UNIVERSITY OF GOTHENBURG

Measurement and Analysis of the Direct Connect Peer-to-Peer File Sharing Network

Master of Science Thesis

KARL MOLIN

University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2009

Measurement and Analysis of the Direct Connect Peer-to-Peer File Sharing Network

Karl Molin

Examiner: Marina Papatriantafilou

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Printed at the Department of Computer Science and Engineering Chalmers University of Technology and University of Gothenburg

Department of Computer Science and Engineering
Göteborg, Sweden June 2009

Abstract

Online social networks and peer-to-peer file sharing networks create a digital mirror of human society, providing insights in social dynamics such as interaction between entities, structural patterns and flow of information. In the past such studies were inherently limited due to the vast supply of information. Today these phenomena can be studied at large scale using computers to process data from this digital mirror.

Findings from such networks have shown interesting structural properties shared by both types of systems. In particular, it is often the case that they show to be *scale-free* and *small-world networks*.

By letting ideas and findings from studied peer-to-peer networks guide the design of novel architectures, improvements on user integrity, usability and performance have been observed.

This thesis presents a study of the Direct Connect peer-to-peer file sharing network. We model abstract tools and methods for measuring the network architecture, and, moreover, custom software tools for data gathering and analysis from Direct Connect networks are developed, presented and discussed.

We look at network topology and properties, statistics on user activities and geographic distribution, characterization/statistics on data shared and correlations of users and their shared data.

We verify the scale-free property, small-world network model, strong data redundancy with clusters of common interest in the set of shared content, high degree of asymmetry of connections and more.

Finally, we discuss the implications of our findings and comparison with results from similar research is done.

Sammanfattning

Online-baserade sociala nätverk och så kallade peer-to-peer-fildelningsnätverk skapar en digital spegelbild av det mänskliga samhället. Denna spegelbild kan ge insikt i social dynamik såsom interaktion mellan parter, strukturella mönster och informationsflöde. Förr var sådana studier inneboende svåra på grund av det stora informationsflöde som detta är förknippat med. Idag kan dessa fenomen studeras storskaligt genom bruk av datormaskiner.

Dessa nätverk har uppvisat intressanta strukturella egenskaper som delas av båda nätverkstyper. I synnerhet är ofta fallet så att de uppvisar egenskaper av att vara så kallade *scale-free*- och *small-world*-nätverk.

Genom att låta upptäckter från undersökta peer-to-peer-nätverk styra utformande av nya arkitekturer har förbättring av bland annat användarintegritet, användbarhet och prestanda observerats.

Denna tes presenterar en studie av peer-to-peer-fildelningsnätverket Direct Connect. Vi tar fram abstrakta verktyg och metoder för mätning av nätverksarkitekturen, och vidare, utvecklar mjukvara för informationsinhämtning och analys.

Vi tittar på nätverkstopologier och egenskaper, statistik över användaraktiviteter och geografisk distribution, karaktäriserar den data som delas och hur användare och deras data korrelerar.

Vi verifierar att Direct Connect är ett scale-free-nätverk, small-world-modellen, stark dataredundans med kluster av liknande filer i mängden av utdelad data, en hög grad av asymmetri mellan användare och mer.

Slutligen diskuteras vad våra upptäckter betyder och jämförelser med liknande forskning utförs.

Acknowledgements

This master's thesis work was conducted at the Department of Computer Science and Engineering at Chalmers University of Technology and Gothenburg University under supervision by associate professor Marina Papatriantafilou and Giorgos Georgiadis during spring 2009.

I am grateful to Marina and Giorgos for opening my eyes to an exciting research topic, providing inspiration and giving me valuable suggestions throughout my work.

Furthermore, a thank-you goes to Eva-Lotta Mattsson for giving me helpful comments along the way of my work.

Finally, my tokens of gratitude go to Bram Moolenaar for his most excellent Vim text editor and to Leslie Lamport for designing the typesetting macro package LaTeX – without these, this thesis would not have existed in its current form.

Contents

1 Introduction **1**
- 1.1 Related work . . . 4
- 1.2 Outline of thesis . . . 5

2 Background **6**
- 2.1 Peer-to-peer networks . . . 6
  - 2.1.1 Topologies of peer-to-peer networks . . . 7
- 2.2 Direct Connect . . . 9
  - 2.2.1 Architecture . . . 9
  - 2.2.2 Protocol . . . 11
- 2.3 Network graph modelling . . . 14

3 How to measure the Direct Connect network **17**
- 3.1 Peers . . . 17
  - 3.1.1 Activities . . . 19
  - 3.1.2 Relations . . . 19
- 3.2 Shared data . . . 22
  - 3.2.1 Correlation between peers and shared data . . . 22

4 System design **24**
- 4.1 DCSpy . . . 24
  - 4.1.1 Libraries and tools used . . . 26
- 4.2 The ShareStat suite . . . 27

5 Results and data analysis **28**
- 5.1 Geographic distribution . . . 30
- 5.2 Peer activities . . . 30
- 5.3 Peer graph analysis . . . 31
- 5.4 Shared data graph analysis . . . 36

6 Discussion **40**
6.1 Geographic distribution 40
6.2 Activities 41
6.3 Peer connectivity 41
6.4 Content and Peer–data correlation 42

7 Summary **44**
7.1 Future work 45

A Software manual: DCSpy **49**

B Software manual: The ShareStat suite **52**
B.0.1 ShareStat 52
B.0.2 analyze.rb 53

C Direct Connect graph **54**

Chapter 1

Introduction

Over the last few years, peer-to-peer (abbreviated P2P) technology has gained an increasing attention among Internet users, companies and academic researchers.

The Napster phenomenon in early 2000s was, by gaining the whole world's attention, probably the one piece of software to really ignite today's trend of P2P networking. By letting anyone from around the globe in an easy way share music files with each other, Napster quickly became popular among young music enthusiasts. Meanwhile, the music industry was panicking and Napster was eventually forced to shut down. But the story far from ended there; after having experienced the power and possibilities of P2P file sharing technology, people refused to take the demise of Napster as the final word. Various new networks and clients for file sharing began to pop up and the trend seems to maintain its popularity when looking at all the alternatives existing today.

Even though P2P often seems to be mentioned in file sharing contexts, a multitude of other uses of P2P exist. Examples of common P2P-based uses and applications are:

Instant messaging An early player in this niche was the instant messaging software homophonically dubbed ICQ. Today, instant messaging is used on a daily basis by people from all over the world; notably by using the popular Messenger software included in Microsoft Windows.

Collaborative environments Being able to share a desktop among users for collaboration is a common option in modern operating systems.

Games In multiplayer action, strategy and role-playing games P2P is a technology often used letting players communicate and discuss strategies.

Internet telephony and video conferencing Within this area Skype is probably the most well-known actor. Skype addresses both voice-over-IP telephony as well as video conferencing.

Music streaming In autumn 2008 music streaming service Spotify was much hyped and gained a lot of attention from both media and public.

Distributed computing SETI@home is a project where users from all over the world let their computers collaborate to find extraterrestrial life. Their computers' idle clock cycles are used in a distributed fashion to analyze data received from a huge radio telescope listening to outer space.

Today it is estimated that the traffic generated by P2P protocols stands for a major portion of the total traffic on the Internet. This hints on the sizes of these distributed systems. Not only have these networks revealed to be huge in size, by nature their structure is highly dynamic – a fact that does not seem to affect their performance or stability.

Recently, quite a substantial amount of research has been made on online social networks as well as on P2P file sharing networks. These networks create a digital mirror of human society, providing insights in social dynamics such as interaction between entities, structural patterns and flow of information. In the past such studies were inherently limited due to the vast supply of information. Today these phenomena can be studied at large scale using computers to process data from this digital mirror. Results obtained cannot only be used by computer scientists, but is of value for any discipline dealing with human behaviour and its patterns. Interestingly, this creates links between sciences usually far from each other [4].

Online social networks can be viewed as a special case of P2P systems and are extremely popular on the Internet today. In contrast to traditional P2P systems which connects users via IP addresses, these systems connects users using web links. Examples of such networks is Facebook, MySpace and Flickr. The main motivation for online social networks is to maintain social relationships, create new relationships based on similar interests and locate information contributed by other users of the system. Common for these networks is that it is often the case that an object is required to connect two people. Such an object can for example be photos, movies, hobbies or jobs. Worth noticing is that in these kind of systems it is most often the case that the users themselves produce and contribute all data in the network. This stands in contrast to the other common paradigm where a service on the web is also the content provider.

The other, more traditional, variant of P2P systems which are intended for file sharing are the focus of this thesis. Although sharing attributes with online social networks, some key differences exist. In file sharing networks it is often the case that the data exchanged is not owned or produced by the participants. The type of object that connects two people in a file sharing network can be arbitrary and is merely based on personal preferences. When participants interact with each other by exchanging a file, the link created is (in contrast to social networks) often arbitrary; a user wishing to download a file simply make a search from the set of currently available peers.

Most current widely used P2P systems rarely facilitate or exploit social relationships between users. Recent research have shown promising results employing such relations in novel P2P systems. By letting ideas and findings from online social networks guide the design, improvements on user integrity, usability and performance have been verified [8] [3]. Examples of this is when querying for data; by employing information such as user taste or social relations, location and downloading of data content can be improved. This stands in bright contrast to the common earlier mentioned paradigm where arbitrary links are created. Also, user privacy can be strengthened by letting social relations between users decide on with whom to share data content. This scheme effectively disables a third party to monitor activities between users forming a social cluster.

Findings from online social networks and P2P file sharing networks have shown interesting structural properties shared by both types of systems. In particular, it is often the case that they show to be *scale-free* and *small-world networks*[1].

Whereas a quite wide range of diverse online social networks have been investigated, much of the research community's focus regarding file sharing has been on Gnutella, Kazaa and recently BitTorrent.

Direct Connect is a huge set of social P2P file sharing networks which almost seem to have completely escaped attention from researchers. The architecture has – besides file sharing – support for mechanisms such as chat, granting friends extra capabilities of downloading and formation of networks specialized in certain types of content. Virtually no investigation on structural properties, content shared and comparison to other P2P networks have previously been done on Direct Connect networks. This, in conjunction with the system's mix of file sharing and social capabilities as well as the maturity of the technology makes the system interesting for analysis.

This thesis presents a study of components in Direct Connect such as: network topology and properties, statistics on user activities and charac-

[1]The terms scale-free and small-world networks are discussed later in this thesis.

terization/statistics on data shared. We investigate if the Direct Connect network exhibits properties of being a scale-free network and look on correlations of users and their shared data. Also, custom tools for data gathering and analysis from Direct Connect networks are developed, presented and discussed. Furthermore, findings are compared to findings from other P2P and real world networks.

1.1 Related work

There is quite an amount of work on P2P networks going on in the research community. In this section we briefly present related research projects that have inspired this thesis.

An interesting study on structural properties of P2P networks has been conducted by Wang et al. [11]. A statistical analysis of the Gnutella network is performed and presented. It is shown that the network is mainly comprised of peers functioning as pure providers. These guarantee high availability and reliability of the system. The connectivity between peers is revealed to be scale-free – both for directed and undirected links.

Another case study of the Gnutella network has been performed by Ripeanu [9]. A crawler was used to map the overlay network, the generated graph was analyzed and network traffic evaluated. Using the findings Ripeanu suggests changes to the Gnutella protocol to improve both performance and scalability.

Mislove et al. [6] has performed an extensive study on the structure of online social network graphs. Communities looked at are for example Orkut, YouTube and Flickr. Data gathered by crawling links from the sites in question was used to model a graph structure. Several properties were shown. In particular, it was shown that these online communities seem to follow the power law distribution (i.e. being scale-free networks)

Although not directly concerning structural properties, TRIBLER and OneSwarm discussed below is of interest for being new P2P file sharing systems based on social relations.

Research has been performed by Pouwelse et al. [8], constructing TRIBLER, a social-based P2P system with focus on sharing of videos. TRIBLER is built upon the BitTorrent protocol and gives possibility to add friends to a list of contacts. These friends are assumed to share your taste and the sharing of content in the network is optimized using the relations formed.

