applications such as Kazaa [31], Direct Connect and Gnutella provide functionality to allow a user to find files provided by the other users on the network. Projects like JaCy [5] provide a distributed web search.

P2P systems can be classified in two categories: Hybrid and pure P2P systems. A pure P2P system (figure 1.1c) is one that does not rely on a centralised entity or function. This type of P2P system should remain functional for as long as there are still some peers alive. Examples of pure P2P systems are Freenet [11] and Gnutella. A hybrid P2P system (figure 1.1b) is one that uses a distributed architecture for some of its services, but still relies on a centralized entity to function. For example, the original BitTorrent protocol may rely solely on other peers in the network to distribute content, but it still requires one or more centralized *trackers* to discover these peers.

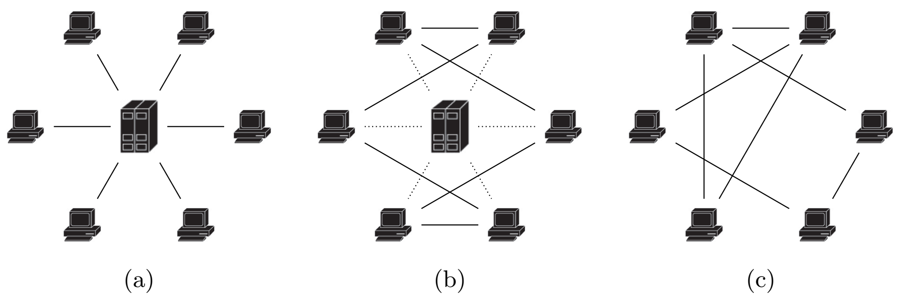


Figure 1.1: Three network architectures. (a) is a centralised, non-P2P network. (b) is a hybrid and (c) a pure P2P architecture.

An underlying P2P system may provide a variety of services to the application. The following services are common to many file-sharing networks:

Content and peer discovery Finding content on the network or finding other peers to communicate with. An application may search for peers that match certain criteria, such as a specific user to chat with, or to find a peer that has a certain file which you are interested in.

Connection establishment When two peers wish to communicate with each other, the P2P network will provide facilities to ensure that these two peers can open a connection to each other. Some networks can route data through other peers if a direct connection is not possible.

File transfers Each peer has some local files or file chunks, and allows other peers to download these file (chunks).

Peer selection Once an application has obtained a list of peers which it could download from, it will decide from how many peers to download from simultaneously, and will then select suitable peers for the file transfer.

1.1 Problem Statement and Research Questions

This report focuses on the latter two services, file downloading and peer selection. Determining from how many and from which peers to download from involves a number of trade-offs. Downloading from a single peer at a time may result in suboptimal downloading performance, whereas downloading from all available peers simultaneously will severely impact the scalability of the network. So a downloading peer has to make a selection among the available peers. The distributed nature of P2P networks means that no peer has a holistic view of the entire network, and information about properties such as where each peer is located and what bandwidth they can provide is not known in advance.

The primary goal of peer selection is to select a minimum number of peers to download from while optimizing for fast download speeds and low resource overhead. This report attempts to answer the following key questions:

1. Which peer selection mechanisms exist?
2. What is Direct Connect?

3. Which peer selection mechanisms are suitable for Direct Connect?
4. What challenges exist for peer selection in Direct Connect?

1.2 Organisation of this Report

The organisation of this report follows the aforementioned research questions. Chapter 2 attempts to answer question 1, and discusses various existing peer selection mechanisms that may be relevant to Direct Connect. Chapter 3 follows by answering question 2 with an overview of the Direct Connect network and protocol. These two findings are combined in chapter 4, which answers question 3 by expanding on which peer selection mechanisms are suitable for Direct Connect. Chapter 5 answers question 4 with a discussion on the challenges of integrating a suitable peer selection mechanism in Direct Connect. Chapter 6 ends this report with conclusions and future work.

Chapter 2

Peer Selection Mechanisms

When downloading a file from a P2P network, the downloading peer has obtained a list of source peers from which it could download from. It then has to make a decision: How many source peers should it download from? And, perhaps even more important, *which* source peers should it download from? This problem is called *peer selection*: The downloading peer has to make a selection on the available source peers.

An important feature of a peer selection mechanism is whether it employs a Parallel Downloading (PD) scheme or Single Downloading (SD) scheme. In the SD scheme, a file will only be downloaded from a single source peer at a time. PD is a technique where the file to be downloaded is divided into several chunks. Each chunk may be downloaded from a different source peer, and multiple chunks may be downloaded in parallel. This technique improves download performance and offers better resilience to the sudden departure or failure of source peers. The scalability of the overall network, however, is affected negatively if each peer attempts to download from too many source peers at once. The Degree of Parallelism (DoP) indicates the maximum number of simultaneous download transfers from a peer. A too low DoP may prevent the downloading peer from reaching optimal performance, whereas a too high DoP will impact the scalability of the network. SD can be seen as a special case of PD with a DoP of 1.

This chapter gives an overview of existing solutions to the peer selection problem, and is organized as follows. Section 2.1 discusses solutions employing topology estimation, section 2.2 follows with solutions that attempt to estimate actual download performance, and section 2.3 discusses general PD schemes. Finally, a short overview of incentive-based solutions is given in section 2.4.

2.1 Topology Estimation

The goal of topology estimation is to make the application aware of the underlying network topology, allowing it to choose peers for which cheaper or shorter network routes exist. Topology estimation techniques do not provide a complete answer to the peer selection problem, since they merely rank the source peers based on some criteria. The downloading peer still has to make a decision on what PD scheme to use.

2.1.1 Oracle-based

In [13] a description is given of how the routing used by P2P overlay networks ignores all the efforts made by Internet Service Providers (ISPs) to improve the routing of the underlying network. They describe how the ISPs, who are in charge of ensuring that the network routes remain efficient, are the most knowledgeable about the network topology and the routing costs between a set of hosts.

They proposed an ISP-operated Oracle: A service run by the ISP that, on request, ranks a given set of peer addresses by the preference of the ISP of communicating with those peers. P2P applications can obtain IP addresses of other peers as they normally would. When deciding which peers to interact with, the application sends its list of IP addresses to the Oracle, and connects to those the peers that the Oracle ranked highly.

The metrics used for ranking are decided by the ISP. Possible metrics mentioned by [13] are:

- Whether the peers are inside the same Autonomous System (AS).
- The number of hops between ASes, as provided by the Border Gateway Protocol (BGP).
- Distance of the peer to the edge of the AS according to the Interior Gateway Protocol (IGP) metric.
- Geographical information.
- Performance information, such as expected delay or available bandwidth.
- Measured link congestion.

The IETF Application-Layer Traffic Optimization (ALTO) Working Group [2] has focussed its research on Oracle-based peer selection and has developed a protocol standard for the communication between P2P applications and Oracle services [36].

The protocol has been implemented and evaluated in [40]. The evaluation is based on a modified BitTorrent client that uses the Oracle-provided information to select its neighbouring peers. The results show that the use of an Oracle allows the ISP to save significant costs by guiding the peer selection process to use cheaper routes, even when only a subset of the peers query the Oracle for routing information. No significant change in downloading times has been observed.

It is argued in [35] that the incentives of ISPs do not necessarily align with the common goal of providing shorter routes. A Tier-1 AS would profit by recommending *more* inter-AS traffic, as their customers pay for the extra transit traffic. Other ISPs have no incentive to optimize traffic once it has left their own AS, since they are not responsible for the bandwidth costs beyond that.

2.1.2 Synthetic coordinate systems

Applications can also construct a global coordinate system, as proposed with Global Network Positioning (GNP) [24]. In their solution, the Internet is mapped onto a fixed geometric space on which each peer has an absolute coordinate. The architecture for obtaining these coordinates is split in two parts.

In the first part, a small set of special peers called *Landmarks* are created and initialized with coordinates that reflect their Round Trip Time (RTT) distances. In the second part of the architecture (figure 2.1), regular peers query the coordinates of each of these landmarks, use Internet Control Message Procotol (ICMP) ping messages to measure the RTT, and use that to calculate their own coordinates.

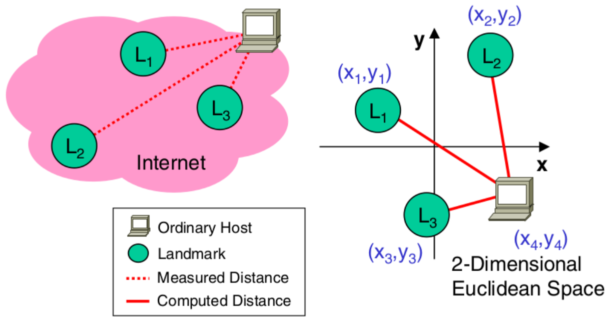


Figure 2.1: The second part of the GNP architecture. An ordinary host queries three landmarks on the internet and positions itself in a Euclidean Space based on RTT measurements. Copied from [24].

The requirement of deploying special Landmark peers makes it less suitable for distributed P2P applications. Vivaldi [23] proposed a fully distributed coordinate system, which does not rely on any such special peers. In their solution, each peer continuously adjusts its own coordinate based on the measured RTTs and obtained coordinates from other peers in the system.

In order to rank a list of peers, a downloading peer connects to each source peer in its list in order to obtain its coordinate, and then ranks the list based on each peers' distance to the downloading peer.

Network coordinate systems suffer from a few problems, including the added signalling overhead of maintaining coordinates by continuously measuring RTTs and obtaining the coordinates of other peers. The accuracy of a coordinate system on large-scale deployment may suffer due to the following problems [28]:

Triangle inequality violation The use of Euclidean distances makes the assumption that the triangle inequality is not violated. For instance when three peers measure each of their RTTs, it is assumed that these measured values can form a triangle. This is not always true in practice. An example is given in [27] with three peers A, B and C, with measured latencies of 1 ms between A and C, 2 ms between C and B and 5 ms between A and B. It is impossible to form a triangle with sides of length 1, 2 and 5, and thus impossible to map the relation between these peers in Euclidean space.

