Secure Decentralized Swarm Discovery in Tribler

Jelle Roozenburg

Parallel and Distributed Systems Group
Delft University of Technology

TU Delft

Delft University of Technology

Secure Decentralized Swarm Discovery in Tribler

Master's Thesis in Computer Science

Parallel and Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Jelle Roozenburg

10th November 2006

Author
Jelle Roozenburg
Title
Secure Decentralized Swarm Discovery in Tribler
MSc presentation
17 November 2006

Graduation Committee

prof. dr. ir. H. J. Sips (chair)	Delft University of Technology
prof. dr. ir. M. J. T. Reinders	Delft University of Technology
ir. dr. D. H. J. Epema	Delft University of Technology
dr. ir. J. A. Pouwelse	Delft University of Technology

Abstract

The decentralized architecture of peer-to-peer (P2P) networks solves many of the limitations of conventional client-server networks. This decentralization, however, creates the need in P2P file sharing networks to find peers who are downloading the same file, a problem which is referred to as *swarm discovery*. In the BitTorrent file sharing network, swarm discovery is solved using a central server (a tracker), which is unreliable and unscalable.

We have designed a decentralized swarm discovery protocol, called LITTLE BIRD. LITTLE BIRD is an epidemic protocol that exchanges swarm information between peers that are currently or were recently members of a swarm. A quantitative measure of the contribution of each peer is calculated to make the protocol efficient and secure against peers that pollute the system with malicious information. Furthermore, we have conducted detailed measurements of the BitTorrent community '*Filelist.org*', in order to study download swarm behavior and optimize the design of our protocol. We have implemented the LITTLE BIRD protocol as an addition to the Tribler P2P network.

For the evaluation of LITTLE BIRD, we have created an experimental environment called CROWDED, which enables us to conduct large-scale trace-based emulations of swarms on the DAS-2 supercomputer. Evaluation results show that our protocol scales with the swarm size, uses moderate bandwidth overhead, and is resilient against denial-of-service and pollution attacks. Therefore, LITTLE BIRD enables secure, scalable and reliable swarm discovery.

Preface

This document describes my MSc project on the subject of secure decentralized swarm discovery for the Tribler peer-to-peer network. The research was performed at the Parallel and Distributed Systems Group of the Faculty of Electrical Engineering, Mathematics, and Computer Science of Delft University of Technology.
This research is done in the context of the Freeband/I-Share project, in which research is conducted on the sharing of resources in virtual communities for storage, communications, and processing of multimedia data.
I want to thank the many people who gave me great help during my research. In the first place my supervisors Dick and Johan, for their extensive contributions to my thesis report and research. Furthermore, Jie, Jan David and Pawel for providing detailed information on BitTorrent and Tribler, and Alexandru, Hashim and Paulo for helping me with the emulations on the DAS-2, the operation of KOALA, and the design of the CROWDED emulation environment. I would further like to thank prof. dr. ir. H. J. Sips for chairing the examination committee, and prof. dr. ir. M. J. T. Reinders for participating in the examination committee.

Jelle Roozenburg

Delft, The Netherlands
10th November 2006

Contents

Chapter 1

Introduction

> *It is vain to talk of the interest of the community, without understanding what is the interest of the individual.*
>
> — Jeremy Bentham

Peer-to-peer (P2P) networks offer a new way of efficient communication and cooperation between computers. They are scalable and more reliable, without a central server that acts as a bottleneck. Services are distributed over all computers in the P2P network, instead of implemented on a single computer, which can become unresponsive or overloaded. P2P networks are mostly used for file sharing, because they can make enormous amounts of content available to many downloaders. File sharing networks can profit from a virtual social community on top of it. A social community helps to structure and filter the file sharing network and stimulate members to be more cooperative. Tribler is a P2P program that builds a distributed social community around the BitTorrent file sharing protocol. To implement such a community over a P2P network, decentralized community management is needed. In this thesis, we will design, implement and evaluate a community discovery protocol for the Tribler social-based file sharing network.

Research for the Tribler project is conducted in the context of the Freeband/I-Share project [30]. Freeband is a Dutch research program that aims to develop a new generation of sophisticated communication technology. The I-Share project focuses on multimedia data sharing in virtual communities.

In this introductory chapter we give a more detailed overview of P2P networks and virtual communities. In Section 1.1 we explain how P2P networks solve many of the problems of the classical client-server architecture and we introduce the technological challenges of P2P systems. In Section 1.2, we describe BitTorrent, a popular P2P file sharing network on which we will focus in our thesis. BitTorrent supports faster downloading and higher availability of files through fair exchange of file pieces between downloaders. Tribler, introduced in Section 1.3, is a P2P file sharing application that adds many social features to the BitTorrent protocol. We will use Tribler as a basis for our software by implementing decentralized swarm discovery for it. In Section 1.4, we discuss the growing importance of virtual communities around file sharing networks. These social networks give new incentives

to users of file sharing networks to be available and cooperative. To create social networks in a P2P architecture, we need distributed community management, described in Section 1.5, which combines the advantages of P2P and social networks. We conclude with an outline of our thesis in Section 1.7.

1.1 Peer-to-peer systems

The classical architecture for computer communication over networks is the *client-server* architecture (see Figure 1.1a). The server is a central computer which offers services to requesting client computers. All the data and software are stored in a centralized fashion on the server. The best known client-server network is the world wide web.
The disadvantage of the client-server architecture is that the maintainer of the server is responsible for most of the needed resources of the system. When a website becomes more popular, new hardware and bandwidth have to be bought to handle the growing number of requests. A server computer is also a *single point of failure*, which means that the total system becomes unusable when the server is unavailable.
In contrast to the client-server architecture, the P2P architecture distributes resources and responsibilities over all computers in the system. There is no hierarchy amongst the computers, hence the name *peers*. All peers play the role of both server and client and communicate using a symmetrical protocol where they both send and handle requests. A typical P2P network is shown in Figure 1.1b. In both networks in Figure 1.1, a file consisting of three parts is distributed over the connected computers. In the client-server network, the server bandwidth scales

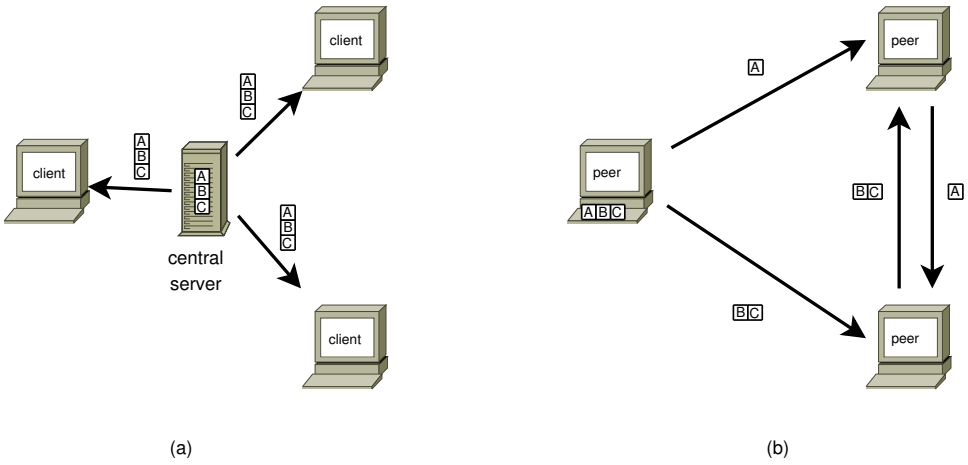

Figure 1.1: The distribution of a file consisting of three parts [A,B,C] through a client-server architecture (a), and through a P2P network without a central server (b).

up with the number of clients. In the P2P network, peers actively redistribute the file parts and the bandwidth of the initial distributing peer does not grow with the popularity of the file.

The distributed architecture of P2P systems has shown to be very effective against single points of failure and the growing need for resources in servers. However, there are some important challenges that have to be addressed in order to realize an effective P2P network, namely *availability*, *incentives*, *integrity* and *distribution*.

Availability and incentives When the quality of a service depends on the number of peers that are online and cooperate, their availability becomes very important. These peers will only donate their resources if they benefit from it, so the system needs incentives to stimulate peers to cooperate and punish peers that *freeride*. Freeriding is the term for taking (much) more resources from a P2P system than one donates.

Integrity Peers in a P2P system are less reliable than a central trusted server. So maintaining integrity of information and data received from other peers is an important research challenge. This is closely linked to security in P2P systems; peers must be able to differentiate an attacker from a cooperating peer.

Distribution The complexity of these challenges is increased because they have to be implemented distributed over all peers. Otherwise, P2P systems would lack scalability and flexibility. Many problems that are reasonably simple to solve in a centralized way are hard to solve in a distributed P2P system.