Another BitTorrent-based file sharing network is OneSwarm [3]. OneSwarm aims to preserve user integrity in a P2P network by introducing the concept of "friend-to-friend" file sharing (F2F). The protocol lets users control of

with whom to share data, either by directly adding people as friends or by looking up your contacts on the Google Talk network. This, and the use of encryption of both IP addresses and data makes the traffic hard to monitor for a third party. To guide the design of OneSwarm, a study of the structural properties of the online social network last.fm is performed.

1.2 Outline of thesis

The aim of this thesis is that any person with a background in computer science and mathematics should be able to read and understand it. The reader does not necessarily have to be familiar with all concepts this thesis will discuss; they will be explained when they appear.

The thesis is divided in seven chapters. Chapter one introduces the subject and gives a motivation for studying P2P networks. Also, research which have inspired this thesis is briefly looked upon.

Chapter two gives the reader a background to P2P networks in general, Direct Connect in particular and mathematical graph theory needed. This chapter provides the necessary knowledge and abstract tools for measurement and analysis of the Direct Connect network.

The third chapter concerns the method that will be used to measure the Direct Connect network. The discussion in this chapter is based on the topics from the previous chapter.

In chapter four, we discuss the actual implementation for collecting data. External tools and libraries used are also looked upon. Since this thesis does not concern software architecture, design patterns etc., this discussion is held at quite a minimum. The enthusiastic reader is encouraged to contact the author of this thesis if interested in obtaining source codes.

The fifth chapter presents the data collected. Tables and figures are used to illustrate topological properties of the networks.

Chapter six gives a discussion on the findings from the previous chapter. The discussion tries to catch good characteristics of Direct Connect networks in order to compare these to results from earlier research on P2P networks.

In chapter seven, we present a summary with conclusions of our work. We look back and see what was good, bad and how future research on the subject can be improved.

The terms *node*, *vertex* and *peer* have all very similar semantics and are used differently throughout the text. The same goes for the terms *link*, *edge* and *connection*. This is not to confuse; rather to emphasize exactly what structure we are discussing. It is the author's aim to be consequent with the terminology based on the context.

Chapter 2

Background

2.1 Peer-to-peer networks

A P2P computer network is a system in which the connectivity of the participants does not rely on one (or some) central resource(s). In a P2P network the traditional server-client hierarchy is flattened by removing these central resources and letting the participants connect directly to each other – in practice making every computer both a server and a client. The computers participating in a P2P network are often referred to as peers or nodes.

It is easy to realize that this architecture implies some often desired properties of a network, such as good scalability, removal of points of failure and elimination of "middlemen". As all nodes provide their own resources (e.g. bandwidth and storage) the P2P network in fact becomes stronger as more nodes connect to it. The properties mentioned make a P2P network an ideal choice when a large number of participants are involved in heavy data traffic such as file sharing, streaming of audio/video or distributed computations.

Miller [5] presents an easy and concise definition of P2P:

> "P2P is a network architecture in which each computer has equivalent capabilities and responsibilities."

Although this definition is elegant and correct, it may not always be accurate for describing a P2P network; the definition should capture the set of all P2P networks but actually covers more.

Miller extends the definition in order to make it more precise. He suggests that a P2P network should show these five characteristics:

- The network should facilitate real-time transmission of data between the peers.
- The peers can operate as both server and client.
- The main content of the network is provided by the peers.
- The network gives control and autonomy to the peers.
- The peers are not necessarily always connected or have permanent IP addresses.

This definition is a great deal more robust and satisfactorily describe a P2P architecture.

A P2P network which totally lacks centralized resources is sometimes said to be a *pure* P2P network. This is the most strict type of a P2P system where the network is comprised of equal peers acting as both servers and clients. By the definition there exists no centralized resource keeping track of, for example, peers or their respective content.

Conversely, there are so-called *centralized* P2P networks in which one or several servers exists handling tasks like for example locating content and which peers offer it.

2.1.1 Topologies of peer-to-peer networks

An overlay network can be viewed as a graph representing a virtual network consisting of nodes and connections between them. The nodes represent participants and the connections represent virtual links between them. An overlay network is built on top of an existing network. The reason for such a construction can be to implement a service the underlying network does not directly support.

In the case of most P2P protocols, the underlying network is the Internet and an often wanted goal is to efficiently locate information among the nodes.

As a P2P system is ideal for ad-hoc connections, the topology of its overlay network is subject to frequent change. A P2P network not keeping track of connections between nodes in an organized manner is said to be *unstructured*.

Although the network's overlay graph may be unstructured, it may indeed implement techniques for organizing information among nodes. For example, queries in the network can be routed in an intelligent manner based on previous knowledge about neighbouring nodes. Another approach is to let a random or statistically based "walker" traverse the network with a query.

Freenet is an example of an unstructured network enforcing a protocol for organization of data [2].

An unstructured network not implementing information organization is often not desirable in the case of systems with a large number of nodes where good performance and scalability is of importance.

In particular, such a network may have to reside to flooding a query through the network to locate content. This method of locating information scales badly due to the high amount of traffic generated by the usage of a broadcast-mechanism. Another problem is that a query is likely to quickly find data that many peers share, but finding rare data can be very time-consuming and might not even resolve at all. Examples of applications using this unstructured approach is FrostWire and LimeWire who both use the Gnutella file-sharing network.

When a protocol organising the overlay links is enforced, the P2P system is said to be *structured*. The goal by employing a structured approach – by structuring information or overlay network – is to minimize the number of hops in the overlay graph when locating information.

For applications with high requirements such as distributed systems in cars, business applications etc. a structured protocol is often preferable.

A good example of a structured P2P protocol is the Chord protocol suggested by Stoica et al [10].

Chord ensures that given a key, it will perform a lookup on the node storing the key's value in $O(\log n)$ time. Briefly, this is achieved by:

- Assigning a unique identifier to each node and key using a (consistent) hash function such as SHA-1.
- Creating a logical ring where the key and node identifiers are placed in a clockwise increasingly order.
- Each node holds a lookup table to a subset of its succeeding nodes. This routing table is organized such that a node "knows" more numbers of nearby located nodes than nodes located far away.
- A search query is done by looking up the key's highest predecessor using the routing table, then forwarding the query to that node. This goes on until the node who owns the key is found. Due to the fact that for each query forwarding, the logical distance to the node owning the key is at least halved, the search is performed in $O(\log n)$ time.

It can be shown that the $O(\log n)$ property holds for any n nodes participating in the network, thus Chord proves to scale well.

2.2 Direct Connect

Direct Connect (from now on referred to as DC) is a semi-centralized unstructured file sharing protocol originating from the late 1990s. It was invented by the by-then nineteen-year-old hobbyist programmer Jon Hess [7]. The protocol is to this date officially undocumented and the existing documentation is derived from reverse engineering of the NeoModus client – Hess' original implementation. Thus the DC protocol is sometimes referred to as the NMDC protocol[1].

The DC protocol draws inspiration from the popular IRC[2] chat protocol offering the same basic possibilities such as public and private messaging.

2.2.1 Architecture

The architecture of a DC network is simple and is comprised of a server – or a *hub* in DC-terminology – and peers who are clients to the hub. Although the peers act as clients to the hub, a peer act as both server and client to other peers as in conventional P2P fashion.

Below we will have a closer look at the details and functionality of the above discussed entities.

Hubs and peers

The hub listens to incoming TCP connections on – by convention – port 411. As its name suggests, the hub's function is just to act like a hub facilitating connections between peers. It is also responsible for bookkeeping of connected clients, forwarding of search queries and delivering of both public and private messages. Public messages are broadcast to the connected clients, while private messages only are sent to the intended recipient. When a client queries the network for a file, the query is sent to the hub and the hub broadcasts it to all connected clients. Clients in possession of the requested file answers directly to the client who made the query[3]. From here a direct P2P file transfer between the two parties can take place.

A peer connects to a hub with a chosen user name (also known as *nickname*). This user name has to be unique for the hub to which the user connects. A user can have the status of normal user or operator. A user granted with operator status is more privileged and possess permission to actions like disconnecting users and banning users from the hub. The set of

[1] NeoModus Direct Connect protocol.

[2] Internet Relay Chat.

[3] In the ideal case – we will see a case when this does not apply.

all peers connected to a hub forms a potential network in which files are being shared. A connected peer can chat, search for files and upload/download content to/from other connected peers. Important for peers are the concepts of *slots* and *active* versus *passive* mode.

Each peer has a number of slots which determines how many peers it simultaneously can upload data to. It is common for client software to feature a function to grant extra slots to bookmarked friends if all slots are occupied. These slots are sometimes called *granted slots*. Another type of slots existing are the so-called *mini slots*. If there exist no other free slots, a mini slot can be granted to a peer wishing to download and view another peer's list of shared files[4].

It is not uncommon that hubs are part of a group or network of hubs. Although not directly connected to each other, they can somewhat cooperate with each other. An example of this is when a hub's user limit is met; in such case the hub can automatically redirect the user to another hub in the group of hubs.

Active vs. passive mode

Active mode is the standard for peers to use. This mode requires the peer to have a dedicated port open for both incoming and outgoing traffic. In other words, this enables peers to connect directly to each other without problems. Active clients performing a search get their results directly from the peers answering the query.

Passive mode is not recommended but (somewhat) works under all circumstances. In this mode there is no need to have a port open for incoming traffic. Results from a query is routed via the hub to the searching client, consuming bandwidth and processing power from the hub. Because of this, many systems will answer a passive client with only five results for each responding user. Another drawback using passive mode is that a passive client for natural reasons cannot connect to another passive client.

Here, the observant reader might wonder how a connection is established between a passive peer and an active peer if a passive party requests a connection from an active party. Since a passive peer cannot function as a server in a P2P connection, a workaround exists. This is done by the passive peer sending a request to the other party asking for it to act as a server accepting a connection. Now the passive and active peers can establish a connection and data can be exchanged even if the passive peer took the initiative of the transfer. This is often referred to as a *reverse connection* operation.

[4]Every peer store an XML-file containing information about its own files being shared.

Needless to say, active mode is the preferred mode since it results in better performance for both the peer and the hub.

2.2.2 Protocol

The original NMDC protocol has undergone quite an amount of extensions and improvements since the original implementation. Most augments to the protocol has been introduced by Jacek Sieka in his client software DC++[5]. This client runs under Microsoft Windows and has for many years been *de facto* standard DC client.

Popular hub server software is YnHub[6] (for Microsoft Windows) and Verlihub[7] (for GNU/Linux).

The protocol operates as clear text messages which are sent between peers and the hub or peers and peers. Each message begins with a `$` and ends with a `|`. It is beyond the scope of this thesis to describe the protocol in full, so only a subset of it will be discussed. This enables the reader to achieve a feeling of how it works and prepare for upcoming discussions in this thesis.

Handshake phase

When the client has established a TCP socket connection to the server (hub), the server immediately responds with

```
$Lock <lock> Pk=<pk>|
$HubName <hubname>|
```

Where `<lock>` is a number used to generate a key, `<pk>` is unused and `<hubname>` is the name of the hub that is being connected to. The procedure of calculating a key from a lock is quite easy. First every key character (except for the first) is calculated using the given lock:

```
for (i = 1; i < lock.length; i++)
    key[i] = lock[i] xor lock[i - 1];
```

Next, we calculate the first key character using the first and the two last characters of the lock:

```
key[0] = lock[0] xor lock[lock.length - 1] xor lock[lock.length - 2] xor 5
```

[5] http://dcplusplus.sourceforge.net/
[6] http://www.ynhub.org/
[7] http://www.verlihub-project.org/

Further, we must swap every nibble (4 bits) for every character in the key:

```
for (i = 0; i < lock.length; i++)
    key[i] = ((key[i]<<4) & 240) | ((key[i]>>4) & 15)
```

Finally, the ASCII characters 0, 5, 36, 96, 124 and 126 are forbidden to send to the server and are respectively substituted with the string `/%DCN000%/`, `/%DCN005%/`, `/%DCN036%/`, `/%DCN096%/`, `/%DCN124%/` and `/%DCN126%/`. The client then responds with

```
$Key <key>|
$ValidateNick <nick>|
```

Where `<nick>` is the desired user name. If this name is unavailable on the hub, the server will respond with

```
$ValidateDenide|
```

If the hub requires a password, the sever sends

```
$GetPass|
```

To this the client replies

```
$MyPass <password>|
```

Where `<password>` is a plain text string. If the password is correct and the connecting user has operator status on the hub, the server responds

```
$LogedIn|
```

In the case the password was incorrect, the server sends

```
$BadPass|
```

Followed by closing of the connection. If the above procedure went well, the handshake phase is finalized by the server sending

```
$Hello <nick>|
```

Where `<nick>` is the user's nickname of choice.