Churn Peers may join and leave the network at any time. It takes in the order of a few seconds to a minute before a new peer has settled on a stable coordinate. Peers may leave before they have obtained a stable coordinate, and synchronising with such short-lived peers will increase the error of the coordinates of other peers as well.

Drift The centroid of the coordinate system continuously drifts away from the origin. While this is not a problem for ranking peers by their relative distances, it does limit the amount of time that a coordinate can be cached. Coordinates that have not been synchronised too often will be inaccurate.

Malicious behaviour Distributed network coordinates require that each peer cooperates by providing valid information. Malicious peers can disrupt the accuracy of the coordinate system by providing false data.

Latency variance RTT measurements between two peers are not always stable. When an RTT of 10 ms has been measured between two peers, another measurement a few seconds later may give an RTT value of 100 ms.

Solutions to several of the above problems are proposed in [28], but these increase the complexity and overhead and do not address the problem of malicious behaviour.

2.1.3 CDN-based proximity

Ono [20] proposed 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 [7] 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, in terms of RTT or available bandwidth.

Ono makes use of the idea that, if two peers query the domain name of the CDN and are being redirected to the same server, then they must be close to each other. In their solution, each peer thus queries a number of large CDNs and maintains a ratio map of the received responses. Peers then exchange each others' ratio maps and use cosine similarity to determine the approximate distance between each other.

The overhead of continuously querying a set of DNS servers amounts to approximately 18 kB upstream and 36 kB downstream per day, and the overhead of exchanging the ratio maps between a downloading peer and its potential source peers is expected to be relatively small. The use of CDN-based information only works well to rank peers that are relatively close to each other. When none of the source peers receive DNS responses similar to the downloading peer, their ratio maps will be orthogonal and no proximity information can be determined.

Ono has been implemented as a plugin for the Vuze BitTorrent client and deployed among over 100,000 users. The results indicate significant improvements in shorter path selection and in the resulting transfer rates when compared to classic BitTorrent peer selection.

2.2 Performance Estimation

Performance estimation is a category of techniques that try to get an estimate on how well a connection between two peers is going to perform. These estimations can be based on RTT measurements or by actually transferring some data over the network and measuring its bandwidth. Unlike the topology estimation techniques discussed in the previous section, performance estimation does not use network proximity: If there is a fast peer on the other side of the globe and a slow peer available in the same building of the downloading peer, performance estimation will favour the faster peer.

Some performance estimation techniques implement a PD scheme, while others are only described in the context of SD. The PD scheme used will be described separately for each solution.

2.2.1 Local RTT measurements

The authors of [47] discuss the client-server situation in which an FTP client can choose to download a file from a known set of servers that each provide the same file (mirrors). They proposed a solution where the client measures the RTT for each mirror, and then selects a number of highest-ranked mirrors to download from.

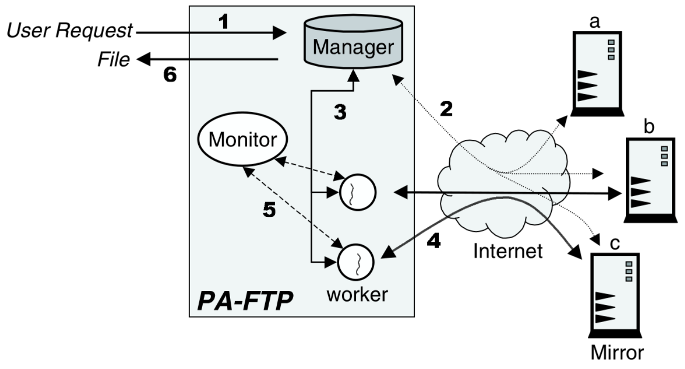


Figure 2.2: Architecture of the scalable parallel FTP downloading client. The manager thread queries the RTT times of the server (2), and assigns chunks to be downloaded to each worker thread (3,4). Progress is monitored by the monitor thread (5). Copied from [47].

Their PD scheme is as follows. The client splits the file into DoP chunks of proportional size to the measured RTT values, and requests these chunks simultaneously from the selected mirrors. A separate manager thread monitors the download progress for each of the chunks, and dynamically re-assigns the chunk of the slowest mirror to a faster one that is about to finish. Figure 2.2 displays the architecture of their client implementation.

As a result, the client downloads from DoP mirrors simultaneously at the start of the file transfer, but as the faster mirrors finish their chunks the slower mirrors will get disconnected. Towards the end of the file transfer, less mirrors are used for downloading. When compared to random mirror selection, their solution requires a lower DoP to saturate the bandwidth of the client, and therefore improves the scalability of the network.

As their solution was designed with static FTP mirrors in mind, the assumption is made that all mirrors are known to the client before starting the download, and that each mirror stays online for the entire duration of the transfer. These assumptions often do not hold in P2P systems.

2.2.2 Machine learning

In [16] the peer selection process of a Gnutella client is studied, and an adaptive solution that employs machine learning techniques is proposed. Their solution consists of two phases.

In the first phase, passively collected information is used to rate each source peer in terms of the expected transfer time. This information consists of attributes provided by the P2P network, such as the peers current load, whether it has successfully uploaded a file before, whether it's behind a firewall or not, and its indicated maximum available bandwidth. In order to automatically learn the correlation of this information with the expected transfer times, these attributes have been collected for a large set of peers and their actual bandwidth has been measured. This dataset has been used as training data in a machine learning algorithm with the goal of constructing a decision tree (figure 2.3).

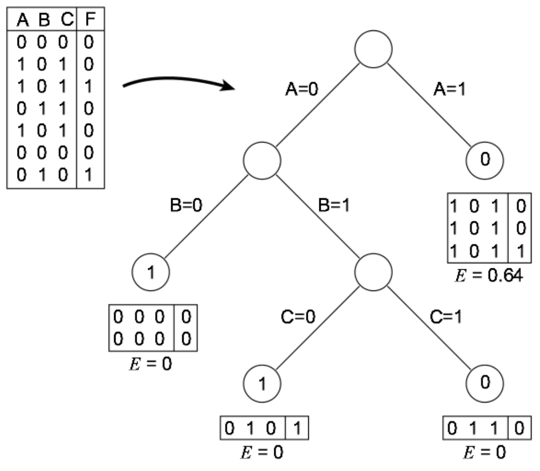


Figure 2.3: Example data set and a possible decision tree constructed from it. The decision tree is a "compressed" form of the data set, and provides an *if-then-else*-style function for mapping the input values **A**, **B** and **C** to an output value **F** and a measure for the confidence of this value **E**. Copied from [16].

The resulting decision tree is then used to construct a function that, given the passively collected attributes of a newly discovered peer, returns its expected upload rating. This rating is relative to the peers in the learning data, indicated in values of VLS (Very Likely Slow), LS (Likely Slow), U (Uncertain), LF (Likely Fast) and VLF (Very Likely Fast). This function allows the downloading peer to rank a list of source peers without requiring prior network measurements.

In the second phase, the downloading peer uses the ranked list of source peers to perform a sequence of partial downloads, eventually settling on one peer. A Markov Decision Process (MDP) is used to model and direct the choices of the downloading peer. The behaviour of the downloading peer is modelled in a *connecting* and a *downloading* phase. In the connecting phase, the peer is attempting a connection with the next source peer. This phase lasts at most 3 seconds, if no connection has been established in that time a new source peer is chosen. If a connection has been successful, the downloading phase begins. If 3 seconds have passed in this phase, the peer will commit to download the rest of the file from this source peer, and the peer selection process has ended. An MDP guides the peer through this process by choosing between two actions: Either to continue with the current peer, or to start over in the connecting phase with a new peer. Such an action is chosen every 0.5 seconds in the connecting phase and every 1.0 seconds in the downloading phase. Time is used as the cost function, in order to make a decision that minimizes the total transfer time of the file.

Their solution uses a large dataset in order to generate the decision tree, and they have shown that peers with different network configurations generate a different decision tree, specifically tailored towards the configuration of the peer. As such, it is preferable to have each peer generate its own decision tree. In order to do this, it will need to actively use the network for a substantial amount of time to obtain enough data, which is not practical for many users. Furthermore, the machine learning approach reacts slowly to changes in the network or the to configuration of the peer.

Most of the attributes used for ranking each peer are provided by the peer itself. A malicious peer is thus capable of influencing its own ranking by observing which attributes are favoured by the decision tree and sending false information to the network.

The proposed downloading process uses the SD scheme. While the authors claim that their solution can be extended to implement the PD scheme, they do not describe how.

2.2.3 Random periodic switching

Observations in [18, 17] indicate that P2P systems often provide worse downloading performance in practice than systems using a centralised network architecture. They attribute this to two factors: Spatial heterogeneity in the available service capacities, and temporal correlations in the service capacity of a given source peer.

Heterogeneity refers to the idea that service capacities from different source peers are very different. Each peer is connected to the internet with a different physical connection, each with different bandwidth constraints and each with different network properties. These factors are relatively constant, and don't change over the timescale of a typical P2P session.

There are, however, factors that do not remain constant within a P2P session. The provided service capacity of a source peer may fluctuate drastically even over time periods of a few minutes. These fluctuations may be caused by the varying number of peers downloading from a single source peer at a time, other applications running on the PC of the source peer that cause a sudden increase in network usage, or temporal congestion at any link in the network. Figure 2.4 displays how the available bandwidth between two peers typically changes over time.

In order to remove the effects of both heterogeneity and correlation from the file download time, they propose the following downloading algorithm. The downloading peer randomly chooses a source peer to download from, and keeps using this peer for a fixed duration. After a timeout (e.g. 5 minutes), the downloading peer disconnects the current source peer, chooses a new random peer, and continues downloading from that one. This repeats until the download has completed.