The P2P architecture can be used for different systems, for instance the Skype telephony network [79] or for chat services like ICQ [39] or MSN Messenger [56], but most popular are P2P file sharing networks. Figure 1.2 shows the percentage of the total Internet bandwidth that is used for P2P file sharing and other services like web-browsing and email over the last 14 years. The enormous growth in P2P file sharing bandwidth during the last eight years gives an indication of the popularity of P2P file sharing networks. The most used P2P file sharing networks are BitTorrent [19], FastTrack (client Kazaa [44]), Gnutella [34]/ Kad Network [52] (client LimeWire [49] or eMule [24]). In this report we will focus on the BitTorrent network.

The absence of a central server in a P2P network protects it against censorship. Since content is not stored on a single place, it can not be easily located and removed. This idea has been developed into fully anonymous file sharing networks using a P2P setup, for instance Freenet [17] and GNUnet [45]. Also, plug-ins are developed to add an anonymity layer to existing P2P networks, like Tor [22] and I2P [38]. On these networks, content can be retrieved without revealing the owner. Peers are used as anonymous proxies to route the data to the requester.

1.2 BitTorrent

The BitTorrent P2P system was designed and implemented by Bram Cohen in 2003 [20]. It is a P2P file sharing system that enables multiple downloaders of a certain file to benefit from each other, by exchanging pieces of the file (see Figure 1.1b). This is called *bartering*. Bartering makes exchange of content scalable, i.e., when content becomes more popular, its availability also increases, because there are more people to barter with. Hence the peer that first uploads the content only needs a constant amount of upload-bandwidth, independent of the popularity of the content.

BitTorrent focuses on *efficient* and *fair* bartering of content. Efficiency is guaranteed by downloading multiple pieces of the content from different peers in parallel, and making these pieces available to others directly. Also, the least available pieces are downloaded first, to guarantee equal availability of all pieces. Fairness is guaranteed by bartering using the *tit-for-tat* [20] protocol. Tit-for-tat strives for an honest exchange of data between peers and a high *sharing-ratio*. The sharing-ratio of a peer is the total amount of data uploaded to other peers divided by the total amount of downloaded data. Protocols and rules that stimulate or oblige peers to donate their upload resources are called *sharing-ratio enforcement*.

Tit-for-tat is designed so that a peer is most loyal to those peers that were most loyal to it. This gives peers an incentive to be available and donate bandwidth to the BitTorrent network. Also, it solves a part of the freeriding problem by stimulating cooperation [4].

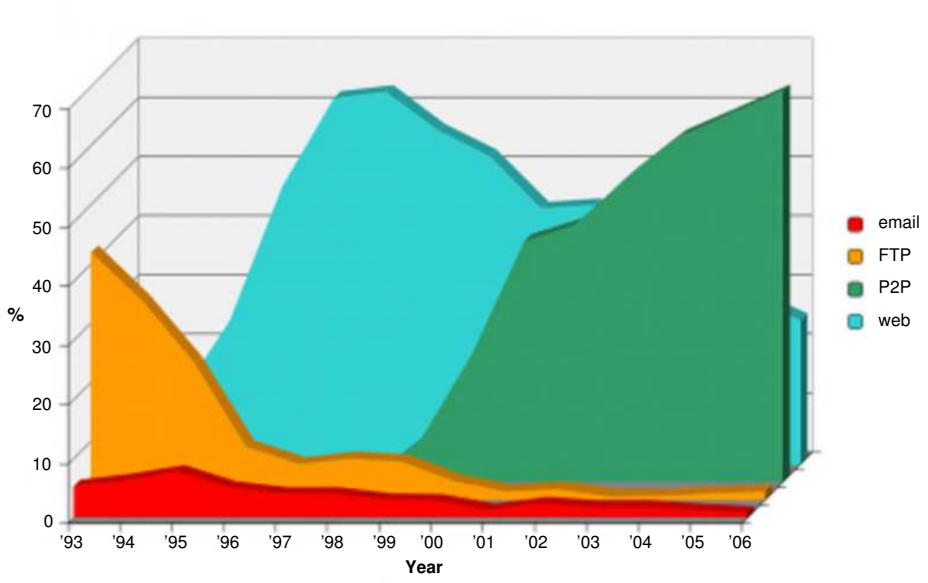

Figure 1.2: The percentage of total Internet traffic consumed by P2P file sharing networks during the last 14 years.

The BitTorrent system is built from different components, some centralized, some distributed. Searching for content is done in a centralized fashion through conventional websites. For each content item, a file with meta-data is created, called a *torrent file*. The torrent file contains the address of a *BitTorrent tracker*. The tracker is used to discover the connection properties of the group of downloaders of this torrent. The total group of downloaders of a torrent is called the *download swarm*. In a download swarm we distinguish peers that have fully completed their download, called *seeders*, and peers that are still busy downloading, called *leechers*. Leechers will use the tit-for-tat protocol to receive a honest sharing-ratio. Seeders do not need to download anymore data and will just increase the torrent's availability without expecting returns. The discovery of the swarm members is called *swarm discovery*. Swarm discovery is needed before a peer can start bartering. It will be explained in detail in Chapter 2.
Protection of the integrity of received pieces becomes an important issue, as pieces are received from different anonymous peers. Content integrity is guaranteed through a list of hash values of all file pieces, which is stored in the torrent file. All peers test their downloaded data against these hash values before they accept it. When peers send erroneous file pieces, they will automatically be removed and re-downloaded from more reliable peers.

1.3 Tribler: social BitTorrent

After years of research on P2P networks and BitTorrent communities, the Parallel and Distributed System group of the Faculty of Electrical Engineering, Mathematics, and Computer Science initiated the design of a new P2P network named Tribler [71]. Tribler is based on the open source BitTorrent client ABC [1] and uses the BitTorrent protocol for general file transfer. Tribler adds a growing list of features to BitTorrent file sharing, focused on adding a social network and streaming media. In the Tribler network, peers are made identifiable so that they are no longer anonymous computers, but real people. Each Tribler user has a *permanent identifier* (permid), which enables it to identify other users when ever they come into contact. Social groups can now be created, like personal *friends* or users with similar download taste. The social network, build from these users around you is used to provide additional social services.
Cooperative downloading [33] is a social feature through which a user can request its friends to help it download a file. With the combined bandwidth of its friend, downloads will be completed faster. When peers are helping their friends with downloading, they will omit the tit-for-tat protocol of standard BitTorrent. In this case the social friendship relation gives users an additional incentive to donate their bandwidth to other users of the network.
Another Tribler feature is *BuddyCast*, a distributed recommendation and content discovery protocol [64]. BuddyCast is a epidemic or gossiping protocol in which all users exchange *preference lists* which represent their current taste. Through

the exchange of preference lists, each user can find a group of users with similar taste, called *taste buddies*. These taste buddies are used for content discovery and recommendation. A more detailed description of the BuddyCast recommendation protocol is given in Section 4.5.1.
Many additional features are in development in the Tribler project, like friends-import from other social networks, social-based sharing-ratio enforcement, firewall puncturing and Tribler browser integration. Tribler will also offer streaming media over its P2P network [72]. Tribler users will then be able to broadcast live video streams or video-on-demand to the whole network over a P2P architecture.
During our thesis work we designed and implemented a decentralized swarm discovery solution for the Tribler software. Our swarm discovery implementation will make the Tribler network more scalable by removing its need for BitTorrent centralized trackers.

1.4 Virtual communities

During the last decade, a growing number of people have found the Internet to be an easy place to make contact with people world wide. The Internet has become so popular that a part the social life of people exists in cyberspace instead of in physical space. The existence of *virtual communities* is an example of this. A virtual community is a group of people on the Internet that form social aggregations based on common interests. This definition is so broad that we can distinguish different types of virtual communities.
Examples of communities that are created for solely social purposes are Orkut [60], Friendster [31], and Hyves [37]. These communities organize existing social relations in a network structure so that people can browse the network and get in touch with new friends. Inside the community, members can form sub-communities and compose their own lists of friends. These online communities also have the infrastructure to chat and exchange content.
Apart from purely social communities, there are many communities that have additional purposes. Examples of *information sharing* communities are Flickr [29], where people can post and share photos, YouTube [78] for video sharing, and Wikipedia [76], which is an encyclopedia written collaboratively by contributors around the world.
We are particularly interested in *file sharing* communities, which are created on top of file sharing networks like the BitTorrent network. They combine social contact with the exchange of files. These file sharing communities are an effective way to protect the content integrity by posting comments and discussions. Fake or poor-quality content is recognized by the community members and removed or labeled as fake. Social communities can hence form a collaborative filter [75] for file sharing networks. Furthermore, the social relations that are created inside communities create additional incentives to cooperate with the file sharing network. People tend to help each other more when peers are seen as real people with similar interests,

instead of downloading computers. Tribler's cooperative download feature (discussed in Section 1.3) is an example of this.
Examples of popular BitTorrent communities are Filelist (see Section 3.3.1), Piratebay [62], and Mininova [53].