Chat and messaging

Messages sent to the public chat differs slightly from other messages of the protocol; these messages are not prefixed with a `$`. A public message from `<nick>` is sent as

```
<<nick>><message>|
```

The hub broadcasts this whole message unmodified to the connected clients.

For private messaging, a client sends to the hub

```
$To: <othernick> From: <nick>$<<nick>><message>|
```

The hub forwards this whole message unmodified to the recipient.

Search procedure

The way a search is performed differs depending on whether the searcher is in active or passive mode. We begin by looking at the way an active search is done. A peer in active mode sends

```
$Search <clientip>:<clientport> <searchpattern>|
```

Where `<searchpattern>` is of the form

```
<sizerestricted>?<isminimumsize>?<size>?<datatype>?<searchterms>
```

The value of `<sizerestricted>` equals `T` if the search should be restricted to files of a minimum or maximum size. Otherwise this is `F`. If `<sizerestricted>` is `T`, the following applies for `<isminimumsize>`: `T` if the search should match files of a minimum size, or `F` for matching on a maximum size. The actual size for restrictions is found in `<size>`. The value of `<datatype>` is a number between 1 and 9 representing file type categories such as audio, archives, pictures, videos etc. The string `<searchterms>` contains search tokens delimited by `$`. A file name containing all these tokens is considered a match.

When the hub receives a search message, it broadcasts it unmodified to the connected peers.

A client operating in passive mode who performs a search sends a similar message to the hub:

```
$Search Hub:<nick> <searchpattern>|
```

As in active searches the hub broadcasts this message unmodified to the peers connected. A peer responding to an active search responds directly with a message to IP `<clientip>` on port `<clientport>` using a UDP packet. This message looks like:

```
$SR <replynick> <response> <hubip>:<hubport>
```

The string `<response>` most notably holds name, size and hash of the matching file. Moreover, it also holds information on the number of available slots the responder provides. The hash of a file is calculated using Tiger Tree Hash (TTH) which, shortly put, is the Tiger hash algorithm used in a binary tree structure.

If a peer responds to a passive search the result is routed via the hub. The responding peer sends

```
$SR <replynick> <response> <hubip>:<hubport> <searchnick>|
```

The hub forwards this message unmodified to the recipient.

This concludes the DC protocol overview. The reader should now be familiar with basic workings and concepts of DC and have a good general understanding of the protocol.

2.3 Network graph modelling

A mathematical graph is a convenient abstract structure in which to model a network, and the DC network makes no exception to this. By using a graph model we can view, measure and reason about a network in a concise manner. The reader of this thesis is most certainly already familiar with the concepts of graphs but to avoid any confusion we will now give and discuss some definitions. We begin by giving two basic definitions:

Definition 1. A *graph* $G = (V, E)$ is a set of vertices V and a set of edges E. An edge e_{ij} connects vertex v_i with v_j, where $e_{ij} \in E$ and $v_i, v_j \in V$.

Definition 2. A *directed graph* $G = (V, E)$ is a set of vertices V and a set of ordered pairs E. An edge $e = (v_i, v_j)$ is considered to be directed from v_i to v_j, where $e \in E$ and $v_i, v_j \in V$.

In a directed graph, an edge is sometimes called an arc. To avoid cluttering of terms we will simply call it an edge and let the context decide whether we are discussing a directed or undirected graph.

A third class of graphs will also be subject of discussion in this thesis; the *weighted graph* – or edge-weighted graph. A weighted graph is a directed or undirected graph with a weight w assigned to each edge. The weight can be specified to positive numbers, natural numbers, integers etc., but in this thesis we restrict w such that $w \in \mathbb{Z}$.

It is often the case that it is of interest to talk about a vertex's *neighbours* or the *degree* of a vertex. The two definitions below straightens this out:

Definition 3. A *neighbourhood* N_i for a vertex v_i is defined as the set of its immediately connected neighbours:

$$N_i = \{v_j : e_{ij} \in E \vee e_{ji} \in E\}$$

Definition 4. The *degree* k_i for a vertex v_i is defined as the number of vertices $|N_i|$ in its neighbourhood.

For an undirected graph, it is sufficient to just talk about the degree of a vertex, but things change if we deal with a directed graph. As we have seen, a directed graph handles edges as ordered pairs of vertices (v_i, v_j) which gives a distinction of ingoing and outgoing edges to/from vertices. As when discussing weighted graphs, we will not bother making another definition but settle with a less formal definition of the concept of in-degree and out-degree of a vertex. As the name suggests, the in-degree of a vertex is the number of incoming edges to a vertex in a directed graph. Conversely, the out-degree is the number of outgoing edges from a vertex in a directed graph.

An important measure on graphs is the *clustering coefficient*. This enables us to quantify over how near a vertex and its neighbours is to form a complete graph[8].

We use the method of measurement introduced by Watts-Strogatz [12]:

Definition 5. The *clustering coefficient* C_i for a vertex v_i is expressed as:

$$C_i = \frac{2|\{e_{jk}\}|}{k_i(k_i - 1)} : v_j, v_k \in N_i, e_{jk} \in E$$

For measurement of how well connected a graph is, we use our previous definition and extend it with:

Definition 6. The *clustering coefficient* $\langle C \rangle$ for a graph is defined as the average of all clustering coefficients C_i in the graph:

$$\langle C \rangle = \frac{1}{n} \sum_{i=1}^{n} C_i : n = |V|$$

Based on these definitions the range of the clustering coefficient is $[0, 1]$. For a fully connected network we have $\langle C \rangle = 1$ and for a random network we have $\langle C \rangle = \langle k \rangle / n$, where $\langle k \rangle$ is the average vertex degree and n is the total number of vertices in the graph.

[8] A complete graph is a set of vertices where every pair of vertices is connected by an edge.

To illustrate this measure we look at results obtained by Watts-Strogatz [12]. A graph modelling a relation between 225226 actors ($n = 225226$) was constructed. The relation was defined as follows: two actors are joined by an edge if they have acted in a film together. The average degree $\langle k \rangle = 61$. The clustering coefficient obtained for the graph was 0.79. When constructing a random network using the same n and $\langle k \rangle$ the resulting clustering coefficient is 0.00027. These values give us an idea of clustering coefficients for highly connected versus very sparse connected graphs.

Graphs showing a high clustering coefficient are often called small-world networks. The name originates from the "small-world phenomenon" which says that it is often the case that two strangers often know each other by a mutual friend. Studies have showed that in fact many real world networks show characteristics of being small-world networks, i.e. having a clustering coefficient higher than what to expect from a random network.

The related term – scale-free network – considers a network's degree distribution. A network is said to be scale-free if the degree k of a given node adhere to a power law distribution such that $P(k) \sim k^{-\gamma}$ where γ often is $2 < \gamma < 3$. Other real world networks have been shown to be scale-free [11][6].

Networks showing these properties are known for being robust and resistant to random node-failures. By deleting a random node it is unlikely that a high-degree node is deleted; therefore such a node failure rarely causes great damage to the flow of information in the network.

Another measure of interest is the *average path length* which quantifies over the average number of minimal hops between all pair of vertices in a graph. Or, to be more concise:

Definition 7. The *average path length* for a graph G is expressed as:

$$\langle l \rangle = \frac{1}{n(n-1)} \sum_{i,j} d(v_i, v_j) : n = |V|, v_i, v_j \in V$$

where $d(v_i, v_j)$ gives the minimal number of edges between v_i and v_j.

Most real world networks exhibits a low average path length but a high clustering coefficient [12]. This stands in contrast to random networks who often have both a small average path length and a low clustering coefficient.

Chapter 3

How to measure the Direct Connect network

In order to perform analysis on the DC network we first have to determine what components that should be measured and how to measure them.

Since DC consists of an unbounded number of networks controlled by hubs, we have to decide on how many networks to investigate. The more networks studied the more accurate should the gathered statistics and data be. Unfortunately, performing a large-scale study is too time and resource consuming for this thesis work. We settle on choosing five random DC networks for our study. That should be enough to render trustworthy statistics and at the same time allow for comparison between them.

Basically, there are two entities of interest to measure; (1) peers and (2) their shared files. We will now discuss what data to collect and how to approach the task of collecting it.

3.1 Peers

A P2P network without interaction between peers is often quite uninteresting and the whole point of the system being a P2P network is lost. This is also the case in DC where peers not interacting with each other merely act as a set of clients connected to a hub. The situation becomes much more interesting when peers start searching for files, getting responses for queries and initiating file transfers. This is of interest because it allows for construction of a graph representing requesting peers, answering peers and the connections between them. Such a structure can provide valuable information such as existence of peers acting as potential providers/requesters, hit-rates of queries made in the network, vertex degree distribution, clustering coefficients, edge

symmetry and more.

So, how to capture activity between the participating peers?

An observer can either act as a peer or a hub. Either way, we cannot know when peers initiate file transfers between each other since that is handled in a pure P2P fashion. Another thing we cannot know is the matches sent to a peer from a query; that is also done in a pure P2P fashion. Since DC does not employ any protocol for endpoint authentication it theoretically opens up for man-in-the-middle attacks which could be used for monitoring traffic between peers. In practice, this would become a very complicated task though.

On the other hand, what we can know – regardless if we observe in the position of a peer or hub – is all peers' *exact search terms*. As discussed earlier, all queries are broadcast in plain text from the hub to connected peers. This fact opens up for *capture and replay of queries*. By this scheme querying peers can be constructed by captured queries and answering peers can be constructed using replies to replayed queries – thus, a graph representing peers interested in other peers can be created.

If considering acting as a peer when collecting data we immediately see some advantages: there is a huge set of existing hubs to choose from for data collecting. Also, it is possible and convenient to modify a ready-made DC client to match our intentions. A client for modification can either be a full-featured common client such as DC++ or some more rudimentary software such as a modified "bot"[1] client. A disadvantage of operating as a peer is that a peer must obey laws set by the hub operator. One common rule is restricted search interval times – which in our case will limit the replay frequency of captured queries.

If opting for the alternative of acting as a hub, we immediately see the benefit of being in total control as we would be hub operators. Collecting data and monitoring activities would be easy. The big issue against this choice is that we would have to attract a large set of users to the hub. This would probably take quite some time and be a really cumbersome task. In fact, this argument is so heavy that it rules out the alternative of acting as a hub. The same reasoning can be applied to the option of acting both as a peer and a hub. The hub alternative simply does not seem as the best choice for our purposes.

The above discussion leaves us with that the data collecting will be performed as a client connecting to a hub, monitoring activities, capturing and

[1] A bot – or robot – is in DC-terminology a small program which often acts as an operator in a hub, handling tasks like informing of rules, scanning peoples' share for illegal files etc.

replaying searches. Next, we need to decide on what data to focus on.

3.1.1 Activities

To analyze the peer activities in a hub says a lot about how people participate in a DC network. As most peer interaction with the hub is broadcast to the other connected peers we have a rich set of parameters to choose from when deciding on what data to collect. Below is a list of our items of interest.

Chatting As DC supports social features like real-time chatting this adds an extra dimension for the user of the network. It makes sense to find out how big part of DC is about communicating with other users.

Quits/joins Is the size of the hub steady, or does it increase or decrease?

File searching How big amount of peer activity is dedicated to searching for files? This number, in comparison to the amount of chatting will reveal how people use DC.

Country distribution How are nationalities distributed between peers? Are there some overrepresented nationalities?

These items will constitute the first part of our peer activity monitoring. We will now discuss the method used to analyze relations between peers.

3.1.2 Relations

The requester-provider graph

Our intention is to model a graph describing relations between peers in terms of searches and replies – or requesters and potential providers. First we have to consider our alternatives when choosing a graph model; should we use a directed or undirected graph and should we assign weights to the edges?

It turns out that the answer is simple; a P2P network graph is directed by peers requesting data and peers providing data. We have a case of a directed edge from a peer to a peer. Thus we will opt for a directed graph representation.

A directed connection between two peers tells that a peer can potentially be a resource provider to the other. Therefore, it is of interest to know how strong this connection is; that is, to measure how useful a peer can be for

another. We do this by introducing weights on the edges. This reasoning leaves us with a directed weighted graph to model our relations.