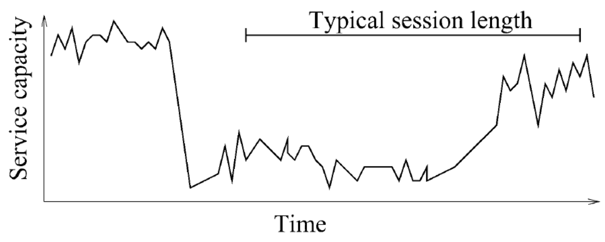


Figure 2.4: Typical variation in end-to-end bandwidth between two peers. Time scale is in the order of minutes. Copied from [18].

Unlike to chunk-based assignments, this random periodic switching algorithm guarantees that the downloading peer will not get stuck downloading from a very slow source peer, thus significantly decreasing the total file download time in such a case [29]. The algorithm is explained for the SD scheme, no suggestions are made on how to use it together with the PD scheme.

Random periodic switching does not provide an optimal download strategy. The uniformly random nature of the peer selection mechanism will not favour peers with more service capacity over peers with less available bandwidth. The overhead of opening a connection to a new peer is not taken into account, either.

2.2.4 Biased random periodic switching

An improvement on random periodic switching was proposed in [19, 17]. Instead of connecting to only a single source peer at a time, they employ a PD scheme. In each time period, the downloading peer opens a random number of parallel connections to a set of selected source peers, and keeps downloading from them within that period. As in random periodic switching, the peers to connect to are chosen randomly from the set of source peers. However, instead of applying the uniform random selection used in the former solution, each source peer is now assigned a value describing the probability that the downloading peer will connect to that source peer in each period. The probabilities are chosen such that the sum of all probabilities is equivalent to the desired DoP. This means that, while the actual number of active connections at any time may be lower or higher than the desired DoP, the average number of active connections should still converge to the DoP.

They have derived an optimal probability assignment algorithm that minimizes the average file download time, while taking into account the heterogeneity and stochastic fluctuation of the network and the competition between all downloading peers in the network. These probabilities are obtained by solving the following optimization problem:

$$\max \frac{\sum_{j \in S}[1-(1-p_j)^{|D|}]c_j}{\sum_{j \in S} c_j}$$
$$\text{s.t.} \sum_{j \in S} p_j = L, 0 \le p_j \le 1$$

Where S denotes the set of source peers, $|D|$ the number of competing downloading peers, L the desired average DoP, and c_j and p_j the expected service capacity and assigned probability for peer j, respectively. A solution for p_j is given in [19], but has been omitted here for brevity.

As this algorithm requires knowledge about $|D|$ and c_j, it is impractical for use in a distributed P2P environment. They therefore propose a second algorithm that allows the downloading peer to create estimates for these values while downloading from each source peer in a sequence of periods. In order to estimate $|D|$, the downloading peer needs to be able to obtain information about the number of peers that each source peer is uploading to.

While their solution includes a PD scheme, they do not provide an algorithm for assigning file chunks among the selected source peers. The overhead of opening a connection to a new peer is not taken into account in their analysis, nor are the effects of churn.

2.2.5 Chunk-based probing

Another improvement on random periodic switching has been proposed in [30]. Their solution divides the file to be downloaded into equally-sized chunks, and dynamically assigns these chunks to source peers. Only one chunk is downloaded at a time.

Their proposed algorithm works in two stages. In the first stage, the downloading peer downloads one chunk from each source peer in sequence and measures the received capacity of the source peer. If this measured capacity is higher than a certain threshold, the peer is considered a *high-capacity* source peer, otherwise it is a *low-capacity* peer. The first stage ends when all file chunks have been downloaded, when a high-capacity source peer has been found, or when all source peers have been probed. In the second stage, the download peer will continue to download the file from the high-capacity source peer if one was found, or the highest-capacity peer if all source peers were probed.

The previously mentioned threshold determines when a high-capacity source peer is found. If it is too low, then the download may continue from a slow source peer even when a faster one exists. If the threshold is too large, then the time spent in the probing stage will be too large. They provide an equation for obtaining the optimal threshold given the number of source peers, number of chunks and the expected (uniform) distribution of the service capacities of the source peers. In practice, the expected service capacities are not known in advance and are not uniformly distributed.

While the authors explain the problem of peer selection in the context of unpredictable variations in the available service capacities, they provide no indication of how their solution deals with these problems. In fact, it is perfectly possible for a source peer to provide a high service capacity in the probing phase, but then degrade significantly after the downloading peer has selected it for downloading. It is also possible for the downloading peer to get stuck on a very slow source peer in the probing phase, thus causing the download time to increase indefinitely. Chunk-based probing makes use of the SD scheme, PD has been noted as future research.

2.3 Parallel Downloading Techniques

Many of the peer selection mechanisms discussed in the previous section already included algorithms for PD. However, PD can also be viewed as a separate problem. Given a file you want to download and a set of source peers which have that file, the previous solutions provided an answer to the question of *which* peers you should connect to, and only some mentioned *how* the file should be split up in chunks and assigned to each source peer. Parallel downloading mechanisms only provide an answer to the latter question.

2.3.1 Dynamic Parallel Access

The authors of [38] discuss the situation in which a HTTP client wishes to download a file that has been replicated over multiple HTTP servers. They propose a very simple strategy in which the client divides the file into many small chunks (number of chunks $\gg$ the number of servers). The client then requests one chunk from every server. As soon as a chunk has been completed from one server and the file has not been completely downloaded yet, a new chunk is immediately requested from the same server.

In this scheme, the file will thus be downloaded from *all* servers simultaneously. As each chunk is dynamically requested from a server only once it has finished with a previous chunk, faster servers are automatically assigned a larger portion of the file. A major benefit of using all servers, in the previously described context, is that the downloading peer no longer has to make a selection at all.

A problem with this approach is that there is a small delay, corresponding to one RTT, between finishing a previous chunk and starting a new one (figure 2.5). The authors propose to use a technique called *pipelining* to avoid this delay. The downloading peer can send a requests for the next chunk a little before the current chunk has finished, thus allowing the server to immediately follow up with the new chunk without delay.

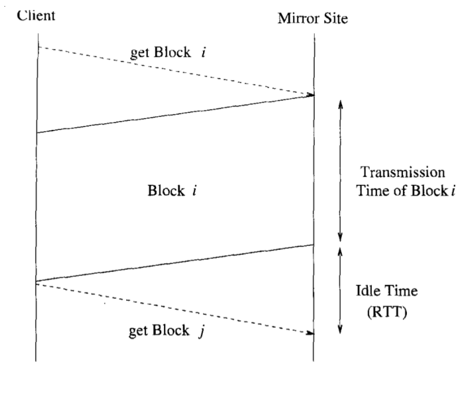


Figure 2.5: Idle time between two block (chunk) requests. Copied from [38].

Another problem is the issue with the last stage of the download. Not all servers finish their chunk at the same time, so it's possible that the peer gets stuck on a few slow servers near the end of the transfer. To remedy this, they propose that the peer requests any remaining chunks from multiple servers simultaneously, and throws away any data offered by slower servers. They argue that the bandwidth wasted on duplicate data would not be very significant, thanks to the small size of each chunk and the fact that any disconnected servers have not sent a full chunk yet.

Two major problems not addressed are the signalling overhead and scalability issues. Due to the small chunk size, a peer has to send out requests for new chunks rather often. Each request and response add a bandwidth overhead that becomes more significant the smaller the chunk sizes are chosen. The problem of scalability applies to both the downloading peer and the server. On the server end, if many peers use this parallel downloading technique, then the server has to handle many more TCP connections than strictly necessary, and can easily get overloaded with requests. The service provided by each individual server degrades significantly as more peers make use of it. Even on the side of the downloading peer, scalability becomes an issue for files that have been replicated over many servers — peers do not always have powerful machines and may not be capable of handling hundreds of connections. A peer may also be behind a low-end router that is only capable of handling a limited number of TCP connections. Thus, in practice, a peer will still need to make a selection among the available peers.

2.3.2 Adaptive Dynamic Parallel Downloading

Adaptive Dynamic Parallel Downloading (adPD) [48] and Speculative Parallel Downloading (sPD) [26] are two similar PD schemes that minimize the required signalling overhead. They do this by dynamically adjusting the chunk size according to the observed download speed of each source peer.

The algorithm works as follows. The downloading peer divides the file into m equally sized chunks, where m is equivalent to the number of source peers. Each chunk is then requested from a source peer. When a chunk has been successfully downloaded, the downloading peer calculates the expected download time for each active chunk and reassigns the source peer that just finished to help out with downloading the chunk with the longest expected download time. The remaining part of this chunk is divided in two parts with proportional size to the measured download speeds of the source peers. Since the first part of this chunk is still being downloaded from the source peer, no signalling is required. The second part of the chunk is requested from the peer that just finished. In the special case that the remaining chunk size is smaller than a certain threshold, the chunk will not be split in two, but will be assigned in full to the faster peer instead.

While the authors of [48] mention that their solution does not require the downloading peer to interrupt any ongoing chunk downloads, they do not describe what happens after a chunk assigned to a peer has been split in two — thus decreasing the chunk size that the peer has to offer — and the peer has completed its assigned chunk. In that situation, the source peer is not aware that its assigned chunk size has decreased, and will send file data that has been reassigned to another peer. In [26] it is made clear that their solution does require interrupting the download process.

2.3.3 Minimum-Signaling Maximum-Throughput

Minimum-Signaling Maximum-Throughput (MSMT) is an algorithm proposed by [45]. The goal is again to allocate chunks to different source peers in such a way that it minimizes signaling overhead while obtaining the maximum throughput. MSMT achieves this by obtaining a Bayesian estimate of the available bandwidth from each source peer.

The algorithm works as follows. The downloading peer first downloads a small fixed-size chunk (say, a few kilobytes) from each source peer in parallel. While doing this, it measures the average received service capacity for each peer and tries to obtain an estimate on the size of each peers' upload queue. This estimate is later used in updating the Bayesian estimate of the bandwidth for each peer. The remaining part of the file is then divided in chunks proportional to the measured bandwidth of each peer, and each chunk is requested for downloading.