1.5 Decentralized community management

The technical architecture of all popular virtual communities still follows the client-server model. Information about the members and their relations and all other community data is stored and managed by a central server.
Our vision is that communities and P2P file sharing networks can both profit from each other. Using P2P technology in virtual communities makes them scalable, cheaper, more fault tolerant, and more resilient against censorship. Communities are a valuable addition to P2P file sharing networks because of the collaborative filtering and incentive advantages.
In order to combine these two powerful ideas, we need scalable *decentralized community management* as a way to manage social communities around a file sharing network in a fully decentralized fashion using P2P.
The most important function of decentralized community management is creating a local view of the active peers that form the community around you following a P2P methodology. This has many applications, for instance when a peer wants to retrieve the members of a certain community or downloaders of a single file.
In this thesis we focus on *swarm discovery* as part of decentralized community management. Swarm discovery is the detection of active users in a download group on a P2P file sharing network. The network information of these users is needed to connect to them and start exchanging content. We show that existing swarm discovery solutions are not fully reliable and scalable. Therefore, we have designed a secure and decentralized swarm discovery solution, called LITTLE BIRD, which we have implemented in Tribler.
The LITTLE BIRD protocol uses epidemic communication between peers to distribute swarm information. When swarm information is gathered from many untrusted peers, the problem of reliability of swarm information is introduced. To prevent attacks in which peers pollute a swarm with malicious information, we have created a quantitative measure of the contribution of peers to the BitTorrent and LITTLE BIRD protocol. Peers that have a low value of this contribution measure will be automatically excluded from the protocol.
We have conducted detailed measurements of a BitTorrent community called *Filelist*. We have measured the behavior of users and statistics about BitTorrent download swarms. Our measurement results are used to optimize and evaluate the LITTLE BIRD protocol. Our decentralized swarm discovery protocol has been evaluated by large-scale emulations on a supercomputer. In these emulations, we reenacted the measured behavior of all peers in a swarm, using the actual Tribler applications with LITTLE BIRD support. Evaluation results show that we managed to create a

scalable and secure protocol.

1.6 Contributions

In this thesis, we make the following contributions:

- We study the behavior of BitTorrent users and swarms in detail using scraped data from the BitTorrent community 'Filelist.org'.
- We present LITTLE BIRD, a decentralized swarm discovery protocol with a focus on security and scalability.
- We develop the CROWDED emulation toolkit in order to carry out large-scale emulations of the LITTLE BIRD protocol on a supercomputer.

1.7 Thesis outline

The remainder of this thesis is organized as follows. In Chapter 2 the problem of swarm discovery will be described and existing solutions and their drawbacks are given. To get detailed knowledge about how swarms and their members behave, we have carried out a study in the BitTorrent community '*Filelist.org*', presented in Chapter 3. Chapter 4 presents our swarm discovery solution for the Tribler network, called LITTLE BIRD. Also, we discuss how our LITTLE BIRD protocol meets the design requirements presented in Chapter 2. In Chapter 5 we evaluate the LITTLE BIRD protocol for Tribler in emulation experiments based on measurements of the Filelist.org community. We have developed an emulation toolkit, called CROWDED, to carry out these large scale emulations on the DAS-2 super computer. Conclusions and recommendations are presented in Chapter 6.

Chapter 2

Problem Definition

I do not seek. I find.

— Pablo Picasso

In this chapter we define the problem that is the main subject of this thesis: the discovery of a swarm in a secure and scalable way. In Section 2.1 we describe the problem of swarm discovery: finding peers that are downloading the same content. In Section 2.2 we define the design requirements for a secure and scalable swarm discovery solution. In Section 2.3, we discuss existing swarm discovery solutions and review how they meet these requirements. We conclude in Section 2.4 that none of the existing solutions complies with the requirements. Therefore, we will design a new decentralized swarm discovery protocol in Chapter 4.

2.1 Swarm discovery

In P2P networks, there is no central computer where all information and services reside. The information and services are distributed over all peers in the network. In practice, a P2P network is very dynamic, with peers constantly joining and leaving. Peer discovery is the problem of finding cooperating peers in a dynamic P2P network.

In P2P file sharing networks we are particularly interested in discovering peers that are downloading the same content. These peers are in the same BitTorrent *download swarm*. Hence, *swarm discovery* in the BitTorrent network is the problem of finding nearly every peer that is bartering pieces of a certain file.

The information about peers in a swarm that has to be found consists of IP addresses and listening ports of each peer. With these network addresses, a peer can start connections to its swarm fellows. We will refer to a list containing the network addresses of the peers in a swarm as a *peerlist*. After connecting to the discovered swarm peers, the bartering process is initiated, which allows a peer to start downloading and uploading pieces of the file.

2.2 Design requirements

Below, we define five design requirements that have to be met in a quality swarm discovery solution. We will use these requirements to review current swarm discovery solutions and as a motivation for the design of our protocol.

Bootstrapping The initial phase in swarm discovery is called bootstrapping. During the bootstrap process, a small number of initial swarm peers are discovered and connected. The subsequent discovery of more peers in the swarm is relatively easy after bootstrapping is successfully completed, because the initial swarm peers can be used for this. Bootstrapping is important, because when it fails, subsequent discovery of a larger part of the swarm is impossible.

When a centralized swarm discovery solution is used, as in the BitTorrent protocol, bootstrapping is not necessary. Each peer knows the static address of the central tracker, which is their initial and subsequent source of swarm information. If peers however depend on a decentralized swarm discovery solution, they will always have to start with a bootstrap step. Because our goal is to design a scalable, and therefore decentralized, swarm discovery solution, bootstrapping is an important requirement.

Swarm coverage Peers need to know enough other peers in the download swarm to be able to use the BitTorrent protocol efficiently. We will call the subset of the download swarm that a peer knows its *swarm coverage*. The goal of swarm discovery is to maximize the swarm coverage of peers, but in bigger swarms it is not necessary to have full swarm coverage. As long as a peer knows enough swarm members to have all pieces of the content available to it, it can barter without the limitation of content unavailability.

When swarm coverage is too small, a swarm can become poorly connected, or even partitioned. The partitioning of a swarm results in two disjoint subsets of the swarm in which no peer knows any of the peers in the other subset. Partitioning is more likely if the swarm knowledge of the peers is not random enough, i.e., if the swarm graph is not a randomly connected graph. The formation of local knowledge (cliques) and the constant stream of peers joining and leaving the swarm, might break up the swarm in separate parts.

Scalability and load balancing Scalability and load balancing are important factors in the design of a swarm discovery solution. For swarm discovery to be scalable, the overhead for each peer has to be minimal and only has to grow slowly with the number of simultaneous downloads or size of the download swarms. Overhead should be evenly divided over all peers in a download swarm (load balancing). For swarm discovery, the overhead consists of bandwidth, CPU usage and storage space. Bandwidth is the resource that is most scarce. In reality, some peers will donate more bandwidth to a P2P network than others, but they should have the possibility to influence this individually.

Incentive to cooperate When swarm discovery is distributed over a network of peers, they need to donate resources (mostly bandwidth) to help others with their discovery process. If peers have no natural incentives for this donation, they tend to behave like rational agents, which only follow protocol if it will maximize their utility [69]. This lack of an incentives mechanism has lead to many freeriding peers in the Gnutella network. Measurements show that only 1% of the Gnutella peers handles half of the file requests, and 70% of all peers do not share any files [3].

There has been much work on providing incentives for cooperation in P2P networks. BitTorrent uses the tit-for-tat protocol (explained in Section 1.2) to give peers an incentive for fair bartering. Other research focuses on reputation systems [57, 42, 2] or systems in which cooperative peers are rewarded by the reception of credits [35, 74]. Feldman et al. [27] take a game-theoretic approach to incentive techniques for P2P networks. They use a model in which all peers save a shared history stored over a distributed infrastructure. For a successful decentralized swarm discovery protocol, similar incentives are necessary to stimulate peers to cooperate and create a reliable system.

Integrity and security Integrity and security of a P2P network are important issues to consider when decentralizing swarm discovery. When a central tracker is used, people trust this third party by name and are sure that the server will give genuine peer addresses as a result. In case of decentralized swarm discovery, the responsibility of delivering a reliable service lies in the hands of every peer. Each query result could be an attack of a malicious peer. We will focus on two types of attacks which could harm a P2P file sharing network, namely *pollution attacks* and *distributed Denial of Service* (dDoS) attacks.