To simplify reasoning about peers and their roles, we introduce the notions of p_q for a requesting peer and p_r for an answering peer.

We recall that a query from p_q is received from the hub as:

```
$Search <clientip>:<clientport> <searchpattern>|
```

We add p_q as a vertex in our graph. Then the IP address and port number in the message is substituted with our own IP address and port number. The modified query is now executed by us sending the message to the hub. In other words, we have done a mimic of the action by p_q, but put ourselves as the sender.

We note the fact that the results received from the mimicked query will be the same results which p_q will receive.

When the mimicked search is executed, peers immediately start to answer us with results. We recall how a search response from a peer looks like:

```
$SR <replynick> <response> <hubip>:<hubport>
```

For each such response message, p_r is added as a vertex and a directed edge is created from p_q to p_r. Here our edge weights come to use; if there already exists a connection between p_q and p_r, we increase the edge weight by one and thus increases the importance of p_r to p_q. Figure 3.1 illustrates this reasoning.

The model theory is now taken care of, but the decision on what data to extract from the graph remains.

What to analyze?

We have already pointed out several interesting things to look at in a graph depicting DC. Below we sum them up and introduce a few new ones.

Symmetric links Are the edges connecting vertices symmetric or asymmetric? Some research has shown P2P networks to be highly asymmetric [11].

Weights The average edge weight will show the importance of the connections in the graph – a high number will hint on high affinity between peers in the network.

Degrees What are the minimum and maximum degrees for in and outgoing edges? What is the average degree in the network? Also degree distributions of undirected, out and in connections will be carefully studied. It is of particular interest to see if the degree distributions show characteristics of being a scale-free network.

Clustering coefficient How well connected are the peers in the network? We will find out if DC is a small-world network by examining the average clustering coefficient.

Average path length The average path length will be looked on and compared with other known networks.

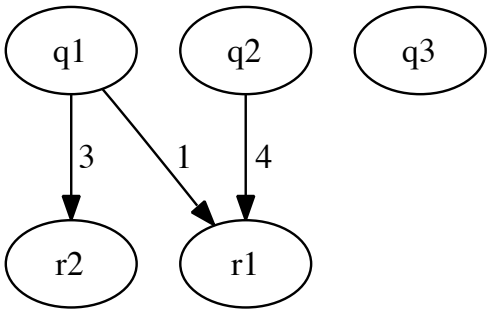

Figure 3.1: A small requester-provider graph. q_1, q_2 and q_3 are peers querying in the network. r_1 answers 1 query from q_1 and 4 queries from q_2, r_2 answers 3 queries from q_1. q_3 does not get any answer from its query, thus becoming an isolated vertex in the graph. q_1 and q_2 are potential *requesters* in the graph, r_1 and r_2 are potential *providers*.

3.2 Shared data

By analyzing the files shared in a DC network valuable data can be obtained. It is possible to determine what kind of files peers share and the proportions of the different file types can be revealed. To enable us to do this, a mechanism for automatic downloading of all peers' shared files lists will be included in our client.