When one chunk finishes downloading, the downloading peer again calculates the average service capacity received from each source peer for the requested chunk, and updates its Bayesian estimate of the available bandwidth of the source peer. It then interrupts all ongoing downloads, and uses the bandwidth estimates to assign new chunks of proportional size to each source peer. The details of the bandwidth estimation calculation are not relevant for this discussion and have been omitted here for brevity. A detailed explanation can be found in [45].

MSMT has been evaluated in both simulations and implemented and tested on the Planetlab platform [21]. The results indicate that the use of Bayesian estimates instead of only relying on the most recent measured average bandwidth improves the assignment of chunk sizes to source peers, and therefore decreases the number of chunk re-assignments and their associated signalling overhead by around 10 to 30%. Their simulations and measurements did not include the actual file download time experienced by the downloading peer, and no information is given on how significant the decrease of signaling overhead is on these downloading times.

MSMT only deals with assigning chunk *sizes* to source peers, no solution is given on how these chunks need to be allocated within the file. Their chunk size assignment algorithm does not guarantee that it is even *possible* to allocate contiguous chunks in each (re)assignment, and may cause a single chunk assignment to become fragmented over different file blocks. Chunk fragmentation will require additional signaling overhead in order to tell the source peer which byte ranges it should send. This problem has not been mentioned by the authors, and it is unclear whether this overhead has been included in the test results.

The initial probing phase, where a fixed-size chunk is downloaded from each source peer, may stall the downloading process for an indefinite amount of time if there is a very slow source peer.

As with adPD and sPD, MSMT requires the chunk downloads to be interruptible in order to perform the reassignments.

2.4 Incentive-Based Peer Selection

A common problem with P2P file sharing networks is that peers do not always have a strong incentive to *upload* data. Providing other peers with the data they want costs bandwidth and computer resources, but does not provide an immediate gain to the user who is uploading. Yet, in order for the overall P2P network to function, there must be enough users who do upload data.

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.

Many such peer selection algorithms are described in literature [42, 12, 46, 39, 44, 32], but due to the limited applicability of incentive-based peer selection mechanisms to the Direct Connect network, only the algorithm used in the original BitTorrent implementation will be described in order to give a general idea about how these work.

2.4.1 BitTorrent

The BitTorrent file sharing protocol [22] employs, among other techniques, incentive-based peer selection in a tit-for-tat fashion. In BitTorrent, a file is divided into many equally-sized chunks, called *pieces*. Peers exchange bit masks with each other to indicate which pieces each peer has. Peers can then request pieces from each other.

The peer selection algorithm works with a mechanism called *choking*. Each peer may be connected to many other peers, but only a small number of connections is actually used for data transfers. These connections are *unchoked*, that is, peers can only exchange pieces on an unchoked connection. All other connections are said to be *choked*, and are used only to exchange information about how the pieces are distributed among the peers and which peers are interested downloading new pieces.

Each peer chooses a number of random peers to connect to, exchanges the bit masks, and then unchokes a small number of connections — typically four — in order to *trade* pieces. This trading provides an incentive for both peers to upload pieces. If either of the peers refuses to upload a piece, the connection may be choked again and neither can download. In order to find faster peers and to give new peers a chance to get started, a mechanism called *optimistic unchoking* is used. At any time, there will a single connection that is unchoked regardless of whether the other peer is uploading or not. Every 30 seconds, the peer will choke the connection with the lowest upload speed and randomly select another connection to unchoke.

Chapter 3

Overview of Direct Connect

Direct Connect (DC) [3] is a hybrid Peer-to-Peer file sharing network developed in the late 1990s. It provides the following features to its users:

User management Everyone can obtain 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 (IRC). 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, TTH) 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.

There exists two separate network protocols for DC. The original protocol is commonly called *NMDC*, named after its original implementation, NeoModus Direct Connect. While this protocol has never been officially documented, reverse-engineering efforts have given rise to a number of open source clones of the original implementation. As the popularity of the original implementation had fallen, these clones had taken over further development of the protocol and added several extensions in order to improve scalability and reliability. More recently, an alternative protocol called Advanced Direct Connect (*ADC*) has been developed as a replacement for NMDC. ADC attempts to address some issues in the NMDC protocol, and is otherwise very similar to NMDC in terms of concept and terminology. The ADC specification is split up in a mandatory BASE protocol [41] and a separate document specifying the various optional extensions [43].

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, including but not limited to StrongDC++ [10] and EiskaltDC++ [4]. Jucy [6] and Ncdc [8] are two modern clients that are not based on DC++.

The main focus of this research is on the more recent ADC protocol, but many of the ideas discussed in this report apply to NMDC as well.

Section 3.1 provides a high level overview of the direct connect network concepts, followed by a more in-depth explanation of the connection initiation protocol in section 3.2, content and peer discovery in section 3.3 and the behaviour of uploading peers in section 3.4. This chapter ends with an overview of the currently used peer selection strategies in section 3.5.

3.1 High-level overview

The architecture of a DC network is conceptually simple. There are three entities in the DC network: *Clients*, *Hubs* and *Hublists* (Figure 3.1). A Hub is a TCP 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, that provides a list of publicly available hub addresses. A Hublist is not required for the functioning of the DC network, they merely help users with finding new hubs to connect to.

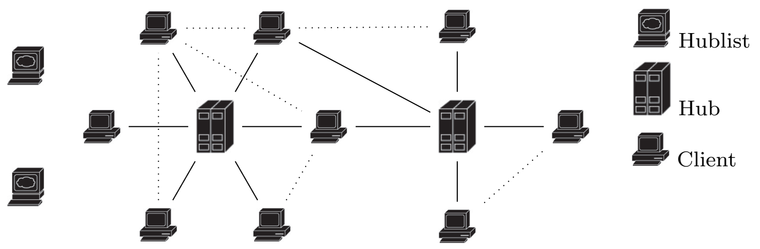


Figure 3.1: Architecture of a small DC network with two hubs and two Hublists. Two clients are connected to both hubs. Some clients that share the same hub have a direct connection with each over, indicated with a dotted line.

On the network level, the hub provides the following services:

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 doesn't 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 3.4), 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. Discussed in more detail in section 3.3.

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 3.2.

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 [34, 14] and to ensure that everyone only shares files allowed according to rules imposed by the hub. Not all hubs are public: DC is also commonly used in local area networks, where users communicate and exchange files using a hub running on the local network itself.

3.2 Connection initiation

The hub is not involved in the actual file transfer process. Instead, peers[2] wishing to exchange files need to be able to create a direct connection with each other. A distinction is made between two client modes: *active* and *passive*. A peer capable of accepting incoming connections from the outside is considered active, whereas a peer behind a firewall or Network Address Translator (NAT) is passive.

The public IP address and the TCP port on which active clients accept incoming connections is part of the user list and thus distributed to all peers. When an active peer A wishes to open a connection with another peer B, the following steps are taken (figure 3.2a):

1. A sends a "Connect To Me" (CTM) message to the hub, indicating that it should be routed to B.
2. The hub routes the CTM message to B.
3. The IP address and TCP port of A are already known to B through the user list provided by the hub, so B can open a TCP connection to A.

The above steps only work if the initiating peer is active. If peer A is passive and peer B is active, the connection is initiated as follows (figure 3.2b).

1. A sends a "Reverse Connect to Me" (RCM) message to the hub, indicating that it should be routed to B.

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

[2]Peers, in this context, are the clients, since those are the only nodes in the DC network capable of doing file transfers.

2. The hub routes the RCM message to B.
3. B replies with a CTM message to the hub.
4. The hub routes the CTM message back to A.
5. A opens a TCP connection to B, using the information provided by the user list.

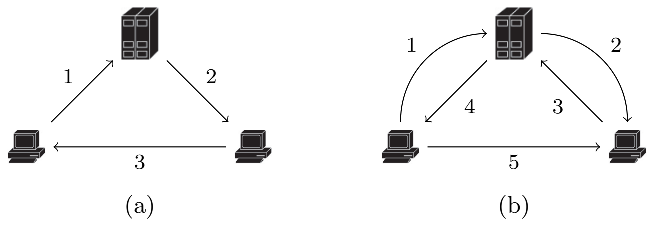


Figure 3.2: Connection initiation between two peers. The initiating peer is active in (a), passive in (b).

Note that, in the passive situation, peer A already knew the public IP address and TCP port of peer B even before sending the RCM message, so it could have skipped step 1–4 and simply opened the connection to peer B without involvement of the hub. This is not done in practice, however. Both the CTM and RCM messages contain a unique *token*. This is a random string that allows the active peer to reliably identify from which hub and from which user an incoming connection originated. The use of this token is mandated in the protocol, so it is not possible to open a connection without involvement of the hub.

In the BASE protocol, it is not possible for peer A and B to create a direct connection with each other when they are both passive. A NAT traversal technique based on TCP Hole Punching [25] has been implemented as the NATT [43] 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 doesn't solve the problem of restrictive firewalls. The steps to open a connection between peer A and peer B are as follows (figure 3.3):

1. A sends an RCM message to the hub, indicating that it should be routed to B.
2. The hub routes the RCM message to B.
3. B replies with a 'NAT' message that includes the local TCP port with which B is connected to the hub.
4. The hub routes the NAT message to A.
5. A attempts to open a TCP connection to B with the received port. Simultaneously, it replies with a "Reverse NAT" (RNT) message to the hub, which includes its own local TCP port.
6. The hub routes the RNT message to B.

7. B attempts to open a TCP connection to A with the received port.

The connection succeeds when either connection attempt from A to B or from B to A succeeds.

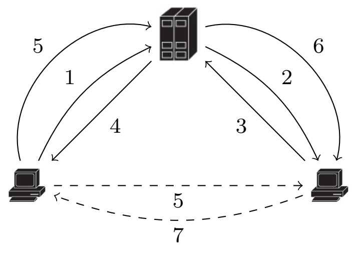


Figure 3.3: Connection initiation between two passive peers using NAT traversal.