In a pollution attack, a group of peers tries to reduce the quality of a P2P file sharing network by poisoning it with fake content or information. It has been reported that the music industry uses this kind of attack to fight piracy in P2P file sharing networks [59, 16, 48].

In case of swarm discovery, a pollution attack would try to disable a download swarm by spreading as much erroneous peerlists as possible. The injection of non-existent network addresses in the swarm will pollute the peers' knowledge, so that they cannot connect to each other anymore and the content becomes unavailable.

Instead of trying to pollute a download swarm, attackers can also try to exploit it for a dDoS attack. In a dDoS attack, an attacker tries to shut down a victim's network service by letting multiple computers send many useless network packages to its address (see Figure 2.1). David Moore et al. show that dDoS attacks are a common threat on the Internet [55] and give detailed measurements of over 68,000 dDoS attacks. Many publications on counteracts show that dDoS attacks are indeed a realistic threat [43, 14, 77].

In the case of swarm discovery, an attacker could send the IP address of a victim's web server in a peerlist to many peers, stating that this peer is active in the download swarm. The peers that receive this IP address could all try to connect to

this address and thereby flood it with network packages. We will describe dDoS exploits related to distributed hash tables in Section 2.3.3.

2.3 Current solutions

In this section we will describe four currently used swarm discovery solutions, namely a *centralized tracker*, *multiple trackers*, *distributed hash tables*, and *peer exchange*. For each solution we discuss to what extent they meet the requirements mentioned in Section 2.2.

2.3.1 Centralized tracker

The decentralized design of P2P systems is one of their major advantages. It makes the BitTorrent system flexible, scalable and reliable. The BitTorrent tracker is one of the parts of the system that still has a centralized design using a client-server architecture.
Peers send swarm discovery requests to a BitTorrent tracker using the HTTP protocol. The tracker replies by sending a peerlist with swarm information. No bootstrapping is needed, because all peers in the swarm can read the static tracker address from the torrent file.
The client-server architecture of the BitTorrent tracker makes its bandwidth grow linear with the amount of peers and is therefore an unscalable swarm discovery solution. Bram Cohen, the inventor of BitTorrent, recognizes that the BitTorrent tracker is indeed a bottleneck of the system [20]. Currently, it is already noticeable that trackers have long response delays and peers often have to do multiple tracker

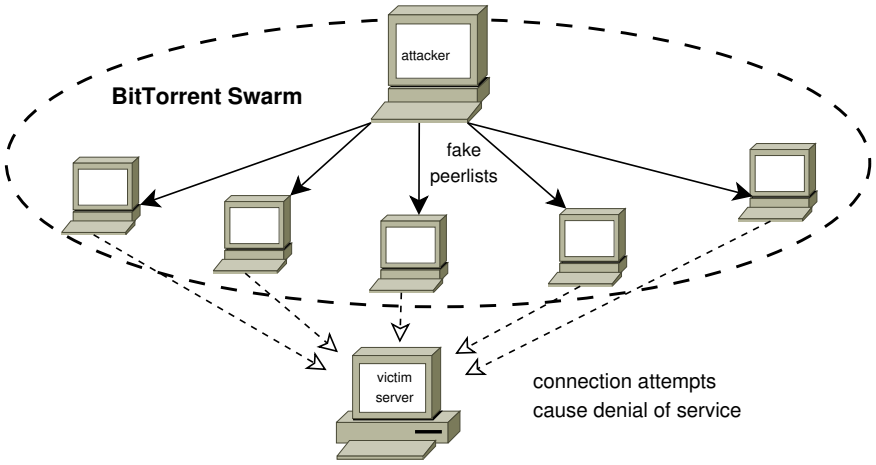

Figure 2.1: An attacker can flood a swarm with the single IP address of a victim computer. If the receiving peers trust the attacker, they will connect to the victim and cause a dDoS attack.

requests due to time-outs. When the number of BitTorrent users will grow in the future this problem will even grow in such a way that building a reliable centralized tracker will be undoable or at least very expensive.
A centralized tracker is also a single point of failure in the BitTorrent system. This means that its failure will cause an interruption of the BitTorrent service of enabling people to join and use download swarms. While the P2P architecture of BitTorrent gives a reliable download environment through thousands of users going online and offline at each moment, a irresponsive tracker immediately makes it impossible for new peers to join a download swarm. The BitTorrent measurement study of Pouwelse et al. [63] also concludes that central components of BitTorrent result in an increased risk of system unavailability.
The lack of reliability and scalability of central trackers under a high request load are shown in Table 2.1. We have measured the response times of the Piratebay [62] tracker after requesting peers for three torrents, every 20 minutes for a day. Half of our requests remained unanswered (we used a socket time-out of one hour). For the other requests, it took the tracker 2 to 3 minutes to respond.

Torrent file	Successful responses (%)	Average response time (s)
Torrent 1	51.6	211.7
Torrent 2	52.3	139.7
Torrent 3	52.3	191.7

Table 2.1: The response rate and average response time of the Piratebay BitTorrent tracker.

The current centralized BitTorrent tracker does not meet our most important requirement, namely to be scalable and to have a balanced bandwidth usage. All bandwidth that is used for swarm discovery has to be donated by the central tracker, which forms a bottleneck to the BitTorrent system.
A final disadvantage of the current centralized trackers in the field of security and integrity is that they are an easy target for an attack at the BitTorrent system. A single dDoS attack can stop a tracker, leaving all downloading peers without means to discover each other and to continue their downloads.
The client-server architecture of centralized trackers makes it relatively simple to add extra functionality besides peer discovery. Trackers often have a feature to give statistics about the downloading peers. More importantly, BitTorrent communities with restricted access use their trackers to enforce a more strict sharing-ratio enforcement. For the Filelist.org community, this is explained in Section 3.3.1.

2.3.2 Multiple trackers

In addition to the single BitTorrent tracker explained in Section 2.3.1, the BitTorrent protocol has been modified to create the possibility of having multiple tracker addresses in a torrent-file [9]. If the first tracker in the list responds to requests, communication between peers and the tracker will be normal. When the primary

tracker is down or has a long response time, clients will try the next tracker from the list until a working tracker is found or the list is exhausted. This addition does not solve any of the problems concerning the lack of scalability of centralized trackers, it merely increases its reliability by a constant factor at the cost of more trackers. It does, however, offer some extra flexibility for torrents that are available through multiple trackers.

2.3.3 Distributed hash tables

A popular decentralized approach to swarm discovery uses distributed hash tables (DHTs). A DHT is a distributed database divided over a network of computers, referred to as *nodes*. All nodes can store (key, value)-pairs on the DHT and later find values by searching for keys. Store and search actions are performed by finding the node responsible for the search key and then sending it a STORE or FIND VALUE command. The responsible node is the node closest to the search key, with a certain quantitative measure of distance. The responsible node is found by sending requests to nodes closer and closer to the search key. A requested node will either return the address of a node closer to the search key, or it is the responsible node itself.
When DHTs are used for swarm discovery, the addresses of all swarm members have to be stored for each BitTorrent swarm. This can be implemented by using the hashes of swarms as keys and the peerlists as values. In this way, a DHT can replace the centralized BitTorrent tracker. For each swarm, a unique peer or group

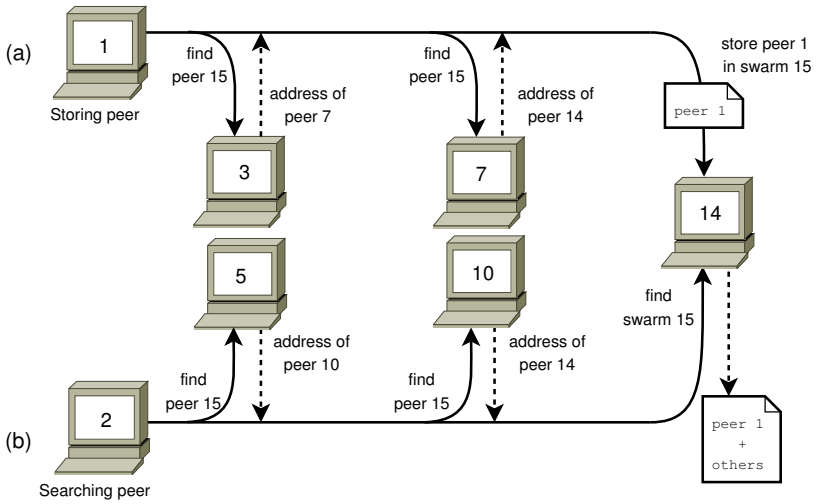

Figure 2.2: Typical store and find value actions on a swarm discovery DHT. (a) Peer 1 finds manager peer 14 through peers 3 and 7. Then it stores itself as a member of download swarm 15. (b) Peer 2 searches and retrieves the peerlist of download swarm 15 on peer 14 through peer 5 and peer 10.

of peers are found to be the responsible nodes. We refer to these peers as *swarm managers*. For all download swarms, a single DHT is needed. Peers will store their network addresses on the DHT when they have joined a swarm. Later, other peers can use the DHT to search for the peerlist of the particular swarm and find all subscribed peers.