We recall that each peer store a public list of his or hers shared files. This list is in XML-format and looks like:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<FileListing Version="1" CID="LZJ2HUGRF2UR45FPAA7JL5E55ULQO7PJDXOJEGI"
Base="/" Generator="DC++ 0.7091">
<Directory Name="Music">
  <File Name="Foo.mp3" Size="8189952"
  TTH="4UT2UPZPDWRAHMMLSY26KK7TXRXHTAABPO34DYY"/>
</Directory>
</FileListing>
```

Since each file shared has a corresponding hash stored in the TTH field, the amount of unique files shared in a network can be computed. This will give us a data redundancy factor for the DC network.

3.2.1 Correlation between peers and shared data

Again, graph theory will come in handy. This time the correlation between peers and their shared files will be analyzed.

As before, each peer will be represented as a vertex. This time, an edge connecting two vertices will represent that they share a common interest – i.e. that they have one or more files in common. We let the connection be undirected and weighted. The weight of the connection represents how many files that are shared by two vertices. This relationship is depicted in Figure 3.2.

Using this graph we will compute and analyze similar things like in the peer requester-provider graph:

Weights What are the ranges of the weights for the DC networks investigated? An average edge weight provides information on how many files the peers have in common.

Degrees Average degree, degree ranges and distribution will be considered. As the degree of a vertex tells how many peers that share files with the peer in question, this – together with weights – says a lot about data redundancy in the network. Also, the degree distribution will be looked upon.

Clustering coefficient As in the case of the requester-provider graph, it will be investigated if this graph shows characteristics of being a small-world network. In this case, a high number would indicate existence of clustered peers sharing similar "interests", i.e. files.

Average path length Here, this number will further help indicate how similar "interests" the peers have.

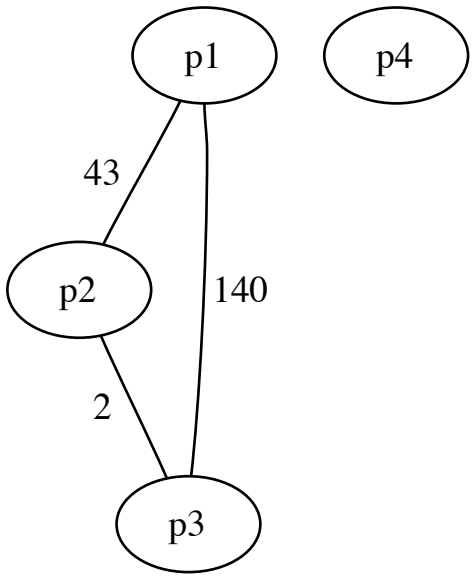

Figure 3.2: A small peer-data correlation graph. p_1 and p_2 has 43 identical files, p_1 and p_3 has 140 files in common, p_2 and p_3 has 2. p_4 has a set of unique files, thus becoming an isolated vertex. Note that this graph representation is undirected.

Chapter 4

System design

The system used for collecting data from peers is a client which connects to a hub. Separate software for analysis of shared file lists was constructed. We call these DCSpy and ShareStat, respectively.

This chapter deals with the implementation of DCSpy and ShareStat. Also, tools and libraries used are discussed.

Software user manuals can be found in the appendix section.

4.1 DCSpy

There exist several open source DC projects for both clients and bots. Rather than reinventing the wheel, we began by investigating the options to reuse and adapt any of these for our purposes.

Clients like DC++ and ShakesPeer[1] were considered for modification but rejected due to being too heavy and over-featured for our needs. When opting for the third alternative, an interesting framework for DC bots written in Java was found: jDCBot[2]. The jDCBot framework proved to be minimalistic and provided basic functions such as connecting to a network and setting up function callbacks for protocol messages received from the hub. Thus, parts of the jDCBot framework was used when constructing the DCSpy client.

The application was first developed as a pure command-line tool, but later got a graphical user interface connected to it. The user interface is far from finished and is currently operated in a hybrid state between graphical and command-line interface. Figure 4.1 shows DCSpy running and connected to a hub.

[1] http://shakespeer.bzero.se/

[2] http://jdcbot.sourceforge.net/

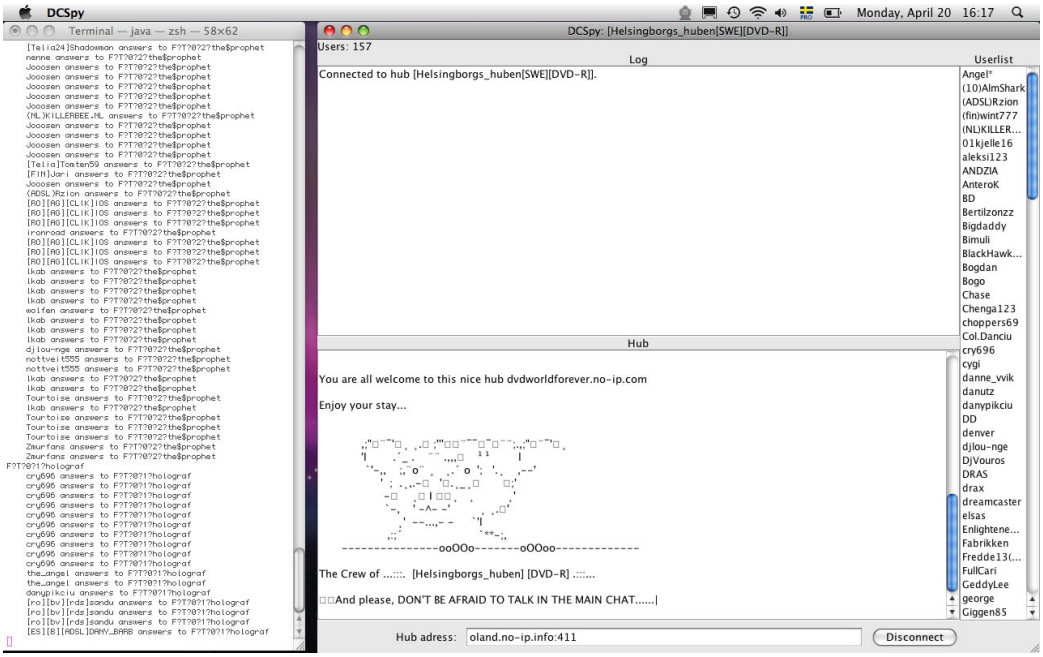

Figure 4.1: DCSpy running and connected to a hub.

The left window seen in Figure 4.1 is a terminal window showing the searches who are mimicked and the results to these. The other window is the main program window. It features three sub-windows; the upper showing miscellaneous log output, the lower presents the hub's public chat and the right window is a list of connected peers. Since the application's sole mission is to act as a watching eye, no kind of features facilitating user interaction exists. On the lower part of the main window, an address field for entering a hub address exists. When the user disconnects by clicking the disconnect-button, calculated data such as graphs, country distribution, clustering coefficients etc. are output in the terminal window. The downloaded file lists are to be found in the subdirectory called "filelists".

In order to be able to connect to hubs that require the user to share large amounts of (often copyrighted) material, DCSpy makes use of a technique called *fake sharing*. A huge list of shared files was downloaded from a random user on a random hub. The file sizes were summed up and that sum is reported as the amount of bytes shared by DCSpy. If another participant on the DC network wants to look at our list of shared files, the random list "borrowed" is uploaded for viewing. If then a peer requests to download one of our files, DCSpy responds with a message telling that no more slots for downloading is available. Using this scheme DCSpy reports on sharing

around 150 Gigabytes of files.

Needless to say, fake sharing is understandably not popular among hub owners and DC users in general. For the purpose of collecting data for a short period of time the method of fake sharing was considered reasonable.

4.1.1 Libraries and tools used

When developing DCSpy various tools and Java libraries were used; below is a short discussion on them.

JGraphT JGraphT[3] is a Java library which gives access to mathematical graph structures and algorithms. The library is released under the GNU Lesser General Public License. Since the library supports both directed, undirected, weighted and unweighted graphs (and more) it suits our needs perfect; all graph representations in DCSpy utilize JGraphT.

GeoIPJava In order to map IP numbers to country codes a Java library called GeoIPJava[4] was used. Like JGraphT, GeoIPJava is free software released under the GNU Lesser General Public License by its developer MaxMind. Besides the library providing the API, a large database containing mappings from IP numbers to country codes is available from the developer's homepage. The database is in binary format and provides fast and accurate lookups. According to the developer, the accuracy of the mappings is 99.5%.

py-dchub To test DCSpy a local DC hub was set up on the development machine. The DC client ShakesPeer and DCSpy were connected to the hub. Using this setup, it could be verified that DCSpy reacted appropriate to user action from the client connected to ShakesPeer. The hub software used is written in Python and called py-dchub[5]. The reasons for choosing py-dchub is that it is small, efficient and easy to set up.

[3]http://jgrapht.sourceforge.net/
[4]http://www.maxmind.com/app/java
[5]http://sourceforge.net/projects/py-dchub/

4.2 The ShareStat suite

Actually, ShareStat is two small separate tools developed to investigate the files shared in the DC network. The tools operate on the downloaded lists of shared files.

The first tool is a small Ruby program which scans all shared files, counts file types, checks for number of unique files and simply prints out the results. The process of file type matching is done by using case insensitive regular expressions. The mapping between file type and extension is based on mappings from the original NeoModus Direct Connect protocol. Table 4.1 summarize the extensions recognized by the program.

Type	Extensions
Audio	mp3, mp2, wav, au, rm, mid, ogg, mod, xm
Archive	zip, arj, rar, lzh, gz, z, arc, pak, lha
Document	doc, txt, wri, pdf, ps, tex, html, htm, xml, iso, bin, cue, m3u
Executable	pm, exe, bat, com
Picture	gif, jpg, jpeg, bmp, pcx, png, wmf, psd, tga
Video	mpg, mpeg, avi, asf, mov, wmv, ogm

Table 4.1: File types recognized by ShareStat.

The other tool is slightly larger. It works in a similar way as the Ruby program, but instead of checking file types it creates graphs describing correlations between users and their shared data (as discussed earlier). This tools is written in Java and just like DCSpy it utilize the JGraphT library for graph handling.

When analysis of the files is ready, and the program terminates, the resulting graphs, calculations and other values are output in the terminal.

Chapter 5

Results and data analysis

In this chapter results and statistics from the studied DC networks are presented and analyzed.

Much of the discussion in this chapter assume the reader to know how to interpret terms such as out/in-degrees, weights etc. in the context of our model of measurement. Figure 3.1 and Figure 3.2 in Chapter 3 summarize the concepts well.

The collecting of data was performed on five different randomly chosen DC networks, DCN1 - DCN5. The networks were found on Internet sites dedicated to providing listings of popular public hubs. The time connected to a network varied between 4 and 22 hours.

When mimicking queries in the networks, on average 7.5% of all queries could be replayed. All hubs had time constraints on search intervals, thus restraining us from achieving a higher number. Even if it had been theoretically possible to mimic all queries, it would be hard in practice due to the often massive amount of queries issued in a network; our system would be flooded with responses and probably collapse as a result.

For the analysis of shared files, a total of 1.1 gigabyte of file lists were downloaded from peers participating in the networks. The graphs representing the requester-provider and peer-data relations do not necessarily correspond regarding their respective number of vertices. This is because all lists of shared files could not be retrieved.

A graph depicting one of the studied networks – DCN2 – can be found in the appendix. DCN2 was chosen to illustrate a network because of it being the easiest to view, not having too many vertices and edges.

Country	DCN1 (Sweden)	DCN2 (Greece)	DCN3 (Romania)	DCN4 (Hungary)	DCN5 (Sweden)
Australia	1	1	0	0	0
Belgium	0	1	0	1	0
Canada	5	1	0	5	1
Czech Republic	1	5	1	1	0
Denmark	2	0	1	1	1
Finland	15	20	5	14	17
France	0	1	2	1	1
Germany	1	1	0	0	4
Great Britain	5	4	3	6	2
Greece	0	3	0	0	0
Hungary	2	1	1	3	1
Ireland	3	0	1	2	0
Italy	2	2	4	3	3
Latvia	3	1	0	3	1
Lithuania	0	5	1	0	1
Netherlands	3	3	2	5	1
Norway	4	4	3	6	9
Poland	5	2	11	6	1
Romania	10	19	47	9	7
Russia	0	2	0	1	1
Slovenia	0	1	0	0	1
Spain	1	0	3	0	1
Sweden	26	13	11	28	40
USA	5	5	1	1	1
Other	6	5	3	4	6

Table 5.1: Country distribution in percent for users in DCN1 - DCN5 and location of hubs. Note the overall high frequency of users from Finland, Romania and Sweden.

5.1 Geographic distribution

Table 5.1 shows the geographic distribution for the peers in the studied networks as well as the location of the hub. All in all, IP addresses originating from 43 different countries were recognized. The table shows only the most represented 24 countries. The remaining 19 countries are represented as "Other" in the table.

The absolute majority of the connections were made from almost all countries within Europe. In particular, Sweden, Romania and Finland were dominating countries for all networks studied.

No connections at all were made from South America and only two countries from Asia were recognized (India and Malaysia).

Although it seems like DC is mostly popular in Europe, this does not necessarily imply that people outside Europe do not use P2P file sharing; probably it is the case that they simply use other technologies than DC.

5.2 Peer activities

	DCN1	DCN2	DCN3	DCN4	DCN5
Chat	11%	0.06%	0.008%	11%	0.01%
Quit	3.9%	7.1%	5.6%	5%	13%
Join	3.7%	7.4%	5.4%	5.6%	13%
File search	75%	72%	76%	69%	52%
Other	6.4%	13.44%	12.992%	9.4%	21.99%

Table 5.2: Frequency of peer activities in the studied networks.

Table 5.2 presents statistics on peer activity for DCN1 - DCN5. All networks show the same trends; the ratio between peers leaving and joining the hub is similar and searching for files is the most common activity.

In most of the networks, almost no (at least public) chatting was done. DCN1 and DCN4 stick out by showing a chat frequency several magnitudes greater than the others though.

The field "Other" – as the name suggests – holds the other activities which can be recorded from the hub. Examples include changing of hub topic and broadcast of user information. The latter plays an interesting role for the outcome of "Other"; each time a user joins a hub all peers update each other with their user information. These updates are recorded as user activities since they are broadcast via the hub. This mechanism is reflected in the

correlation between a high rate of peers joining and a high rate of "Other" activities – that is, peers updating each other with user information.

5.3 Peer graph analysis

	DCN1	DCN2	DCN3	DCN4	DCN5
$\|V\|$	458	313	869	211	604
$\|E\|$	2601	432	5808	730	3940
$\langle k \rangle$	11.35	2.76	13.36	6.90	13.04
k_{out}	$0 \sim 104$	$0 \sim 55$	$0 \sim 253$	$0 \sim 83$	$0 \sim 98$
k_{in}	$0 \sim 51$	$0 \sim 9$	$0 \sim 76$	$0 \sim 23$	$0 \sim 105$
$\langle w \rangle$	1.21	2.18	1.20	1.23	1.30
w	$1 \sim 7$	$1 \sim 10$	$1 \sim 10$	$1 \sim 4$	$1 \sim 14$
$\langle l \rangle$	2.70	3.66	2.79	2.70	2.80
$\langle C \rangle$	0.20	0.03	0.27	0.16	0.20
Symmetric links	9	0	26	0	14

Table 5.3: Topological properties of peers in the studied networks.

In this section we analyze the graph built from mimicking searches in the studied networks. We begin by looking at Table 5.3 which presents topological properties of these.

Table 5.3 begins by listing the number of vertices $|V|$ and the number of edges $|E|$ connecting them. Further, the average degree $\langle k \rangle$, range of out degree k_{out} and in degree k_{in}, average weight $\langle w \rangle$ and range of weights w is listed. Finally, the number of symmetric links and average clustering coefficient is given.

By studying $\langle k \rangle$ for all networks we see that the average degree of a vertex in the networks is 9.48 – that is, on average each peer has 9.48 neighbouring peers. This means that a peer acts as a potential provider to 4.74 peers and potential requester from 4.74 peers.

In general, the maximum number of outgoing connections k_{out} in each network is greater than the maximum number of ingoing connections k_{in}. This hints on the existence of many providers that each serve a relatively small number of peers.

When comparing all networks' maximum vales k_{out} and k_{in} with the predicted $\langle k_{out} \rangle$ and $\langle k_{in} \rangle$ we find that the values differ a lot. This indicates that a small number of peers are highly active as requesters and/or providers.

The average weight $\langle w \rangle$ for all five networks is 1.424 which means that two peers having a connection are likely to connect 1.424 times. The maximum connection strength showed is in DCN5, where a connection weight of 14 is recognized.

All networks exhibit a low average shortest path, $\langle l \rangle$. The values found here are on parity with the average shortest paths found in many real world networks [1].

The clustering coefficients $\langle C \rangle$ are strong except in the case of DCN2. The clustering coefficients for the networks are all several magnitudes above what random networks would show. The strong cluster coefficients marks good connectivity in the network and implies that a DC network indeed exhibits characteristics of being a small-world network.

All networks shows to be highly asymmetric with only around 0.3% to 2% of the connections being symmetric. This tells us that when a peer p_1 is interested in something another peer p_2 has, p_2 is most often not interested in the files offered by p_1.

The figures in this chapter illustrating distributions all use a log-log scale and a complementary cumulative distribution function (CCDF). The CCDF is defined as $F_c(x) = P(X \geq x)$ where X is a numerical random value and P the probability function.

It is also worth pointing out that when for example depicting a degree distribution, only the set of peers satisfying $k > 0$ is considered. This policy holds for all figures showing distributions.

Figure 5.1, 5.2 and 5.3 shows the overall, out and in degree distribution for peers in the networks. We see that all networks seem to follow a common curve; the number of connected peers drops and leaves only a small number of peers with a huge number of connections.

Figure 5.2 and 5.3 reveal that the curves depicting the out-degree are the strongest. This suggests that there exists a good pool of content-providers in the networks.

It is interesting that DCN2 seem much weaker in all three figures discussed. The curve depicting DCN2 drops and dies much quicker than the other curves. This stands in direct relation to our findings in Table 5.3 where DCN2 shows weak statistics in comparison to the other four networks. The only number in DCN2 exceeding the other networks is the average edge weight $\langle w \rangle = 2.18$. This tells us that even though the average amount of connections are few, the connections made are strong. This suggests that DCN2 probably has a few peers acting as strong content providers.

In Figure5.4 the edge weight distribution is depicted. As in earlier figures, most networks follow the same pattern but with individual strengths. Noteworthy, DCN2 differs from the trend by having a remarkably steady

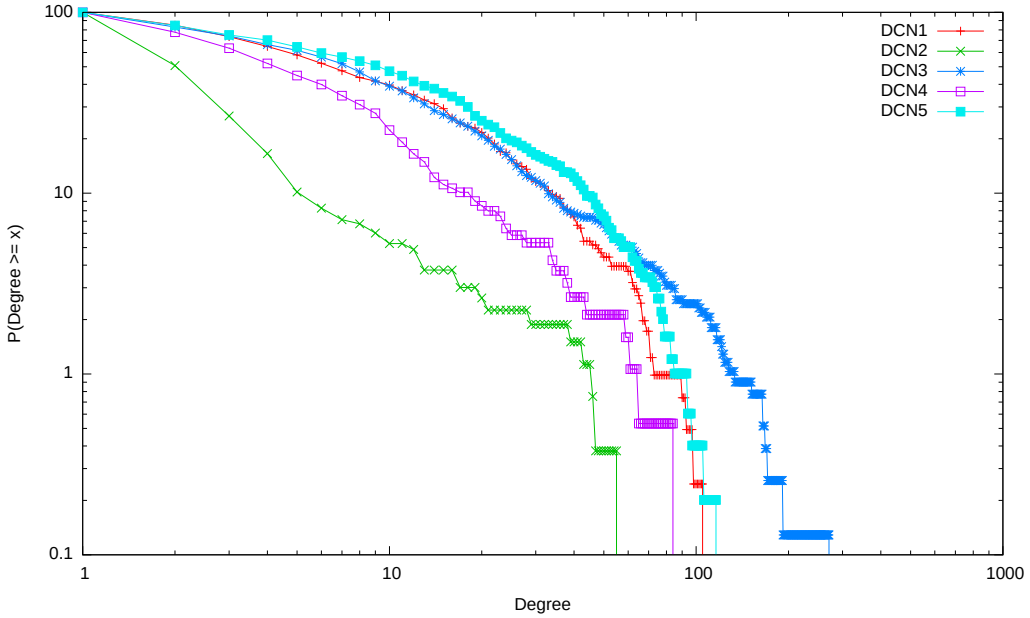

Figure 5.1: Vertex degree distribution from the requester-provider graph.

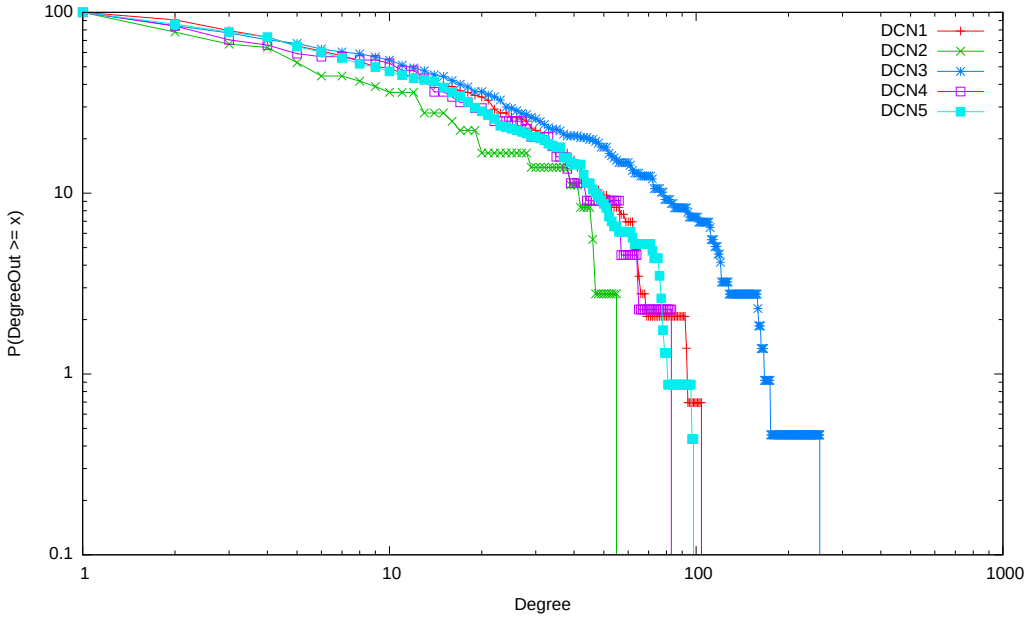

Figure 5.2: Vertex out-degree distribution from the requester-provider graph.

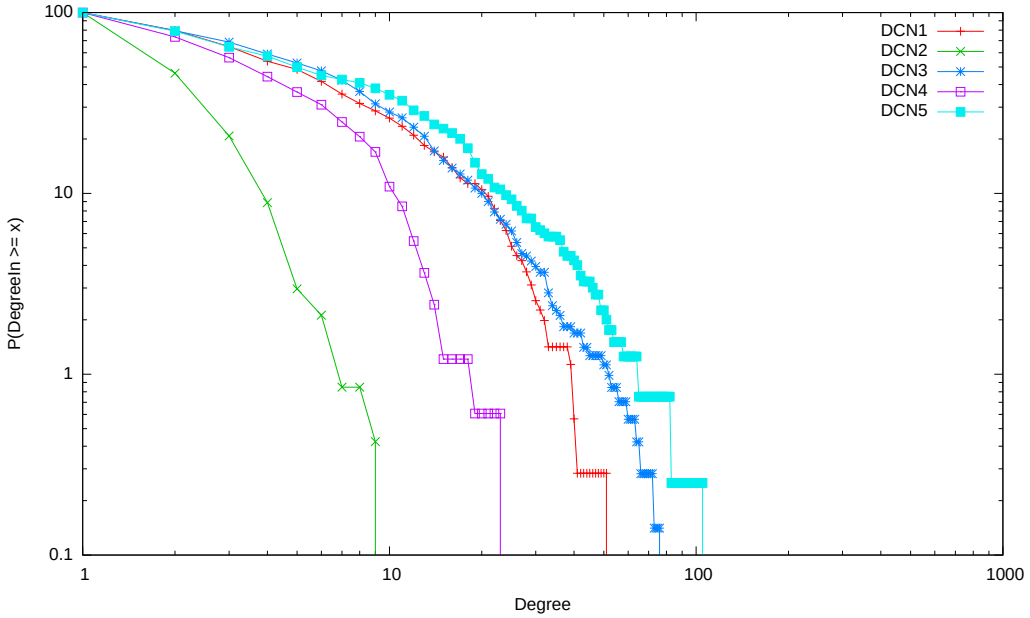

Figure 5.3: Vertex in-degree distribution from the requester-provider graph.

distribution of reoccurring connections among peers.

We continue our investigation of the peer graphs by looking at Figure 5.5 in which the fraction of peers having an out-degree, in-degree, both out- and in-degrees and peers being isolated. We see that 27% of the nodes are requesting peers and 76% are potential providers having answered one or several queries.

We note the intersection between the set of requesting and providing peers. This is the set of peers acting as both requesters and providers and constitute 16% of all vertices.

13% of the vertices are isolated, i.e. having made one or several queries but not gotten any answer.