Two peers can not exchange files if they are unable to establish a direct connection with each other. The protocol does not provide a mechanism to allow file transfers through intermediate peers as fallback.

3.3 Content and peer discovery

Users can find files by sending a search request to the hub. The hub will broadcast this message to all[3] other peers, which then send back a response if they have found something in their locally shared files. Responses are sent back via the hub if the searching peer is passive or via UDP otherwise. A Response includes the number of available upload slots of the responding client (see section 3.4) and the size, hash and full path of the file.

Such a flooding search is an expensive operation, and almost all hubs have restrictions on how often a peer is allowed to search. The minimum interval between two search requests from a single peer often varies between 15 seconds and one minute.

Search requests come in two forms: Keyword search and hash search. Keyword searches are usually initiated directly by the user in order to find files matching certain filters. Such a search can be used to find, for example, video files smaller than 500 MB with “farbrausch” in the file name, or directories named “linux”. Hash searching is used to find peers who have a file that matches a specific TTH hash, and is often used by clients to (automatically) find alternative peers to download a file from. As an optimization, the BLOM [43] ADC extension allows clients to send a bloom filter of all their file hashes to the hub, giving the hub the opportunity to avoid a broadcast for hash searches.

In addition to flooding, a peer can request the file list of another user. This works like a regular file transfer: The requesting peer initiates a connection with the other peer, and requests the special file `files.xml.bz2`. This XML file is a representation of the directory structure shared by the user and includes the name, size and hash of each shared file. Some clients also include other

[3]Or, in some cases, a subset. E.g. If the searching peer is passive, then the message may not be forwarded to other passive peers.

information like audio/video bit rates or image resolution. The size of this file list grows with the number of files that the user has shared, and usually ranges between a few kilobytes and several megabytes.

The StrongDC++ client has extended the DC network with a global DHT based on Kademlia [33], allowing a peer to perform hash searches without incurring the overhead of flooding the network and without being restricted to the limitations imposed by the hub. This feature, however, is not widely used and only few peers participate in this DHT.

3.4 Uploading peers

Each peer has a set of local files which it shares with the other peers in the network. The number of peers that can simultaneously download from an uploading peer is restricted by the uploading peer. In DC terminology, this is called the *slot count*, where a *slot* represents one upload transfer. This number can be configured by the user, and usually varies between 1 and 20.

Two strategies are currently in use for assigning slots to downloading peers. The most simple strategy is queue-less assignment: When a downloading peer requests a file, it is immediately assigned a slot if the uploading peer has a slot free. The downside of this approach is that it rewards peers that continuously hammer the uploading peer with download requests over peers that have a longer retry interval.

An alternative strategy that is recently being adopted in many clients is to use a FIFO queue. When a downloading peer requests a file, the peer is added to an upload queue. When a slot becomes free, the uploading peer waits for the next peer in the queue to request the file again, and assigns it the slot. A peer is removed from the queue if it has not sent a file request within a timeout, so it must still continue to poll[4].

A peer can bypass this slot assignment process in the following cases.

- An additional number of slots (usually 3) is allocated for files below a certain size[5] or for the special `files.xml.bz2` file, in order to prevent long waiting times for short transfers. These are called *minislots* in DC terminology.
- A user can also *grant* a slot to another user, allowing the other user to download files regardless of how many slots are taken by the uploading peer.
- Many clients have a feature to automatically create new slots if the cumulative upload speed is lower than a certain configurable threshold. This prevents the case where an uploading client has all its slots assigned to slow peers, and still has enough bandwidth available to handle more file transfers.

Due to these special cases, the total number of active upload transfers is not strictly bounded by the configured slot count.

[4]Some clients implement a queue notification mechanism, where the downloading peer is notified as soon as a slot has become free. This is, however, not widely used.

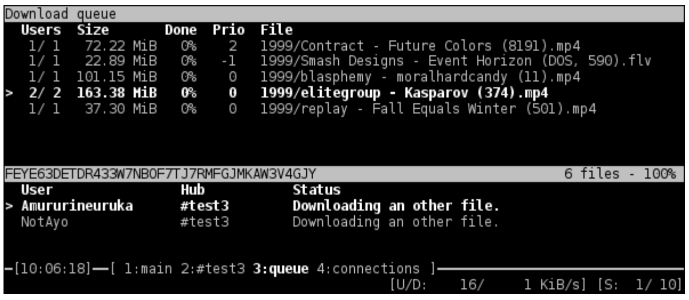
[5]Commonly 64 kB, but the recently released DC++ 0.802 has increased this to 512 kB. In most clients this default size can be overridden by the user.

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 uploading client does not accept a new request for data until the current file transfer has been completed. In at least DC++ (0.802) and ncdc (1.14), the client still reads new incoming messages from the TCP 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 clients can not currently use the pipelining technique discussed in section 2.3.1.

3.5 Peer Selection Strategies

A number of peer selection strategies have been used by both historical and modern DC clients. This section only discusses the mechanisms in use by the current versions of ncdc (1.14), DC++ (0.802), and Jucy (0.86). The various derivatives of DC++ tend to use the same peer selection strategies as the main DC++ client.

All clients implement a *download queue* — a list of files that the user wishes to download (figure 3.4). For each file in the queue, the client maintains a list of peers from which the file can be obtained. It is common for a single peer to be listed with multiple files in the queue. All clients provide functionality to allow the user to prioritize the downloading of certain files over others.



```
Download queue
  Users  Size       Done  Prio  File
   1/ 1   72.22 MiB   0%     2  1999/Contract - Future Colors (8191).mp4
   1/ 1   22.89 MiB   0%    -1  1999/Smash Designs - Event Horizon (DOS, 590).flv
   1/ 1  101.15 MiB   0%     0  1999/blasphemy - moralhardcandy (11).mp4
>  2/ 2  163.38 MiB   0%     0  1999/elitegroup - Kasparov (374).mp4
   1/ 1   37.30 MiB   0%     0  1999/replay - Fall Equals Winter (501).mp4



FEYE63DETDR433W7NBOF7TJ7RMFGJMKAW3V4GJY                        6 files - 100%
  User              Hub            Status
> Amururineuruka    #test3         Downloading an other file.
  NotAyo            #test3         Downloading an other file.

-[10:06:18]—[ 1:main 2:#test3 3:queue 4:connections ]———————————————————————
                                                [U/D:    16/     1 KiB/s] [S:  1/ 10]
```

Figure 3.4: Screenshot of the download queue view in ncdc. The top pane displays the list of queued files, the bottom pane the list of users that have the selected file.

All clients have a configurable limit on the Degree of Parallelism (DoP). This is called the number of *download slots* in DC terminology, analogous to the upload slots discussed in section 3.4.

Of the three clients mentioned above, the peer selection mechanism in ncdc is the most primitive. Ncdc tries to establish a connection with all unique peers in the download queue. As soon as a connection has been established, the file with the highest priority available from that peer is requested for downloading. If that file is already being downloaded from another peer, the next file in the queue is requested instead. If there are no more files in the queue for that particular peer or if all download slots are in use, nothing is requested of the peer and the connection will be closed after a timeout.

An effect of this approach is that, since all peers are connected to simultaneously, file transfers are assigned to the peers that have a faster connection time and peers that have a slow connection time do not stall the peer selection process. However, the selected peers may not necessarily be the fastest peers in terms of bandwidth, and no attempts are made to try other peers once all download slots are in use. While multiple files can be downloaded simultaneously, ncdc does not employ PD — If there is only a single file in the download queue, then there will never be more than a single transfer at a time.

The peer selection mechanism in DC++ is roughly similar to ncdc, but has a number of improvements. In order to prevent a download slot from being "wasted" on a slow or inactive source peer, DC++ has an option to disconnect a peer if the received download speed is lower than a configurable threshold. This setting defaults to 1 kB/s.

DC++ implements a PD[6] scheme by requesting dynamically-sized file *segments* from each source peer. The first segment requested from a source peer is relatively small (say, 256 kB), but the segment size for subsequent requests is adjusted to approximate a download time of 2 minutes per segment. The calculation of the next segment size is based on the size of the previously downloaded segment and the time it took to download. Two subsequent segment sizes differ at most by a factor of two.

Jucy also implements PD, but takes a different approach to assigning segments to peers. Instead of requesting small chunks like DC++, Jucy requests the entire remainder of the file from the first peer with which a successful connection has been made. When a second connection is made with another peer for the same file, the remainder of the file is split in two evenly-sized segments, and the second segment is requested from the new peer. The peer that was previously assigned the full remainder of the file can continue to download until it reaches the start of the second segment. At that point Jucy has to interrupt the download by disconnecting the peer.

[6]Parallel downloading is often called *segmented downloading* or sometimes *multisource downloading* in DC terminology.

Chapter 4

Peer Selection in Direct Connect

This chapter looks back at the peer selection mechanisms discussed in chapter 2, and an evaluation of each mechanism is given based on a list of criteria. The goal of this chapter is to find both a peer selection mechanism and downloading algorithm suitable for use in Direct Connect.

This chapter is organized as folows. The criteria used for evaluation are described in section 4.1. Sections 4.2, 4.3, 4.4 and 4.5 offer evaluations of the peer selection mechanisms described in sections 2.1, 2.2, 2.3 and 2.4, respectively. A final comparison of all solutions is provided in section 4.6.

4.1 Criteria

The following criteria will be used to evaluate each peer selection mechanism.

Integration How well will the solution integrate into the existing Direct Connect network? That is, which and how many nodes need to be modified for this the solution to be effective? Ideally, a downloading peer that implements the new peer selection mechanism can benefit from it even if no other peers in the network have been modified.

Structure awareness A peer selection mechanism may make assumptions on the structure of the P2P network in which it may be used. The solution should fit within the structure of the Direct Connect network.

Parallel downloading The peer selection mechanism should use a PD scheme.

Dynamicity The list of source peers in DC is very dynamic — new peers are found and existing peers may leave while the file download is in progress. It is important that the peer selection process can handle this dynamicity.