In Figure 2.2a and Figure 2.2b, typical store and find actions in DHT swarm discovery are shown. In this example the IDs of both peers and swarms are integers. Swarm information is routed to the swarm manager peer (the peer with its ID closest to the swarm ID). In the example case, peer 14 is swarm manager for swarm 15. Peer 1 want to store itself as swarm member, and peer 2 wants to retrieve the peerlist for swarm 15. Swarm discovery starts when peer 1 has just joined download swarm 15. It then searches for a peer with the ID closest to swarm ID 15 and finds and queries peer 3. Peer 3 returns the address of the peer with the ID closest to the swarm ID that it knows: peer 7. Peer 7 will send peer 1 to peer 14, the swarm manager. Peer 1 then stores itself to the peerlist residing on peer 14.

A similar process is executed by peer 2, which is searching for the peerlist of swarm 15. It is routed (through peer 5 and peer 10) to peer 14, where it can retrieve the target peerlist. The peerlist will contain the address of peer 1.

Different BitTorrent systems use DHTs in addition to the central tracker. BitTorrent 4.1.0 introduced 'trackerless' torrents [11, 10], which uses the Kademlia DHT implementation [52] for peer discovery. Also the BitTorrent clients Azureus [5] and BitComet [8] have implemented Kademlia peer discovery solutions, both not compatible with the BitTorrent standard or each other.

Some of the other better known DHT implementations are CHORD [70], Content Addressable Networks (CAN) [66], Pastry [68], Symphony [51] and Viceroy [50]. Their theoretical performance of different actions (for instance store, find, join) are fairly similar [36] and depend on optimizations and parameter tuning [47].

DHTs are a well distributed solution to peer discovery. Their advantage is that searches for the same keys (in our case download swarm IDs) will always route to the same peers within a single network. Peers that store data and peers that search for it will both find the swarm manager to do their operation and thus be able to exchange data. Hence, DHTs give peers a high swarm coverage over a scalable architecture.

For a new peer to join a swarm discovery DHT, it has to bootstrap on a peer that is already member of the DHT network. It will then build routing tables and collect information about a part of the DHT peers. A user has to bootstrap the DHT once per session. After that, it can use the DHT swarm discovery for multiple swarm that it want information about. The initial peer has to be located using some external bootstrapping method. In the Azureus BitTorrent client, the Kademlia DHT is bootstrapped with peers received from a centralized tracker. Azureus uses its DHT not as a replacement for the centralized tracker, but merely as an addition if a centralized tracker is offline or unreliable.

Load balancing is a problem on DHTs. Information about the peers in a download swarm is stored on manager peers that have been chosen based on the hash value

of their peer IDs. These peers have no semantic relation with the swarm they are managing and are mostly no downloaders in that swarm. Every peer in the network can be chosen as the manager of some swarm that is popular, which may result in many requests. The load is unbalanced and enormous swarms can be mapped on peers with relatively poor connectivity. Another problem is related to integrity and security. In a DHT, the routing to a peer is done in some steps through other peers. The routing depends on the quality of the responses of multiple peers in the network that are not related to the requesting peer. These peers are not downloading similar files or in other ways reliable. Still the information from these peers is used to build routing tables and find peers and content. Ensuring integrity is very hard in more complex algorithms like a DHT, and ways to use DHTs for a dDoS attack have already been studied [58].

Some research has been done on the weaknesses of Kademlia. The fact that nodes can choose their own random IDs is a weakness. An attacker can use this freedom to take over the control over a certain swarm or the complete system. When the DHT nodes try to route their queries to a certain node, they will end up with an attacker and receive a fake response. Cerri et al. [15] propose a solution by constraining the ID selection of peers. Peers use a hash value of their IP address and some bits of their port number to create their ID. This ID is then challenged by sending a Kademlia *ping* message to the corresponding IP address. Furthermore, they propose a resource rotation strategy, in which the IDs of resources (in our case: download swarms) depend on temporal information. The changing location of resources in the DHT makes attacks more difficult.

Finally, DHTs lack incentives for cooperation for peers. There is no semantical or social relation between our peer and the others. If peers act like rational agents, there is no reason to donate bandwidth for the operation of the DHT.

2.3.4 Peer exchange

Peer exchange is another addition to swarm discovery using a tracker. It assumes that all peers already know a part of the swarm, for instance from a central tracker. Instead of contacting the tracker again, peers contact each other and exchange the subset of peers that they know. This obviously does not solve the bootstrapping problem, but reduces the load of the central tracker when more peers are needed during a download. Peer exchange has been implemented in Azureus versions later than 2.3.0.0 [61].

From the documentation, we can conclude that Azureus peer exchange does not provide an incentive to cooperate with the protocol. There is also no sign of load balancing, so many peers may be exchanging peers with a single peer. For integrity and security purposes, Azureus peer exchange will label peers as spammers when they are flooding the swarm with peerlists. There is, however, no protection against fake peerlists that may be exchanged.

2.4 Discussion

In Table 2.2, we give a summary of how the discussed swarm discovery solutions score in complying with the five design requirements defined in Section 2.2. All solutions can score good (+), reasonable (+/-), or poor (-).
The BitTorrent centralized tracker and multiple tracker solutions score good on bootstrapping, because their network addresses are statically embedded in torrent files. They also manage to deliver full and secure swarm discovery. The big problem is, however, their lack of scalability and load balancing which makes the tracker component a bottleneck for BitTorrent. The need for incentives is not applicable (NA) for these swarm discovery solutions, because peers do not need to cooperate as the tracker does all the work.
The DHT solution scores reasonable in bootstrapping, because it has to be bootstrapped every session. It delivers a good swarm coverage and is well scalable as there are no central components in a DHT. Still we considered scalability and load balancing only reasonable, because peers have to donate bandwidth for swarms they are not related to. The lack of incentives to cooperate and security in a DHT gives two poor scores.
Peer exchange has scores similar to the DHT solution. Bootstrapping is rated poor, because it is outside the scope of peer exchange and has to be done for each swarm. Furthermore, peer exchange offers good scalability, poor load balancing and bad security.

	Centralized tracker	Multiple trackers	DHT	Peer exchange
Bootstrapping	+	+	+/-	-
Swarm coverage	+	+	+	+
Scalability and load balancing	-	-	+/-	+/-
Incentive to cooperate	NA	NA	-	-
Integrity and security	+	+	-	-

Table 2.2: The qualities and weaknesses of current swarm discovery solutions, rated from good (+) to poor (-).

We conclude that in swarm discovery solutions there is a tension between integrity and security on one side and scalability on the other. Swarm discovery solutions that have a scalable design, like DHTs and peer exchange, suffer a lack of protection against attacks and pollution. In our swarm discovery protocol LITTLE BIRD, presented in Chapter 4, we aim to combine the best of both worlds to realize both secure and scalable swarm discovery.

Chapter 3

Swarms: a Behavioral Study

Measure what is measurable, and make measurable what is not so.

— Galileo Galilei

In Chapter 2 we have presented the shortcomings of the current swarm discovery solutions based on our design requirements. To make our swarm discovery solution meet these requirements, we have conducted a detailed study of a BitTorrent file sharing community, which we present in this chapter. We have measured the individual behavior of all the users in this community. This chapter will explain which properties of swarm members are measured and how these data can be used for the evaluation of our protocol. In Section 3.1, we introduce the general life cycle of a download swarm. In Section 3.2, we discuss related measurements on the BitTorrent network and communities. In Section 3.3, we present the structure of our measurement software and our target BitTorrent community, *filelist.org*. The results of these measurements are given in Section 3.4. Finally, in Section 3.5, we have measured the overhead of the BitTorrent centralized tracker and used Azureus statistics to estimate the overhead of the Kademlia DHT. These data show the bandwidth usage of the two most used swarm discovery solutions.

3.1 Swarm life cycle

When we measure the properties of a swarm and its members, we will always see certain phases of the swarm, namely *creation*, the *flash crowd effect*, *gradual decrease of popularity*, and finally, *death due to lack of popularity* or *deletion from the tracker*. These four phases are schematically shown in Figure 3.1 and explained below.