All in all, this means that most peers are active as providers and some even as both requesters and providers.

To conclude this section, some key points regarding connectivity are observed:

Few peers have a rich set of providers, but when having that, the number is substantially large. And conversely, many peers act as providers, but not many serve a large number of peers.

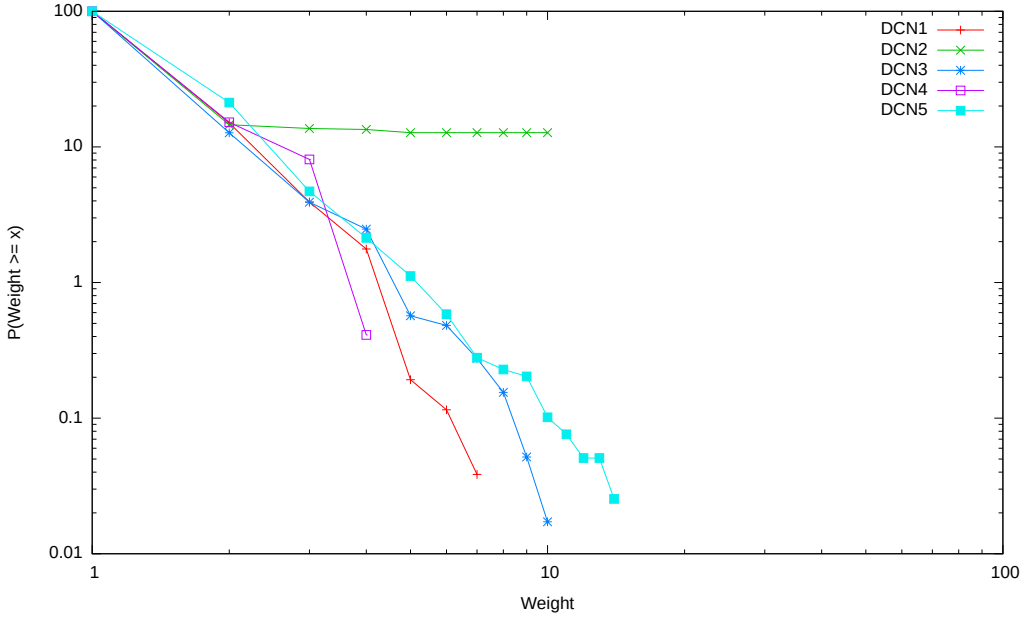

Figure 5.4: Edge weight distribution from the requester-provider graph. Note the steady distribution of DCN2.

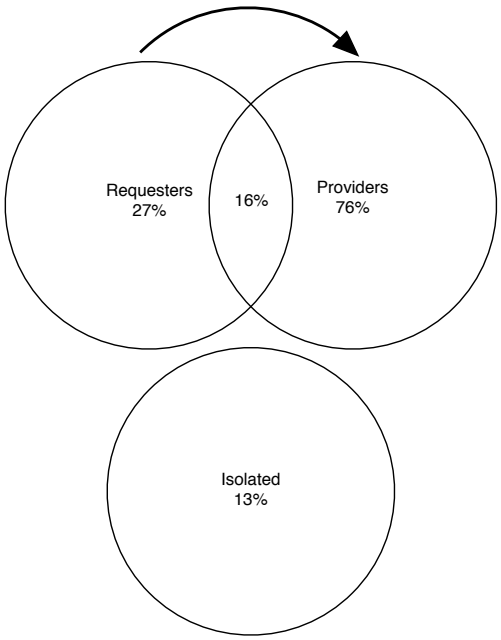

Figure 5.5: Diagram showing fractions of out-degree (requesters), in-degree (providers), both out- and in-degrees (the set intersection) and isolated peers.

5.4 Shared data graph analysis

We now turn to look at the data extracted from the lists of shared files downloaded from the studied networks DCN1 - DCN5.

Table 5.4 gives a basic presentation of file type frequency, percentage of unique files shared, number of files shared as well as the total size of them.

A quick glance at the table reveals that of the recognized file types, the set of audio files is the most well represented followed by the set of known picture types. Although not very surprising, it is interesting to see that the "Other" category has such a high percentage. Since "Other" is the complement to the set of recognized files[1], it is predicted to give a high percentage. Most probably the "Other" category holds a multitude of different file extensions which are more or less frequent. It is not likely that it represents some file type which is close to the frequency of the audio files. This reasoning gives that we can most certainly conclude that audio files are the absolutely most common files in DC networks

Audio	43%
Archive	1%
Document	7%
Executable	1%
Picture	17%
Video	2%
Other	29%
Percentage of unique files	58%
Total number of files	10235120
Total size of files	104 TB

Table 5.4: Statistics on shared files.

Table 5.4 also shows the percentage of unique files shared in the studied networks. We note that 58% of the shared files are unique; this means that roughly two out of five files in the networks are duplicates, i.e. redundant in the network.

Using the total number of files shared and their total size we obtain that on average, each file shared is around 10.65 megabyte in size.

Table 5.5 has a similar layout as Table 5.3 but shows the topological properties of the graph describing correlations between the peers and their shared data.

[1]See Table 4.1 for a presentation of all recognized file extensions

	DCN1	DCN2	DCN3	DCN4	DCN5
$\|V\|$	217	422	394	139	369
$\|E\|$	6496	30276	16714	1948	8430
$\langle k \rangle$	59.87	143.48	84.84	28.02	45.69
k	$0 \sim 154$	$0 \sim 299$	$0 \sim 230$	$0 \sim 85$	$0 \sim 209$
$\langle w \rangle$	40.73	90.27	67.18	69.90	19.66
w	$1 \sim 38311$	$1 \sim 68630$	$1 \sim 82035$	$1 \sim 12771$	$1 \sim 18295$
$\langle l \rangle$	1.77	1.69	1.76	1.91	1.90
$\langle C \rangle$	0.73	0.76	0.73	0.70	0.68

Table 5.5: Topological properties of the correlation between peers and shared files in the studied networks.

Here, the average degree $\langle k \rangle$ tells us the average of how many other peers P that have at least one file in common with a peer p. The average degrees are quite high; in fact, when looking at all networks, on average p has at least one file in common with 23% of the elements P.

The average edge weight $\langle w \rangle$ shows the average number of files a peer p_1 has in common with a peer p_2, given that they have any in common at all. The strongest connection is in DCN3 where there exist peers that have 82035 files in common.

As expected, the average shortest paths, $\langle l \rangle$ all show to be low; this is interpreted as an indication on peers having a quite homogeneous taste.

The clustering coefficients for the sets of shared files is very strong for all networks with DCN2 showing the strongest coefficient $\langle C \rangle = 0.73$. This is not very surprising since the parameters show characteristics of possibly being well connected graphs.

Figure 5.6 and Figure 5.7 illustrates the distribution for vertex degrees and edge weights for all shared files in the studied networks.

The distributions for the networks in Figure 5.6 all follow the same pattern with varying strength. We note that the trend is that peers have data in common with many other peers; the probability decreases with the degree until a point where the it rapidly drops. The network having the most users with common files are DCN2 – which previously has showed weak statistics. These weak statistics might be explained when looking at Figure 5.6 and Figure 5.7. It seems like many peers in DCN2 have many files in common with many other peers. This could result in a lack of attractive files not common between the peers, thus resulting in weak graph statistics. The content of the network can be said to be "inbred" – peers are having too many files in common for the network to be interesting for its participants.

In Figure 5.7, it is noteworthy that the patterns of the networks are very similar. We see that the probability of having many files in common with other peers is not especially large and steadily decreases as a function of the number of files.

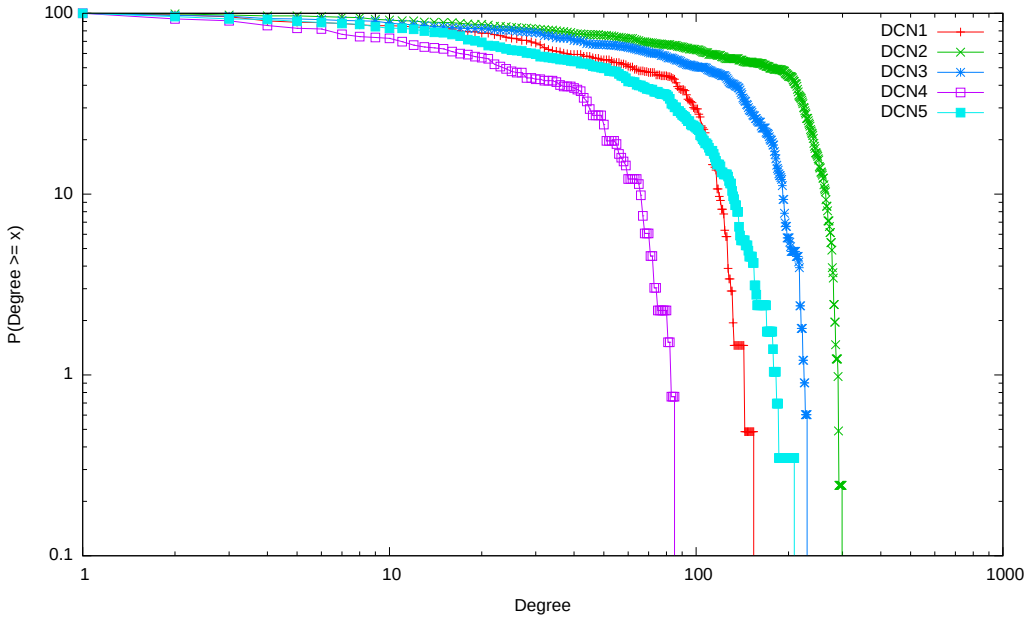

Figure 5.6: Vertex degree distribution from the peer-data graph.

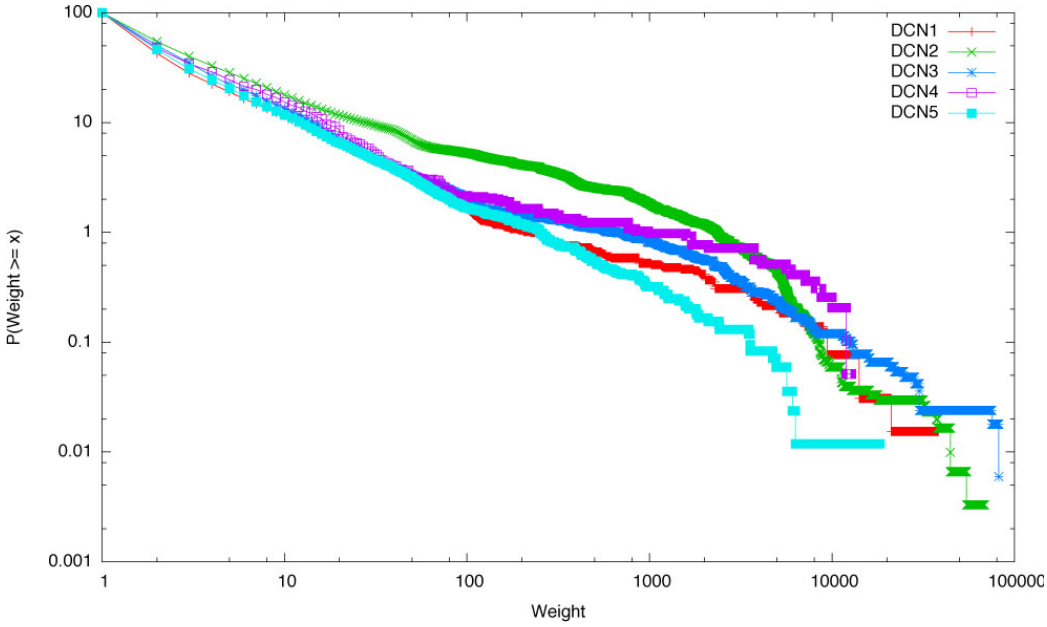

Figure 5.7: Edge weight distribution from the peer-data graph.

Chapter 6

Discussion

In this chapter we discuss the implications of our findings presented in the previous chapter. We will cover topics like geographic distribution, peer activities and connectivity, the content shared as well as the correlation of peers and their data.

6.1 Geographic distribution

It seems like the use of Direct Connect is highly concentrated to the Scandinavian countries (except Denmark) and Romania.

Two things can be noted about this; first, it was surprising to see such a big representation of Romanian peers. It is not apparent why this is the case. Most probably the explanation is a combination of several factors. One of the hubs used in the study was Romanian, hence an increased possibility of attracting many Romanian users. Another explanation can be that it might be the case that DC has gained a strong user-base in Romania, having many users connecting to hubs located both inside and outside Romania.

A second observation is that when Sweden, Finland and Norway show on high representation, Danish representation is diminutive. An explanation for this could be possibly stricter Danish laws against file sharing of copyrighted material.

It also should be remembered that we obtained the hub addresses from public Internet listings; our geographical findings might as well say that "most people using public hubs found on the Internet are from Scandinavia or Romania".

6.2 Activities

Although two of the networks both showed a quite high frequency of chatting, this does not necessarily prove strong social interaction between users. It is often the case that hubs implement a bot acting as a "quiz master", delivering questions and hints in the hub's public chat. As such messages often come within a very short time interval, statistics on chatting can be greatly affected and thus somewhat misleading. In the case of DCN1 and DCN4, the high frequency for chatting was actually caused by such a bot.

It seems like the overwhelmingly majority of DC users ignore the social functions (at least the public functions) and more or less only focus on traditional file sharing.

6.3 Peer connectivity

By looking at obtained data and distributions of both degrees and weights the DC network, in terms of requesters and potential providers of content, indeed exhibits properties of being scale-free.

Since the distributions does not adhere to an exact power law function, DC cannot be said to be a pure power law network though.

We have seen that there exists a large base of providers each potentially supplying content to a smaller number of peers. We have also found evidence for peers being highly active. This activity implies making many queries in the network and/or being a potential provider to many of the other participants.

The affinity – or connection weight – between peers is generally low in DC. This implies that two peers are not likely to interest each other much more than one time. This strengthens the observation of peers acting as major providers are rare.

Furthermore, we have showed that the DC network show high clustering coefficients, i.e. being a small-world network. This finding further strengthens our discussion about the network accommodating peers acting as highly active – or *hubs* – in the structure. Since it is often the case that a small-world network is scale-free, this also helps verifying our finding of DC being a scale-free network.

As we will see, most of our findings can be confirmed by findings from research on other P2P file sharing systems and social networks.

Mislove et al. [6] have performed a study on online social networks[1]. These networks are reported to exhibit power law, small-world and scale-free

[1]Flickr, YouTube, LiveJournal and Orkut.

properties. The degree distributions found by Mislove et al. are very similar to those found in the DC network. Key differences between their findings and ours, is that online social networks show a high degree of link symmetry and, more, the clustering coefficients found are slightly higher than the ones found in DC. The high degree of link symmetry can be credited to the fact that relations captured by social networks most often are bidirectional. The high clustering coefficients in online social networks are explained by Mislove et al. as expected in these kind of networks; people often get introduced to other people via mutual friends, which will result in the "small-world phenomenon" and thus, a higher clustering coefficient.

The structural analysis of the Gnutella P2P file sharing network made by Wang et al. [11] also verifies many of the above discussed properties. As expected (since both Gnutella and DC being file sharing networks), the properties of the Gnutella network showed even more similarities with DC than with online social networks. Like DC, the Gnutella network exhibited a very asymmetric connectivity. Differences do exist though. Gnutella's clustering coefficients were much weaker than in DC. Moreover, Wang et al. showed that Gnutella seems to have a larger set of strong providers than DC.

6.4 Content and Peer–data correlation

Before starting to investigate the peer overlay structure of DC, we had quite strong suspicions on what to discover. This was not really the case before investigating the content being shared in the networks; we could assume that common data would include audio, pictures, videos and archives but not to what extent.

As shown in the previous chapter, audio files were the absolute majority of data being shared. Picture files turned out at a good second place. From this we draw the simple conclusion that most people use DC to share music files.

Another interesting finding was the data redundancy factor of roughly 40%. This implies that availability of data is good despite the dynamic nature of P2P networks.

We also looked at the correlation between peers and their shared files. It turned out that, roughly, a peer have at least one file in common with 25% of the other participants. Not surprisingly, the number of files in common with other peers were not that strong; it is quite unlikely that two peers should possess a nearly identical set of shared files.

The clustering coefficients for the network formed by the peer-content correlation showed to be extremely high. From this we can draw the con-

clusion that there exists many strongly connected clusters of peers sharing a similar taste, i.e. content. Moreover, from this (and the low average path length) it can be speculated that the users of DC share a quite homogeneous taste.

Chapter 7

Summary

We have successfully modelled a method of analysis, developed a set of software tools and performed measurement and analysis of the DC P2P architecture.

Our method used was based on foundations from mathematical graph theory as well as from previous research in the field of P2P. These abstract tools proved to function strong, concise and reliable. Besides analyzing only the obvious part of the DC network – the participants – we applied the same set of theoretical tools to the content being shared. This not only added depth to our research, it also shed light on the user–data correlation that exists in P2P file sharing networks.

The tools developed to perform the actual monitoring, data gathering and post-processing performed highly satisfactory. On the downside, since pieces of DCSpy were based on a ready-made framework of what showed to be of questionable quality, that particular implementation unfortunately ended up a bit messy at places. Probably that implementation would have benefit from being written in a modern, more compact high-level language like Python or Ruby instead of Java. The data post-processing tools ended up having nicer code though.

The study of DC presented in this thesis is – if not the first – probably the first of its kind. We found a network with much in common with a range of online social networks, and in particular, with other file sharing networks such as Gnutella. We also noted strong data redundancy and clusters of common interest in the set of shared content.

The scale-free properties found along with good data redundancy speaks for high fault tolerance in the networks studied. Although this most often is good, it makes a DC network vulnerable to strategic attacks of peers acting as hubs in the network overlay graph. This is not the weakest point though; since the network cannot exist at all without the central hub entity, this is

in fact the single point of failure.

The hub is also the bottleneck when it comes to the capabilities of scaling; a DC network scales proportional to the hub's capacity of handling the connected peers.

A third flaw related to the hub in a DC network is worth being mentioned. The NMDC protocol has a feature of redirecting connected peers to other hubs. This is supposed to be used when the hub has reached its maximum user limit. If the hub is compromised, it would be possible to redirect all connected users to a target of choice, thus performing a DDoS[1] attack. If performed by many hubs having many users, this could cause a great deal of damage.

DC has also proved to be quite unique in the sense of "self-sanitizing"; every hub acts as a small community, employing operators and robots for the purpose of preventing users from fake-sharing and sharing of inappropriate material. The potential problem of users "free-riding" – that is, only downloading and not contributing to the network – is avoided by having minimum share size rules to enter hubs.

We have also seen collaborations between hubs in the form of hub networks.

All gathered data and developed program source code can be made available upon request from the author.

7.1 Future work

It would be interesting to perform a large scale study on DC networks, monitoring hundreds or even thousands of hubs for several months. This could be done by sequentially crawling addresses to hubs from websites providing hub-listings. Comparison of the results of such a large scale study with the results presented in this thesis could then be done.

There could be more done using the data gathered. Investigation of finding a best-fit power law function for degree distributions or looking at clustering coefficient distributions are examples of this.

The hub addresses listed publicly on the WWW probably only stand for a small subset of all existing hubs. There exist hubs which are known only to a few chosen people which probably act more community-like. For natural reasons, these hubs are much harder to find (and get access to), but it would be of interest to compare data from those hubs with the data presented in this thesis.

[1] Distributed Denial of Service.

An intriguing question which this thesis has overseen is how availability of data varies over time. To monitor the sustainability of the connected peers would give good information about this. In particular – as they play an important role in the network – it would be of interest to look at the sustainability of the peers acting as larger providers.

Experiments using various graph rendering technology could also be made. To take parameters such as interests and peer sustainability in account when rendering a graph might be interesting.

The DCSpy client is far from user friendly; it should be decided on whether to go for a graphical user interface, or a command-line user interface. In its current state, it is something in between.

Bibliography

[1] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, January 2002.

[2] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system, 2000.

[3] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Friend-to-friend data sharing with OneSwarm. Technical report, University of Washington, department of Computer Science and Engineering, February 2009.

[4] Jon Kleinberg. The convergence of social and technological frameworks. *Communications of the ACM*, November 2008.

[5] Michael Miller. *Discovering P2P*. SYBEX Inc., 2001.

[6] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference*, 2007.

[7] Annalee Newitz. Sharing the data. *Metro, Silicon Valley's Weekly Newspaper, July 12-18*, 2001.

[8] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system, 2007.

[9] Matei Ripeanu. Peer-to-peer case study: Gnutella network. In *Proceedings of the First Intenational Conference on Peer-to-Peer Computing*, 2001.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001, San Diego, CA, August 2001*, 2001.

[11] F. Wang, Y. Moreno, and Y. Sun. Structure of peer-to-peer social networks. *Physical Review E 73, 036123*, 2006.

[12] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, (393):440–442, June 1998.

Appendix A

Software manual: DCSpy

DCSpy is a tool developed during the writing of this thesis. Its purpose is to connect to a Direct Connect hub, monitor activities in the hub and perform data processing and analysis.

Usage

DCSpy requires three arguments; desired nickname, IP address of client and port for incoming connections (as DCSpy operates in active mode).

As DCSpy makes use of some external packages[1], the Java classpath should be set accordingly – both when compiling and running the program. Below is an example of how to start the client after compilation, using the nickname cheburashka, IP address 127.0.0.1 (localhost) with port 2002 open for listening:

```
$ java -cp .:tools/GeoIP.jar:tools/bzip2.jar:tools/jgrapht.jar
DCSpy cheburashka 127.0.0.1 2002
```

You should now be presented with a window similar to the one seen in Figure A.1. As DCSpy outputs some of its log data to the standard output stream, it might be a good idea to have the terminal window adjacent to the DCSpy window.

Next step is to enter a hub address and port in the address-field in the lower part of the window. Default is `127.0.0.1:2001`. When satisfied with the address, the user might press the "Connect" button to log on to the hub.

If everything went well, the window is populated with various information; number of users connected to hub, a list of the users, log information and finally, the hub's public chat.

[1] Discussed in the chapter describing the system design.

From this point DCSpy starts collecting information about the hub and its users. It tries to collect all users' lists of shared files (stored in `./filelists`), user activities are logged and searches are mimicked. Figure A.2 shows how a session with DCSpy can look like.

To disconnect from the hub, the "Disconnect" button should be pressed. This makes DCSpy process the gathered data and finally the results are written to the standard output stream.

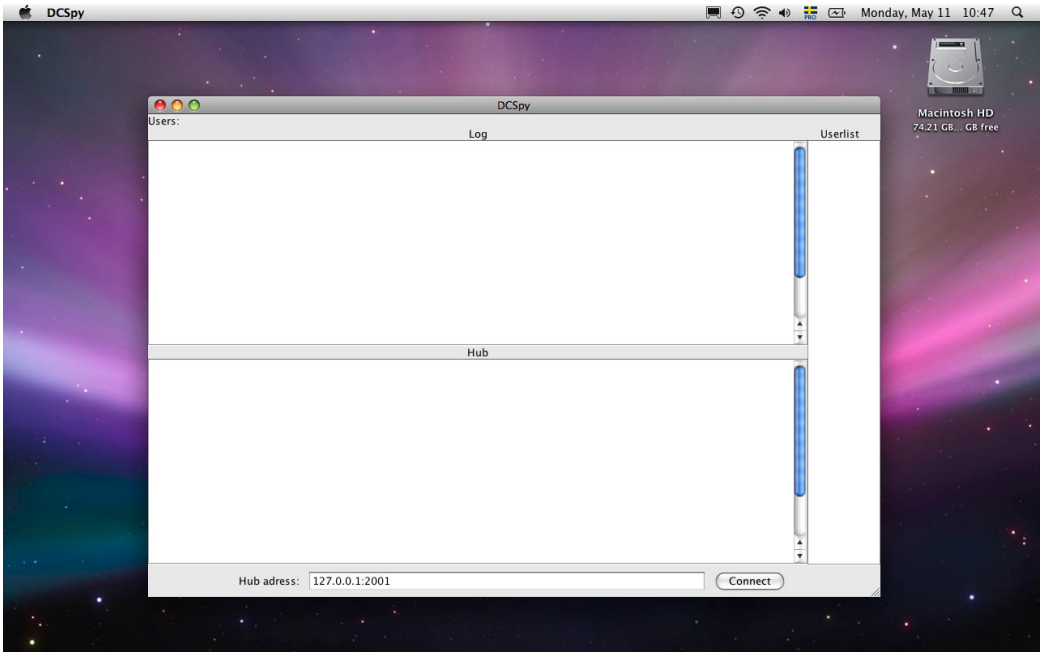

Figure A.1: DCSpy waiting for connecting to a hub.

Other / bugs

As discussed in the thesis, DCSpy currently uses both a graphical and a text-based user interface. Since the program's purpose is to act as a pure monitor, it would probably suffice to keep the interface as text-only.

When developing the program, little thought was spent on the usability; the focus was on the task of collecting data and processing it in an adequate manner.

There exists some known bugs. When connecting to a hub not implementing the standard Direct Connect protocol, the process of logging on can halt. Also, when disconnecting, occasionally there can occur an exception from one of the threads.

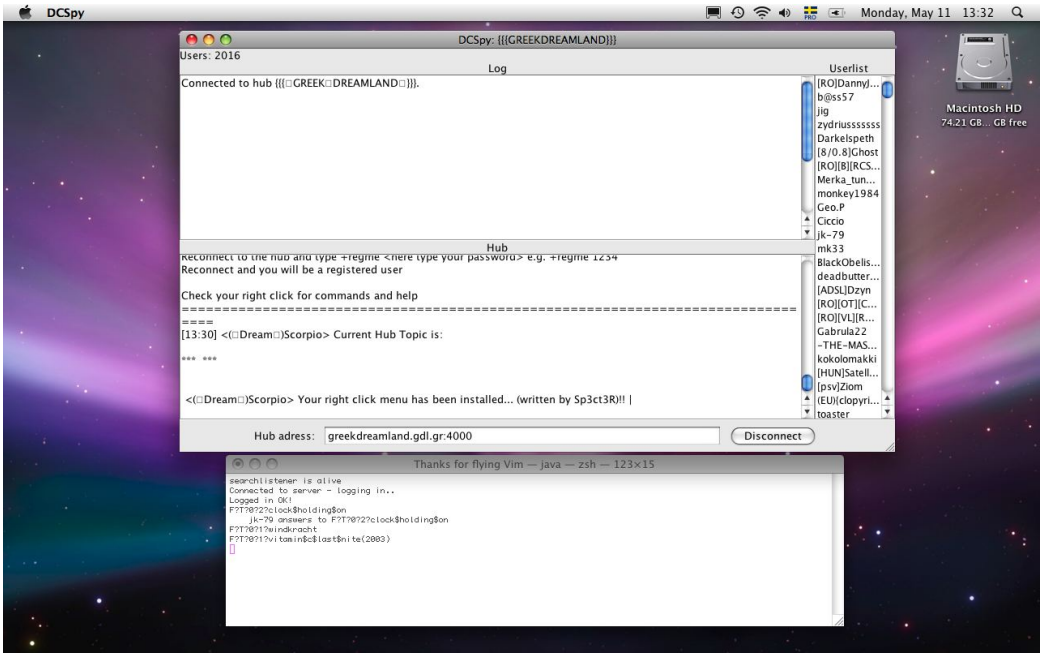

Figure A.2: DCSpy connected to a hub.

These bugs do not endanger the data gathered and they should not be particularly hard to fix. They have not been fixed due to time constraints and prioritization of other tasks.

Appendix B

Software manual: The ShareStat suite

Before discussing the programs in more detail, a word about data input:

The input to the programs in the ShareStat suite is expected to be a valid and decompressed Direct Connect list of shared files (in XML format). If wishing to perform statistics calculations on a set of many files together, a handy tip is to first concatenate them using for example the Unix `cat` command.

B.0.1 ShareStat

ShareStat is a Java program which constructs a graph describing correlations between users and the files shared. Also, several graph properties are calculated and presented. Examples of such properties are average edge weight, average vertex degree and graph clustering coefficient.

Usage

ShareStat takes only one argument – the name of the file containing all file lists that will be used when constructing the graph. Often the amount of data to process is huge, and Java often finds itself running out of memory. Therefore it is a good idea to increase Java's memory allocation pools when executing the program. After the program has been compiled, it can be used as:

```
$ java -Xms32m -Xmx2048m -cp .:jgrapht.jar ShareStat filelists.xml
```

(Also note that the jgrapht graph package is needed by ShareStat.)

Other / bugs

No bugs are known in ShareStat.

B.0.2 analyze.rb

The Ruby script analyze.rb investigates the frequency of different file types in users' lists of shared files. It also computes the number of unique files and the total size of all files.

Usage

The program only expects one or several files as parameters. Example of usage:

```
$ ruby analyze.rb cheburashka.xml
```

This will examine the user cheburashka's list of shared files.

Another example:

```
$ ruby analyze.rb gena.xml shapoklyak.xml
```

This examines gena's and shapoklyak's shared files.

Figure B.1 illustrates a sample output.

```
Number of audio files: 35660 (23.0922654509662%)
Number of compressed files: 2515 (1.62863285499663%)
Number of document files: 3525 (2.28267626793763%)
Number of exe files: 441 (0.285577371393048%)
Number of picture files: 21068 (13.6429570533078%)
Number of video files: 983 (0.636559084080195%)
Number of other files: 90232 (58.4313319173186%)
Number of unique files: 119726 (77.5306947106667%)
Total number of files: 154424
Total size of files: 0.137216388612615 terabyte
```

Figure B.1: Example output from the analyze.rb script.

Other / bugs

The script is easily extendible for recognition of other file suffixes since it uses regular expressions for matching.

No bugs are known in analyze.rb.

Appendix C

Direct Connect graph

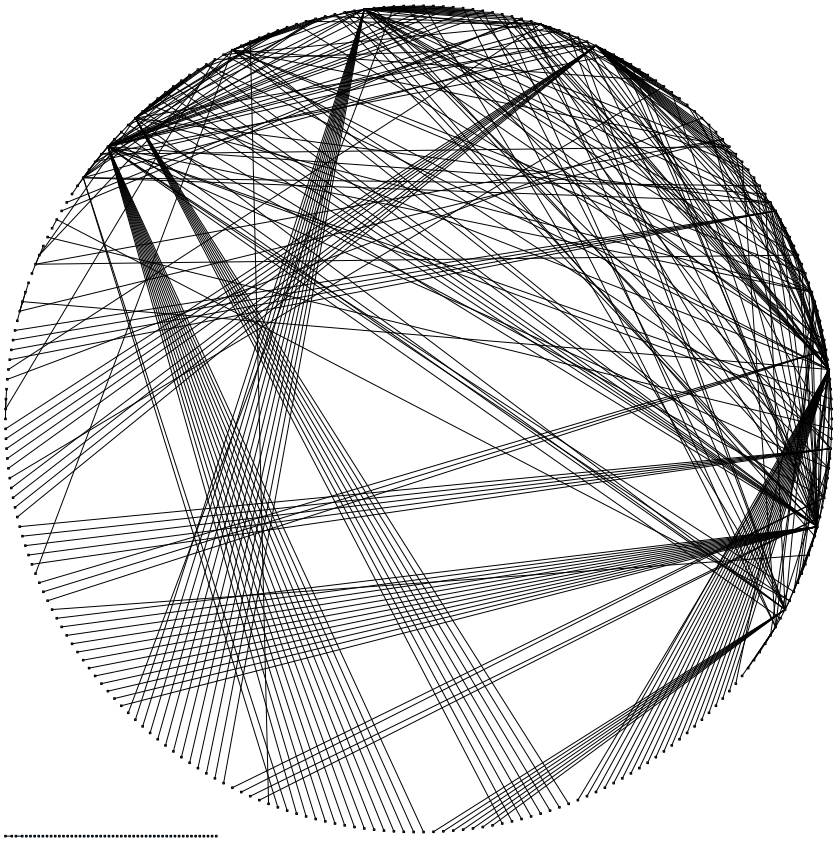

Figure C.1: A graph illustrating DCN2, one of the studied Direct Connect hubs. The dense half is comprised by requesters and/or providers. The other half shows pure providers.