Scalability The solution should scale well as the network grows. In particular, the number of connections with other peers, messages exchanged, or data to process should not increase significantly as the number peers in the network grows.

Signaling overhead The solution should not add too much overhead in the form of messaging.

Security The peer selection mechanism should provide acceptable service even in the face of incorrect and/or malicious peers.

Performance The ultimate goal of any solution should be to select the best peers suitable for downloading and to be able shorten the time it takes to download a file.

4.2 Topology Estimation

Topology estimation is a class of techniques that provide a ranking of source peers based on their (network) proximity. The existing techniques have been described in section 2.1.

Two of the criteria mentioned in section 4.1 can be evaluated for topology estimation techniques in general.

Structure awareness Topology estimation techniques, in general, optimize the case where source peers are distributed over the entire globe. In Direct Connect, many hubs focus on users in a particular geographical area, and even ISP-local hubs are not uncommon. Topology estimation techniques may not provide significant improvements with such hubs. Direct Connect is also used in LAN environments, but that case is not considered by any of the topology estimation techniques discussed below.

Parallel downloading Since topology estimation techniques merely provide a ranking of source peers and not a downloading algorithm, none of these solutions provide a PD scheme. A separate PD algorithm should be used when choosing a topology estimation technique.

The other criteria will be evaluated separately for each topology estimation technique in the following subsections.

Regarding the ‘Performance’ criteria, it should be noted that topology estimation techniques do not include a downloading algorithm, and it’s therefore not possible to say anything about their effects on the actual download time. Furthermore, the goal of topology estimation is to find peers within a close proximity, but such peers may not necessarily provide the highest bandwidth. What *can* be meaningfully said about the performance of topology estimation techniques is the accuracy of the proximity estimates. This is what will be discussed for the Performance criteria in the sections below.

4.2.1 Oracle-based

In an Oracle-based peer selection technique, each downloading client will query one or more centralised servers to obtain proximity information for all source peers, as described in section 2.1.1.

Integration No changes to the Direct Connect protocol are necessary at all for a client to rank peers, so any downloading peer capable of talking to an Oracle can benefit from the improved peer selection. This does, however, assume that the infrastructure that provides the Oracle already exists for a particular user. This infrastructure is independent of Direct Connect.

Dynamicity The downloading client can query the Oracle each time a new source peer appears to obtain its ranking within the list of source peers, so this solution has no problems with handling dynamic peer lists.

Scalability Each client only has to contact a fixed number of Oracle servers, and the size of the information to be exchanged is dependent only on the number of source peers. This does not hinder the scalability of the Direct Connect network.

Signaling overhead Since a downloading client needs to contact a fixed number of Oracle servers for every addition to the list of source peers, there is some extra signaling overhead. But only few Oracle servers — ideally only one — are needed to provide a complete ranking, so the signaling overhead is not expected to be very significant. There is no extra signaling necessary at all between peers within the Direct Connect network.

Security While other peers in the Direct Connect network should not be able to influence the Oracle-provided ranking, the assumption is made that the Oracle itself is trusted and secure.

Performance The accuracy of the ranking is solely dependent on the implementation of the Oracle. Since the Oracle is — ideally — operated by the network operators, it is assumed to be very accurate.

4.2.2 Synthetic coordinate systems

Synthetic coordinate systems, described in section 2.1.2, assign a (virtual) coordinate to each peer in the network, where network distances between peers can be obtained by calculating the distance between these coordinates.

Integration As synthetic coordinate systems are created and maintained among peers, every client in the Direct Connect network that may upload or download files needs to be modified implement coordinate management.

Dynamicity Synthetic coordinate systems have no issues with dynamic peer lists at all. A downloading client can request the coordinate of each source peer when it is needed and then rank the peers accordingly. Synthetic coordinate systems do, however, have trouble obtaining accuracy for peers that have only just joined the network.

Scalability In order to maintain its coordinate, a peer only needs to exchange coordinates and measure RTTs with a fixed number of other clients. In order to rank peers, a downloading client needs to obtain the coordinates only for the list of source peers.

Signaling overhead Exchanging coordinates between peers adds some signalling overhead, but this is expected to not be very significant due to the small size of the information needing to be exchanged (a coordinate) and the limited number of peers with which this exchange is necessary.

Security Synthetic coordinate systems are vulnerable to incorrect or malicious peers.

Performance The accuracy of a synthetic coordinate system on large-scale deployment may suffer due to problems with triangle inequality violation, churn, drift, malicious behaviour and latency variance. Finding good solution to these problems in an on-going research topic.

4.2.3 CDN-based proximity

CDN-based proximity, covered in section 2.1.3, uses the results of DNS queries from large-scale internet CDNs to determine whether two peers are close to each other.

Integration CDN-based proximity requires that each uploading peer maintains a ratio map and offers this map to downloading clients. This means that all uploading clients need to be modified for a downloading client to be able to rank its list of source peers.

Dynamicity When a downloading client discovers a new source peer, it only has to obtain its ratio map in order to update the ranking of source peers.

Scalability A downloading client has to obtain the ratio maps only for the list of source peers, and no other communication between the peers is necessary. Each peer also needs to regularly resolve a fixed number of DNS names, but this should not hinder scalability of the Direct Connect network and it is assumed that the DNS system is already capable of handling the additional load.

Signaling overhead A downloading client needs to request the ratio maps for all possible source peers. These maps are expected to be relatively small. The overhead of continuously querying a number of DNS servers is approximately 18 kB upstream and 36 kB downstream per day.

Security A malicious source peer can trivially provide an orthogonal ratio map that guarantees that it will be given a low ranking. If the source peer can also obtain the ratio map of the downloading peer — which is likely in a P2P scenario — then it can also provide a ratio map that guarantees a high ranking. This only affects the ranking of the malicious peer itself, it is not possible for a malicious peer to influence the ranking of other peers in the network.

Performance The solution only works in finding peers close to the downloading peer. If the distance between the downloading peer and all source peers is larger than the granularity of the CDN routing, then CDN-based proximity will not offer any information.

4.3 Performance Estimation

The performance estimation techniques provide both a solution for ranking source peers on some performance-based metric and include an algorithm for downloading the actual files. As such, each of the criteria listed in section 4.1 has to be evaluated for each mechanism, no general remarks can be made.

4.3.1 Local RTT measurements

The peer selection mechanism described in section 2.2.1 uses locally obtained RTT measurements to rank the list of source peers, and then assigns block sizes proportional to the measured RTTs for a number of top ranking peers.

Integration A downloading client needs to be able to obtain RTT measurements from its source peers, but this can be done without requiring changes to the Direct Connect protocol.

Structure awareness While the solutions was discussed in the context of FTP mirrors, it can be applied to the Direct Connect network as well.

Parallel downloading The solution includes an algorithm for non-interruptible parallel downloading.

Dynamicity While it is easy for the downloading client to obtain RTT measurements of newly found source peers, the used PD scheme does not take dynamicity of the peer list into account, and will require some modifications in order to handle this.

Scalability The downloading client only needs to contact each source peer once in order to obtain the RTT measurements, and can then proceed to download from a fixed number of peers. This solution does not add any scalability issues.

Signaling overhead The overhead from obtaining the RTT measurements should be relatively small, and these measurements are only obtained once. The parallel downloading algorithm assigns large file chunks to each source peer and attempts to minimize the number chunk requests.

Security The only information influencing the peer selection are the results of the RTT measurements. It is possible for a source peer to intentionally add a delay to get a low ranking, but the opposite is not possible.

Performance The solution was described and analyzed in the context of FTP mirrors, the accuracy of using RTT measurements to estimate bandwidth has not been studied in the context of P2P systems or with Direct Connect in particular.

4.3.2 Machine learning

The machine learning approach described in section 2.2.2 uses passively obtained information about peers to calculate a rough estimate of their expected performance. Using these estimations, the downloading peer will perform a sequence of partial downloads with different source peers to eventually settle on the peer with the best performance.

Integration Any downloading peer that implements this machine learning technique can immediately benefit from it, no other peers need to be modified. However, the efficiency of the peer selection may be improved when other peers are modified to provide more information. Hubs are also required to broadcast as much information as possible in the user list, some hubs don't do this.

Structure awareness The solution was designed for the Gnutella network, but the assumptions on the network structure hold for DC as well.

Parallel downloading The solution does not support a PD scheme.

Dynamicity Obtaining a performance estimate for newly found source peers can be done easily, but the downloading scheme provides no methods to handle dynamicity in the list of source peers.

Scalability The performance estimates for source peers are calculated from passively obtained information, and does thus not require any more messaging than already exists within the DC network. The downloading scheme connects to only a single source peer at a time and thus does not pose any scalability issues.

Signaling overhead The solution does not add any more signaling overhead than what is already present in the current DC network.

Security It is possible for a malicious peer to cheat its own rank in the performance estimation algorithm by broadcasting incorrect information about itself.

Performance The accuracy of the machine learning technique is dependent on the usefulness and interpretation of the passively collected information. This can vary depending on the network configuration of the client and which fields are distributed or filtered by the hub. The downloading algorithm attempts to find a single source peer to continue downloading from, but there is no guarantee that the chosen source peer will continue to provide the same bandwidth. It may happen that the download process gets stuck indefinitely if the performance of this peer degrades badly over time.

4.3.3 Random periodic switching

Random periodic switching has been described in section 2.2.3. The algorithm chooses a random peer from the list of source peers and starts downloading from it for a fixed amount of time. After a timeout, the algorithm starts from the beginning again and chooses a new random peer to download from.

Integration Random periodic switching does not rely on any specific protocol features, and can thus be applied directly on a downloading client without requiring other peers to be modified.

Structure awareness No assumptions are made on the structure of the P2P network.

Parallel downloading This solution makes use of the SD scheme, and it does not support PD.

Dynamicity The algorithm does not keep state between each downloading period and does not require extra knowledge about source peers, so it can adapt easily to a changing list of source peer.

Scalability No extra messaging is necessary and no extra state is kept.