A swarm is created when an initial seeder creates a *torrent* file from the file it wants to share and adds this torrent to the tracker. After the addition, the initial seeder stays online to ensure data availability. The addition of new (and popular) content often makes a swarm grow rapidly directly after its creation, which is called the *flash crowd effect*. During this phase many peers join the swarm, making it the

most active phase of the swarm. Swarm members that have finished download the data, are expected to stay online and help seeding it.
The flash crowd period ends when many users that have finished downloading the data leave the swarm. The number of joining and leaving swarm members becomes balanced and the swarm size starts to fall slowly. The reducing popularity of the data will also cause the join rate to fall. The gradual decrease of popularity phase takes much longer than the fast flash crowd phase.
The life of the swarm can end in two ways. If there are no more peers left to guarantee full content availability, because all active peers lack a full copy of the file, the swarm dies. The existence of a swarm can also end when the torrent is automatically deleted from the tracker.

3.2 Related BitTorrent measurements

Several studies and measurements of the BitTorrent file sharing protocol have been published, because of its growing popularity.
There are several studies that evaluate the properties of the core algorithms of BitTorrent. Legout [46] focuses on the choking and rarest first protocols, which are part of the tit-for-tat sharing-ratio enforcement and piece distribution in BitTorrent. Pouwelse et al. [63] have focused their measurement on the content that is distributed through popular BitTorrent communities and the behavior of the users. Izal et al. [41] measured the statistics of a single torrent over a period of five months. The amount of cooperation of different BitTorrent communities has been measured by Andrade et al. [4], to test if the tit-for-tat protocol increases the amount of cooperation. Qiu et al. [65] have modeled the behavior of BitTorrent users mathematically and compared their results with actual measurements.

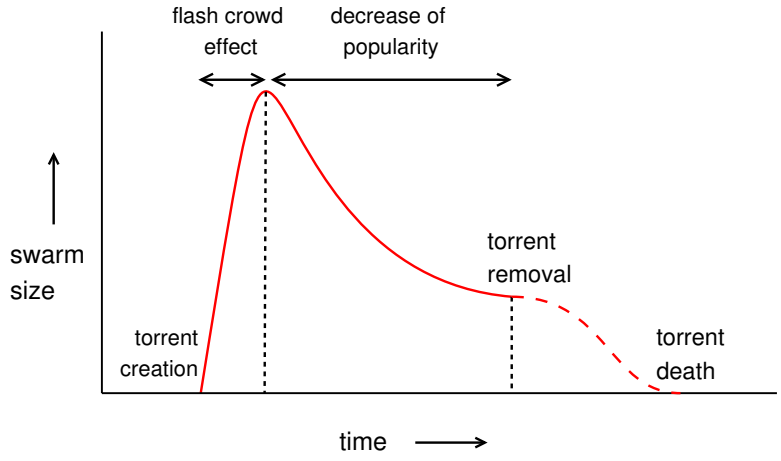

Figure 3.1: The life cycle of a download swarm.

To our knowledge, these measurement studies all use the IP address or BitTorrent id of a peer to identify it. P2P activity from the same IP address, however, does not have to be invoked by the same user, because people may have dynamic IP addresses or may be located behind a *network address translator* (NAT) box. The BitTorrent peer identification does not solve the problem of peer identification either. According to the BitTorrent specification, a peer can randomly regenerate this id for each download swarm. Hence, all these measurement studies suffer from the problem of identifying a peer over multiple sessions and in different download swarms.
We differ from the studies above in that our measurements are based on unambiguous peer identification that is valid through multiple swarms and sessions of a peer. Identification of peers is done by a tracker that relates all peer requests to existing user accounts. This enabled us to map measured statistics on users. When one user was in different swarms on different times, these statistics still could be combined. Hence, we have collected an individual network activity history for each user. With these additional data we can extract more sophisticated statistics from our measurements than the studies mentioned above.

3.3 Measurement setup

In this section we introduce the Filelist BitTorrent community as the community where we conducted our measurements, and we describe the setup of our measurement software.

3.3.1 BitTorrent community

We were looking for a BitTorrent community that would resemble the Tribler community in the near future. With measurements of such a community we can design, optimize and evaluate our swarm discovery protocol for Tribler. The social network and sharing-ratio enforcement features of Tribler will influence its community to be active and cooperative. Therefore, our target BitTorrent community should be active and have sharing-ratio enforcement and disclosed user behavior statistics. The *filelist* BitTorrent community [28], further referred to as filelist, meets these requirements and was chosen to do our measurements on.
Filelist is a BitTorrent community and tracker. People have to become a member to be able to seed or download files through filelist. Currently, filelist has around 110,000 members. These membership accounts are primarily used to execute extra strict sharing-ratio enforcement, thereby stimulating the members to cooperate more intensively.
The normal BitTorrent clients use the Tit-for-Tat protocol [20] for sharing-ratio enforcement (see Section 1.2). This protocol is not very strict and sharing-ratios are reset whenever a BitTorrent client is shut-down. In filelist, additional sharing-ratio enforcement is handled in the tracker. This creates the possibility to save

member statistics for their complete lifetime. Members that have too low sharing-ratios are punished by refusing them access to the newest content. Because of these strict rules, members have an extra incentive to upload. Hence, filelist tends to be a BitTorrent community with more cooperation and content availability.
Strict sharing-ratio enforcement increases the likeliness of *Sybil attacks* in which people create a new identity when their former membership account has received a negative reputation. In the filelist community this problem does not exist, because new members with an upload amount under a certain threshold face the same download restrictions as freeriders. Also, filelist has a user account limit, which has been reached for some time now.
The administration of users data in the tracker of filelist has the advantage that we can measure detailed statistics about all users in all swarms of the community. These statistics are presented in Section 3.4. The user level administration gave us the opportunity to link different sessions of the same user and create individual activity histories.
Apart from the basic BitTorrent tracker functionality, the filelist community consists of a community website where members can search and discuss the hosted files. The community offers social features for its members, like a forum, the possibility to register members as your friends, and send messages to each other, to increase the social relations between members. An increased social community feeling will increase altruism and content availability on the filelist file sharing network.
We showed the lack of scalability and reliability of the BitTorrent central tracker in Section 2.3. Filelist has chosen to remove all torrents 28 days after their creation to reduce tracker bandwidth. The amount of available content on the tracker is thereby significantly decreased, because of the limitations of the BitTorrent system.

3.3.2 Measurement software

Filelist gives detailed statistics about each of its swarms and users. We wrote software that continuously downloads, compresses and stores the statistics from the filelist website, a process called *web scraping*. By combining our scraped data, we

GENRE	RELEASE						
XVID	[redacted] 2006-01-17 11:16:03	8	672 hours	714.69 MB	5 times	3	76
XVID	[redacted] 2006-01-17 11:00:52	8	672 hours	1.40 GB	0 times	1	155
XVID	[redacted] 2006-01-17 09:22:28	8	670 hours	715.35 MB	51 times	42	60
TVEPS	[redacted] 2006-01-17 08:35:46	5	669 hours	181.75 MB	59 times	39	21
TVEPS	[redacted] 2006-01-17 08:33:14	0	669 hours	183.14 MB	5 times	3	1

Figure 3.2: The first five torrents in the torrent list of the *filelist* community.

can add a time factor into the filelist statistics, creating individual online activity histories for each user. In this section we describe our web scraping software and the collected data.

There are two sorts of statistical pages in Filelist: the *torrent list*, and for each active swarm a *swarm member list*. We will describe these pages in turn. The torrent list shows all currently downloadable torrents (active swarms) and their properties, see Figure 3.2. For each swarm, the torrent list shows the *genre, torrent name, number of comments, hours until deletion, file size, number of view, number of seeders*, and *number of leechers*. Because around 25 new torrents are added to filelist every day, and the torrents are deleted after 28 days, there are about 700 torrents in the torrent list.

Per download swarm, a swarm members list is given, with details about the connection and behavior of each swarm member, see Figure 3.3. For seeders and leechers, the swarm member lists shows the *username, connectivity, upload and download statistics, sharing-ratio, percentage downloaded, tracker connection statistics*, and *the name of the BitTorrent client*. Some filelist members use the swarm members lists to manually select or ban users based on their long term cooperativeness and activity.

We have built our web scraping software using multi-threading to guarantee reasonable frequent download of all pages. The program's main thread downloads the torrent lists to acquire knowledge over all available swarms and their sizes. The torrent list is downloaded, compressed and stored at a fixed interval of 200 seconds. After each download of the torrent list, its active swarms are divided over four download threads. These threads will scrape the swarm member lists of all received swarms. Larger swarms with larger swarm member lists take a longer time to download. To guarantee similar download frequencies for all the swarms, we calculate the expected download size of each swarm member list and distribute the swarms over the download threads in such a way, that each thread receives a similar amount of bytes to be downloaded. When a new swarm is detected by a download thread, it downloads and compresses the torrent meta-file in addition to the swarm member list.

With the use of four download threads, download frequency of the swarm member lists was around every 6 minutes. In this configuration our web scraper used a bandwidth of 220 MBytes per day for scraping torrent lists and 8.5 GBytes per day

Seeders [Hide list] — 337 Seeder(s)

User/IP	Connectable	Uploaded	Rate	Downloaded	Rate	Ratio	Complete	Connected	Idle	Client
Arbo	Yes	10.36 GB	904.54 kB/s	362.95 MB	464.00 kB/s	29.226	100.00%	3:40:32	20:23	BitTorrent/4.1.2
revox	No	6.29 GB	544.16 kB/s	359.35 MB	360.05 kB/s	17.929	100.00%	3:36:01	13:57	uTorrent/1400
Krillle	Yes	6.18 GB	912.74 kB/s	360.85 MB	218.00 kB/s	17.535	100.00%	2:14:58	16:39	uTorrent/1400
SlayTG	Yes	5.97 GB	1.16 MB/s	359.35 MB	225.20 kB/s	17.001	100.00%	1:43:51	16:27	uTorrent/1400
tregie	Yes	3.72 GB	327.68 kB/s	359.35 MB	435.47 kB/s	10.596	100.00%	3:34:18	15:59	uTorrent/1400
rylin	No	2.89 GB	254.32 kB/s	359.35 MB	459.96 kB/s	8.241	100.00%	3:39:24	20:39	uTorrent/1220
swither	Yes	2.88 GB	337.65 kB/s	359.52 MB	228.24 kB/s	8.213	100.00%	2:35:40	6:25	Azureus/2.3.0.6

Figure 3.3: The first seven seeders in a typical swarm member list on the *filelist* community.

for scraping swarm member pages and downloading torrent files.
Our web scraper has been running from 14 December 2005 until 4 April 2006, when filelist changed the setup of their website and removed the online swarm member lists. We have collected a total of 80 GBytes of compressed data, which contains statistics about the members of over 4,000 download swarms. In Appendix B.1.2, we give the details of where and in what format our trace data is stored.

3.4 Filelist.org measurement results

From our collected trace data, we have extracted statistics to give insight into the behavior of member of an active social file sharing community. From all swarms, we have selected one large example swarm to show these statistics. The swarm was active 3–30 January 2006, 8,963 different users have been bartering in it, and the file that was bartered was a movie with a size of 730 MBytes. We miss the data of some periods in the life time of the example swarm, due to technical difficulties of the filelist tracker and the web scraping software. This has caused the interruptions in the figures.
In this section, we will present measurements of swarm size, churn and peer behavior for the example swarm. Furthermore, we will present the online probability

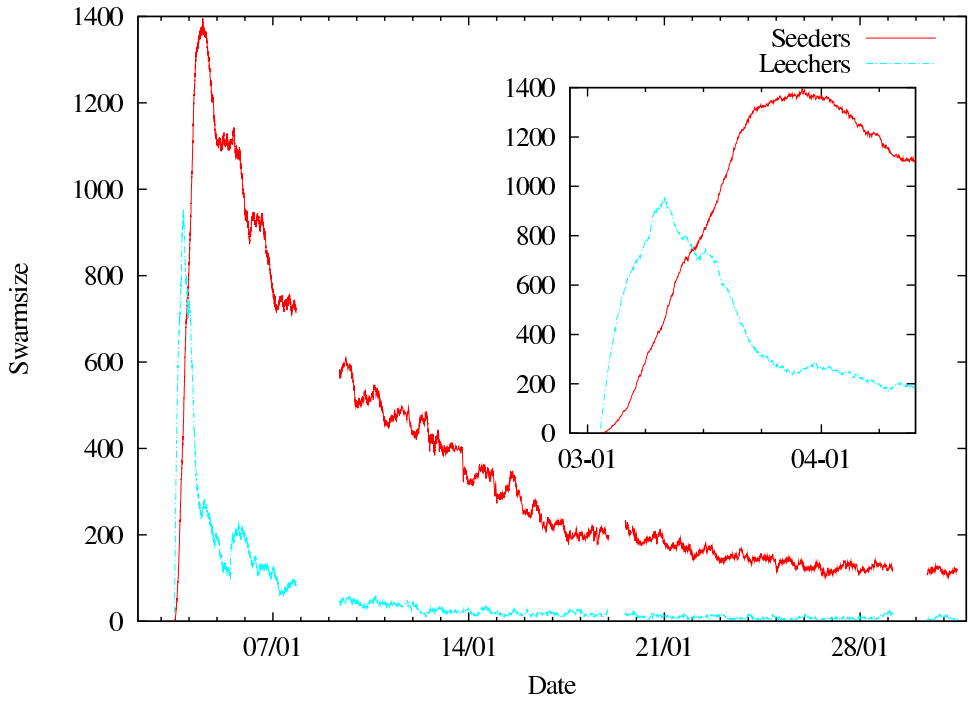

Figure 3.4: The number of seeders and leechers in our example swarm over a period of 28 days.

and session length for the users in all swarms of the filelist community.

3.4.1 Swarm size and seeders/leechers ratio

Figure 3.4 shows the measured data of the life cycle our example swarm. The number of seeders and leechers are shown during the active period of the swarm.

The first five hours after creation the swarm consists of almost only leechers, because no one has had the time to complete the download. This situation changes after 10 hours, when there are more seeders than leechers. Until deletion, after 28 days, about 90% of the swarm consists of seeders, which creates a swarm that gives fast downloading speeds for the new downloaders. We have found this seeders/leechers ratio in many of the swarms on filelist. In other BitTorrent communities without additional sharing-ratio enforcement, the majority of swarms often consists of leechers. For instance, in the Mininova.org community, only 40.6% of the 4.6 million swarm members are seeders (on 6 October 2006). This difference shows that the strict long-term sharing-ratio enforcement of the filelist community stimulates seeders to stay online.

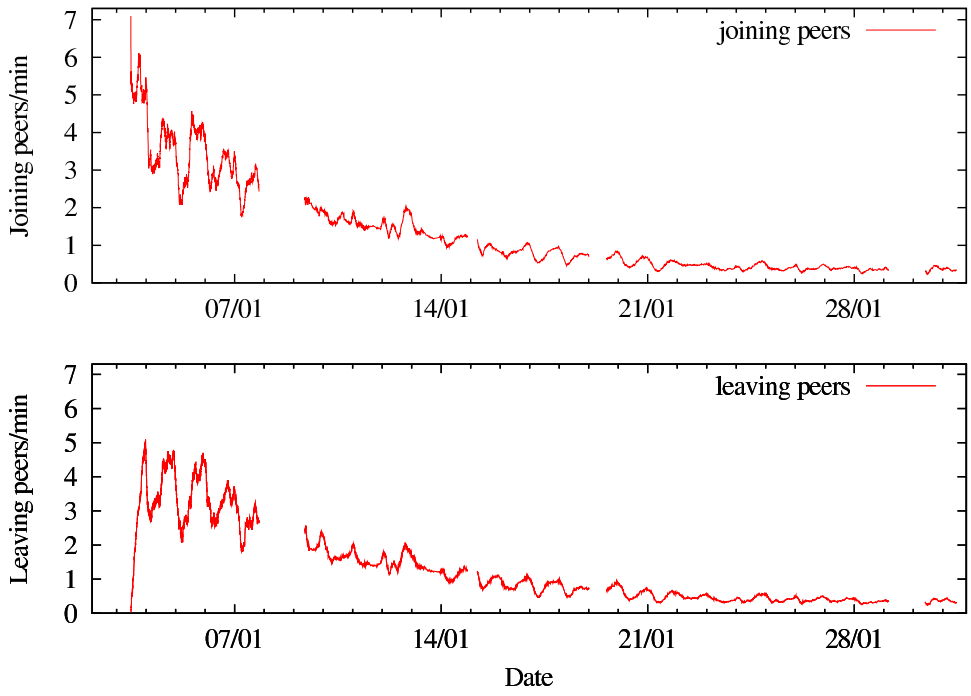

Figure 3.5: The churn of a swarm: the moving average (over 90 minutes) of the number of joining and leaving peers per minute.