Signaling overhead No extra messaging is necessary. Since the downloading algorithm only changes its single active source peer once in each time period, the number of download requests should not be very significant.

Security Since the peer selection process is random, other peers do not have a way to influence it.

Performance The solution removes the effects of network heterogeneity and temporal fluctuations in network capacity, and therefore ensures that the download will not get stuck on a slow source peer. However, the solution does not favour faster source peers over slower ones, and is therefore not optimal in reducing the downloading time.

4.3.4 Biased random periodic switching

Section 2.2.4 describes an improvement on random periodic switching that assigns a different connection probability to each possible source peer. The solution includes a PD scheme and the probabilities are normalized in such a way that their total sum is equivalent to the desired DoP.

Integration In order to obtain good probability assignments, the solution needs to be able to obtain the number of active uploads on each source peer. This information is already available in the DC protocol through various means such as the user list or the search results.

Structure awareness No new network assumptions are added compared to the random periodic switching technique.

Parallel downloading The solution supports a PD scheme, but no algorithm is given for assigning file chunks to source peers.

Dynamicity The downloading algorithm can accept and use new source peers as they are found, but no analysis has been made on how this affects the downloading performance.

Scalability No extra messaging is necessary that is not already part of the normal DC network.

Signaling overhead No extra messaging is necessary, and the downloading algorithm only connects to a limited number of source peers in each time period.

Security It is possible for a malicious peer to cheat its own connection probability in by providing incorrect information about its load, but the effects of such actions are limited by the randomness inherent in the solution.

Performance The periodicity and randomness of the algorithm prevents the download process from getting stuck on a single slow source peer, yet the algorithm favours faster source peers over slower ones in order to decrease the total download time.

4.3.5 Chunk-based probing

In chunk-based probing, described in 2.2.5, the file to be downloaded is divided into equally-sized chunks. These chunks are assigned to source peers in sequence in order to find a single high-capacity peer to download the remaining chunks from.

Integration No additional information on other peers is necessary, a downloading client can benefit from chunk-based probing without modifications to existing peers.

Structure awareness No assumptions are made on the structure of the P2P network.

Parallel downloading This solution makes use of the SD scheme, and it does not support the PD scheme.

Dynamicity The downloading algorithm assumes that peer lists are static. Extending this algorithm to give newly found source peers a try is possible, but has not been discussed or analyzed.

Scalability No extra messaging is necessary and only a single peer is used for downloading at any time.

Signaling overhead The number of chunk download requests depends on the chosen chunk size. The overhead of download requests will be insignificant with larger chunks.

Security The peer selection algorithm is dependent solely on information gathered by the downloading client and can not be influenced by other peers. A source peer can, however, influence its own ranking by providing different performance in the initial stage.

Performance The probing phase using fixed-size chunks may cause the download process to stall indefinitely if the chunk size is too large and there is a very slow source peer, or may decrease the accuracy of the measurements if the chunk size is too small. There is also no guarantee that the performance of the chosen source peer will not degrade badly over time.

4.4 Parallel Downloading Techniques

Parallel downloading techniques only provide a downloading algorithm, and do not make any selection among the list of source peers. As such, the following criteria can be evaluated on parallel downloading techniques in general:

Structure awareness Pure downloading techniques do not assume any structure in the P2P network beyond the basic concept of a downloading peer and a set of source peers.

Parallel downloading All techniques discussed in this section make use of the PD scheme.

Security Pure downloading techniques do not use any information outside of local performance measurements. It is therefore not possible for other peers to influence the downloading process by means other than changing their own provided performance.

4.4.1 Dynamic Parallel Access

Dynamic Parallel Access has been described in 2.3.1. The solution involves splitting the file to be downloaded into many equally sized chunks, and then assigning each chunk to a source peer. As soon as a chunk has been downloaded from one peer and there are still chunks to be downloaded, a new chunk is requested from the same peer again.

Integration The basic downloading algorithm can be implemented without requiring modifications in any other peers. However, as discussed in section 3.4, the pipelining technique to avoid the RTT delays between chunk requests can not be implemented without modifying all source peers to allow this.

Dynamicity The small chunk size allows for quick response to dynamic changes in the list of source peers. Remaining file chunks can be downloaded from a new source peer as soon as one is found.

Scalability The algorithm assumes that all available source peers are used to aid the download process simultaneously. When more downloading clients implement this scheme, the overall performance of the network will degrade significantly.

Signaling overhead Due to the small chunk size, there will be significant signalling overhead with the many chunk requests.

Performance The downloading time will increase significantly if the pipelining technique is not employed. Even if pipelining is used, the signaling overhead may be significant enough to negatively affect the downloading time.

4.4.2 Adaptive Dynamic Parallel Downloading

Section 2.3.2 discusses Adaptive Dynamic Parallel Downloading (adPD), a technique where the file to be downloaded file is split into as many equally-sized chunks as there are source peers. Each chunk is then downloaded from each source peer, and once one chunk has been downloaded the remaining chunks are divided proportionally to the measured bandwidth of each source peer.

Integration The solution assumes that active file downloads can be interrupted at any time and that a new chunk can be requested immediately. In DC, the only existing way to interrupt a download is to disconnect the source peer and re-open a new connection for the next chunk request. There is a much higher cost associated with this action than is assumed with adPD. A more efficient approach to interrupting downloads requires a modification the protocol and an update to all source peers.

Dynamicity The file is divided into as many chunks as there are source peers, so in order for the algorithm to handle newly found peers, the file has to be re-divided and chunks need to be re-assigned. The effect of this operation on the downloading performance has not been analyzed.

Scalability The solution assumes that a separate peer selection mechanism is used to select a limited number of source peers to use in the downloading process. With that assumption, adPD itself does not degrade the scalability of the network.

Signaling overhead The solution optimizes for minimizing the number of chunk requests, and thus adds very little signalling overhead.

Performance The cost of interrupting and immediately starting a new chunk request from the same source peer is not considered in the algorithm, but the algorithm provides good performance under the assumption that that cost can be minimized with protocol modifications.

4.4.3 Minimum-Signaling Maximum-Throughput

MSMT, discussed in 2.3.3, uses a Bayesian estimate as a indication of a source peers' bandwidth. The file to be downloaded is divided into chunks proportional to the estimated bandwidth. When a chunk has been downloaded, all existing transfers are interrupted, the Bayesian estimates are updated, and then each source peer is assigned new chunks.

Integration As with adPD (section 4.4.2), MSMT assumes that file transfers can be interrupted and resumed with a low cost.

Dynamicity The algorithm first downloads a small chunk from each source peer in order to get a starting estimate. If a new source peer is found while the download is already in progress, the new peer can't easily be used without starting the algorithm from the beginning. The effect of this operation on the downloading performance has not been analyzed.

Scalability Similar to adPD.

Signaling overhead The solution optimizes for minimizing the number of chunk requests, and thus adds very little signalling overhead. However, MSMT does not offer an algorithm for allocating chunks within the file, and may therefore require non-contiguous chunks to be assigned to source peers. This will increase the required signalling overhead.

Performance Similar to adPD, with the added note that the signalling overhead caused by assigning non-contiguous chunks to source peers has not been analyzed, but may cause the performance to degrade.

4.5 Incentive-Based Peer Selection

Incentive-based peer selection techniques have been chiefly introduced in section 2.4, followed by a short explanation of the BitTorrent protocol in section 2.4.1 as an example of such a technique.

Incentive-based techniques use peer selection to solve the problem of peers that try to benefit from the network without contributing back (*free-riders*). In the case of a file sharing network such as Direct Connect, this means that peers are expected to upload files to other peers if they wish to download something themselves. A peer selection technique can take this into account and either mandate or strongly encourage that only peers that upload files to the network will be able to use the network for downloading.

In the Direct Connect network, as described in section 3.1, the centralized Hub node is responsible for deciding and enforcing the policies on free-riding. Peer selection in the Direct Connect network, however, is not a centralized function. Each downloading client performs its peer selection independently of other peers. As such, when a client implements its own incentive-based peer selection technique, it may work against the policies set by the Hub administrator. Providing better mechanisms to grant the Hub more power in enforcing file sharing policies is outside of the scope of this research, and incentive-based techniques are therefore not considered in this comparison.

4.6 Comparison

A comparison of all previously discussed peer selection mechanisms is listed in table 4.1. Each solution is assigned a score from −− (bad) to ++ (good) for each of the criteria.

	Integration	Structure	PD	Dynamic	Scalable	Overhead	Security	Performance
Topology Estimation								
Oracle-based	++	−	N/A	++	++	+	++	++
Synthetic coordinates	−−	−	N/A	+	+	+	−	+/−
CDN-based proximity	−−	−	N/A	++	++	+	+	−
Performance Estimation								
Local RTT measurements	++	++	++	−	++	+	+	+
Machine learning	+	++	−−	−	++	+	+/−	−
Random periodic switching	++	++	−−	++	++	+	++	+/−
Biased RPS	+	++	+	+	++	+	+	++
Chunk-based probing	++	++	−−	−	++	+	+	−
Parallel Downloading								
Dynamic Parallel Access	+	++	++	++	−−	−−	++	+/−
adPD	+/−	++	++	−	++	+	++	+
MSMT	+/−	++	+	−	++	+/−	++	−

Table 4.1: Side-by-side comparison of the various peer selection techniques.

When choosing a solution for use with Direct Connect, it is important to keep in mind that a complete solution includes both a peer selection mechanism and a downloading algorithm. The topology estimation techniques only provide peer selection and the parallel downloading techniques only provide a downloading algorithm. These solutions are considered *incomplete*, and require a second solution from another category to complement it. The performance estimation techniques all include both peer selection and a downloading algorithm, so a single choice in that category will suffice.

It should also be noted that the criteria for incomplete solutions only evaluate what the solutions do provide. For example, the performance criteria for topology estimation techniques only evaluates their *accuracy*, and does not involve the notion of file download time. A highly accurate peer selection technique can not fully compensate for a badly performing downloading algorithm, so both solutions must perform well for the combination to provide good performance. Similarly, many of the criteria are cumulative in an unfavourable way. For example, if the peer selection technique requires extra signalling overhead, then this overhead will be present regardless of the choice of the downloading algorithm. Using multiple techniques to provide a complete solution is therefore only favourable when both solutions score well on all criteria.

Out of the discussed solutions, only Biased Random Periodic Switching has a + or better score on all criteria. It includes both peer selection and a downloading algorithm, and will therefore be the most suitable choice for the continuation of this research.

Chapter 5

Challenges

The peer selection mechanism most suitable for Direct Connect is, in chapter 3.5, chosen to be Biased Random Periodic Switching. Some challenges, however, still remain when applying this solution to Direct Connect.

5.1 File chunk allocation

As mentioned in chapter 2.2.4, the Biased Random Periodic Switching algorithm only determines which source peers should be downloaded from at which point in time. It does not provide a mechanism to allocate byte ranges within the file and it does not assign such chunks to source peers. To make matters more complicated, the algorithm is time-based rather than chunk-based, so it is not known in advance how many file data a peer ends up downloading from each source peer.

For Biased Random Periodic Switching to be applicable in practice, a file chunk allocation algorithm needs to be designed and analyzed. The primary goal of such an algorithm is to assign contiguous and non-overlapping chunks to each active source peer such that the number of file download requests is minimized, but there may be other considerations, too.

One such consideration is related to the verification of downloaded data. The TTH hashes used in Direct Connect allow the downloading peer to verify the correctness of individual fixed-size blocks within the file. If a downloaded block has failed the verification, then it would be beneficial to the downloading peer if the source peer that provided the incorrect data could be identified. In order to do this, the downloading peer may want to avoid assigning parts of the same fixed-size block to more than one source peer.

5.2 Source peer availability

Biased Random Periodic Switching works on the assumption that all source peers can be downloaded from at any time. This assumption does not hold in practice for Direct Connect. There are various reasons why a source peer can not be downloaded from at a certain point.

1. The source peer has left the network, either temporarily or permanently,

2. There are connectivity issues between the downloading peer and the source peer,
3. All upload slots of the source peer are taken.

The first reason is the only one that the downloading peer can reliably know in advance, since the list of all online peers is synchronised with the user list, described in chapter 3.1. The number of upload slots in use at each source peer may also be available in the same user list, but many hubs remove this information in order to save resources. Even if it is available, the slot count may be outdated by the time that a connection has been made with a source peer, and is therefore not a reliable indication of whether downloading from a source peer is possible. Combined with the possibility of connectivity issues, that means that the only reliable way for a downloading peer to know whether a source peer can be downloaded from is to actively try to create a connection and request a download.

The temporal unavailability of source peers causes the obtained average DoP to be lower than desirable in the Biased Random Periodic Switching algorithm, and may severely degrade the downloading performance as a result.

A special case of the above problem happens when the source peer implements a FIFO queue for slot assignment, as described in chapter 3.4. If the source peer has an upload slot reserved for the downloading peer, then the downloading peer should try to make use of that. Not doing so may end up in a situation where the source peer has all its upload slots allocated for downloading peers that aren't requesting a download within the allocated time frame, thus leaving valuable upload bandwidth unused.

A related problem is that a source peer may become unavailable while downloading from it.

Further research is needed to analyze the effects of source peer (in)availability and FIFO queue interaction on the downloading time, and to propose possible improvements to the Random Biased Periodic Switching algortihm to better handle realistic networks.

5.3 Source peer discovery

The problem is source peer discovery is as follows. Given a list of files to download, a peer needs to obtain a list of possible source peers to download from. Thorough this report, the assumption has been made that each downloading peer has already solved this problem. After all, one can only focus on the problem of peer selection *after* having obtained a list of possible source peers.

The mechanisms available in the Direct Connect protocols for source peer discovery have been described in chapter 3.3. But what has not been discussed is *how* these mechanisms should be used. Finding a suitable algorithm involves a number of trade-offs.

Accuracy The primary goal of a source discovery algorithm is to find as many source peers as possible. Preferably, it is able to find all source peers available in the network.

Performance A file download can not start before some source peers have been found, so if peer discovery takes a long time to complete, then this will severely affect total time taken to download a file. For small files or high-bandwidth networks, finding source peers can easily take more time than what is required to actually download the file.

Overhead and scalability The signalling overhead required to find source peers should not become significant when compared to the actual file download. Similarly, finding source peers should not severely degrade the overall performance and scalability of the network.

Searching for source peers may involve the flooding search mechanism provided by the Direct Connect protocols. This mechanism can provide quick and accurate results, but using it often may severely degrade the overall performance and scalability of the network. Alternatively, a downloading peer can obtain the file lists of other peers to see if they have some of the requested files, but the accumulated size of all file lists can easily exceed the total size of the files to be downloaded, wasting precious time and bandwidth.

Future research is necessary to determine whether the available mechanisms are sufficient, to analyze the trade-offs and to improve the algorithms currently used by Direct Connect clients.

5.4 Free-riding

This research focussed on using peer selection in Direct Connect to improve the downloading performance as seen by each downloading peer. The problem of free-riding has only been briefly discussed in chapters 2.4 and 3.1. Further research on this topic is out of the scope of this report and it was assumed that Direct Connect hubs already have suitable policies in place to prevent or strongly discourage free-riding.

In practice, the existing mechanisms that are available to the hub to prevent free-riding may be too limited. **Future research is necessary to analyze whether the existing mechanisms are sufficient, and to find possible improvements to combat free-riding.**

Chapter 6

Conclusions and Future work

This chapter finalizes this report with a short summary concluding the research, followed with a list of topics that may benefit from more future research.

6.1 Conclusion

This report has focussed on the problem of file downloading and peer selection in the context of the Direct Connect P2P filesharing network. We have found answers to the following research questions.

1. **Which peer selection mechanisms exist?**

   A variety of general peer selection mechanisms exist employing techniques to estimate network distances, estimate individual performance for each source peer, perform parallel downloading, or try to combat free-riding.

   Topology estimation techniques attempt to estimate the network distances between peers. Obtaining these estimates can be done either using a centralised Oracle server, in a more P2P fashion using synthetic coordinate systems, or by exploiting widely deployed CDN redirection mechanisms.

   Performance estimation techniques rank peers by obtaining an estimate of their performance. Existing solutions employ local RTT measurements, machine learning, (biased) random periodic switching or chunk-based probing.

   A few parallel downloading algorithms that do not provide peer selection has been described as well. These include Dynamic Parallel Access, Adaptive Dynamic Parallel Downloading and Minimum-Signaling Maximum-Throughput.

   Various incentive-based peer selection mechanisms exist to combat free-riding. These are out of the scope of this research and have not been discussed in detail.

2. **What is Direct Connect?**

   Direct Connect is a hybrid P2P filesharing network, consisting of three entities: Clients, Hubs and Hublists. A Hub is a TCP server that provides user authentication, user list synchronisation and general routing facilities for Clients. Each client may share its local files with other clients in the network.

   In order to perform file transfers, two clients can initiate a direct TCP connection between each other with the help of the hub. Using this TCP connection, a client can also request the complete file list shared by the other client. Clients can search for content and for alternative source peers by means of a flooding search coordinated by the hub.

   Uploading peers (clients that offer their files to the network) limit the number of simultaneous upload transfers using a *slots* mechanism. Existing Direct Connect clients implement a variety of peer selection and downloading strategies. Peer selection is commonly based on the time it took to connect to the peer. Most client implement a parallel downloading scheme.

3. **Which peer selection mechanisms are suitable for Direct Connect?**

   Important characteristics of peer selection mechanisms include how well it integrates with the existing Direct Connect network, whether it includes parallel downloading, how well it deals with dynamicity of the network, scalability and security, and how well it performs. Out of the discussed mechanisms, Biased Random Periodic Switching has been found to be the most suitable to Direct Connect.

The final question was **what challenges exist for peer selection in Direct Connect?** The answer is discussed in the next section.

6.2 Future work

The following list is a short summary of the challenges discussed in chapter 5.

- A file chunk allocation algorithm needs to be designed and analyzed.
- The effects of source peer (in)availability on the downloading time needs to be analyzed and further improvements to the Biased Random Periodic Switching algorithm need to be proposed.
- The source peer discovery mechanisms and algorithms in Direct Connect need to be analyzed and evaluated.
- Further research may be necessary to provide better mechanisms to allow Direct Connect hubs to combat free-riding.

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] JaCy - The Peer to Peer Search Engine. http://yacy.de/.

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

[7] Limelight Networks. http://www.limelight.com/.

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

[9] Spotify Music Streaming Service. http://www.spotify.com/.

[10] StrongDC++. http://dcplusplus.sourceforge.net/.

[11] The Freenet Project. https://freenetproject.org/.

[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] Brent N. Chun, David E. Culler, Timothy Roscoe, Andy C. Bavier, Larry L. Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *Computer Communication Review*, 33:3–12, 2003.

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

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

[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] Jianming Fu and Hui Fan. Assigning Block Size Based on Speculation for Parallel Downloading. In *Sixth International Conference on Computer and Information Technology*, page 119, 2006.

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

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

[29] 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.

[30] 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.

[31] J. Liang, R. Kumar, and K.W. Ross. Understanding KaZaA. *Submetido para publicaçao*, 2004.

[32] 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.

[33] Petar Maymounkov. Kademlia: A peer-to-peer information system based on the XOR metric. 2001.

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

[35] 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.

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

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

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

[39] 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.

[40] 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.

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

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

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

[44] 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.

[45] Rita Hanna Wouhaybi. *Algorithms for reliable peer-to-peer networks*. PhD thesis, Columbia University, 2006.

[46] 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.

[47] 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.

[48] Xu Zhou, Xianliang Lu, Mengshu Hou, and Chuan Zhan. A speed-based adaptive dynamic parallel downloading technique. *Operating Systems Review*, 39(1):63–69, 2005.