3.4.2 Churn

The *churn* of a download swarm is defined as the number of joins in the swarm and leaves from the swarm per minute. This is an important property when designing a swarm discovery solution, because it influences how fast a peerlist (containing all swarm peers) becomes invalid (because of a leaving peer) or incomplete (because of a joining peer).
In Figure 3.5 the churn of our example swarm is shown. We have plotted a moving average over 90 minutes of the churn data to smooth out the plot. During the creation and flash crowd phase of the swarm many downloaders join the swarm and few leave. This explains the rapid growth in the swarm during the first 10 hours. After this period, the churn stabilizes and the join rate falls just under the leave rate making the swarm size decrease slowly. Both the join and leave rate scale down with the falling swarm size.
The figure shows a certain cycle in which the join and leave rate increase and decrease. This is due to the part of the day or night in which filelist users are more and less active (also reported in [4]). This cycle has a much higher amplitude in the first five days of the swarm's lifetime, revealing more peer activity. The day-night cycle can also be noticed in Figure 3.4.

3.4.3 Peer behavior

With the statistics from the swarm member lists we have created an individual peer behavior history for each peer in each swarm. Such a history lets us follow all peer properties mentioned in Section 3.3.2 over time during the sessions of a peer.
In Figure 3.6, we show the download history of a typical peer from our example swarm. This figure includes the percentage of the file that the peer has completed, its sharing-ratio (explained in Section 1.2), and its upload and download speed. The figure shows that this peer profits from the large number of seeders in the example swarm and is able to download the complete file in a short time. The peer stays online after the completion and seeds the file, until its sharing-ratio rises above a value of 1.0.
We will use these individual peer behavior histories for the trace-based emulation of our swarm discovery protocol, which will be presented in Chapter 5. During our emulations, the measured behavior of all peers in a swarm is re-enacted to test our swarm discovery in a realistic setting.

3.4.4 Online probability and online length

We want to know the probability that a peer, who is online on a certain moment, is still online on a later moment. This is important for the bootstrapping phase of swarm discovery, described in Section 4.5.1. Therefore, we have measured which users were online in the complete Filelist community at every hour over the measurement period.

We choose an initial online user group consisting of 15,000 users on 16 January 2006.Then we measured how many users of the initial group were online over the following 10 days. The days following 16 January were chosen, because we have uninterrupted trace data for this period. Figure 3.7 shows that during our measurement period, always at least 48% of the users from the initial group were online. A significant number of users also was continuously online since the start of our measurement. Note that Figure 3.7 shows the active users in the complete filelist community and not only in a single swarm. These data show that within an active community like Filelist, their is a high probability to find users online again, that were online before. This is an important fact, if these users are the sources for swarm discovery.

A typical swarm discovery situation is that peer A leaves a swarm, but has stored for instance the network addresses of 100 peers that were bartering on that moment. Two days later, peer B wants to discovery that swarm. During swarm discovery bootstrapping, it finds peer A as an initial peer related to the swarm. Therefore, peer B sends a request to peer A, and receives the out-dated peerlist of the swarm, containing the 100 peers that were online two days before. According to Figure 3.7,

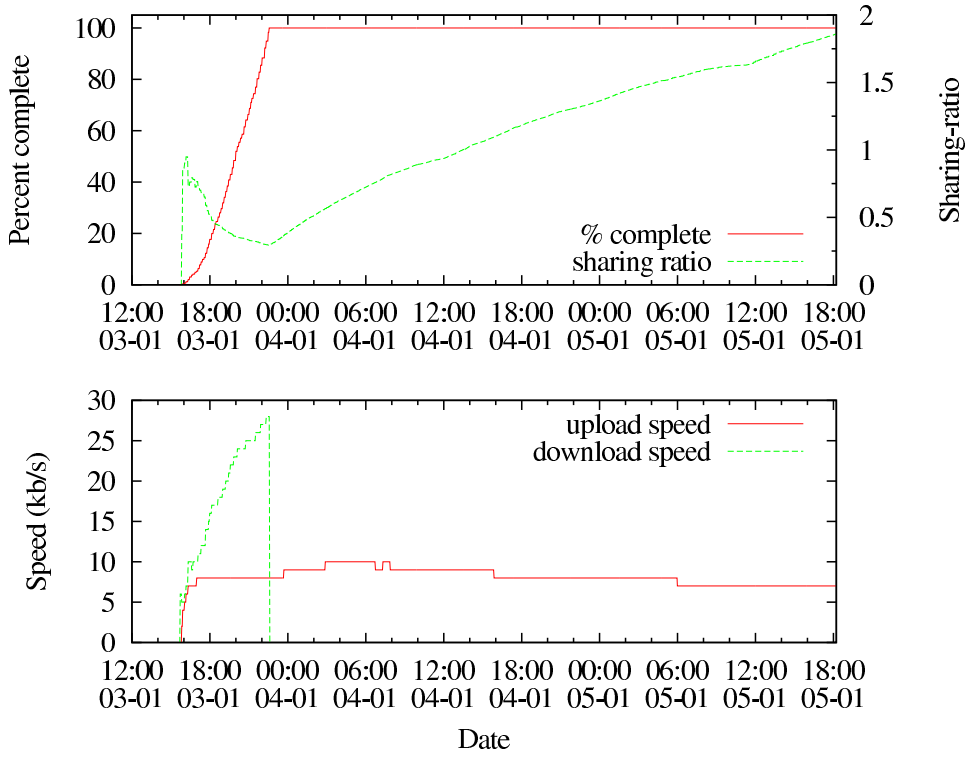

Figure 3.6: An example download history of a peer: The percentage of the file that this peer has downloaded and its sharing-ratio (top), and the upload and download speed of the peer during this period (bottom).

probably 50 of those peers are still online. Those online peers are either still active in the swarm, or can give peer B additional help. When the online probability is lower, old views of swarms or the community have less value.

The effect that it is likely to meet peers again can also be seen in Figure 4.8 in Chapter 4. There we show that many of the members of a swarm will be online in other swarms in the days after they have left the swarm.

As a final prove that users are likely to be online, we also measured the distribution of the session lengths of peers over the complete trace data. A session is the period that a peer is online in the community. This measurement includes 2.3 million sessions of over 91,000 distinct community members. Figure 3.8 shows the percentage of sessions that are longer than a certain time. From the figure can be concluded that 10% of the sessions of peers are longer than one day and three percent are longer than two days. The small discontinuities in the plot are caused by interruptions in our measurements.

We can conclude that although swarms and file sharing communities consist of a dynamical group of peers, still it is likely to encounter the same active peers in multiple sessions.

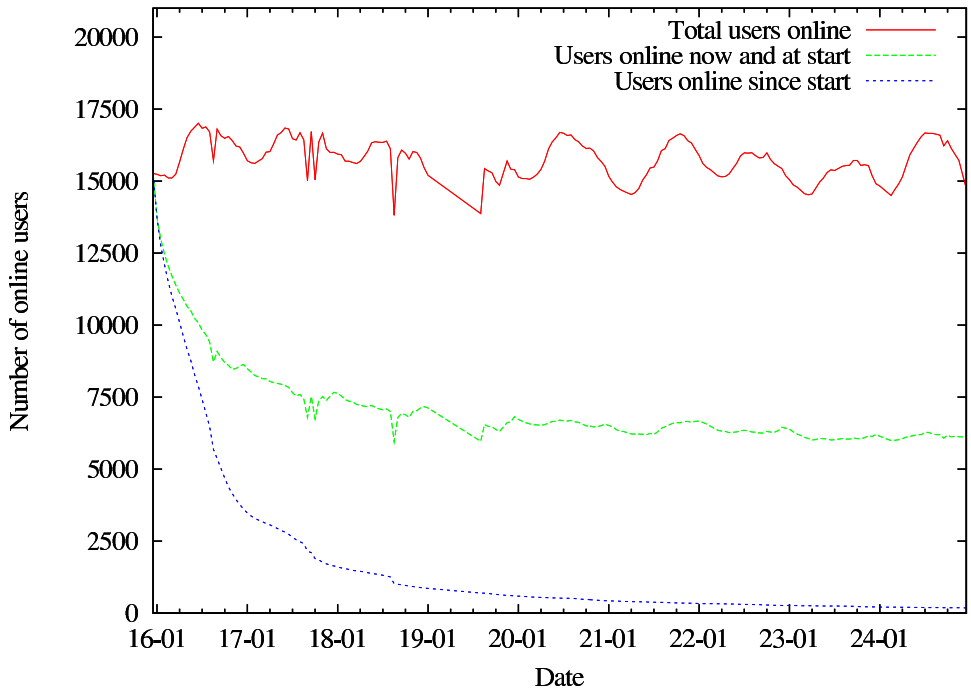

Figure 3.7: The total number of peers online, the number of peers that were online on 16 January 2006 and at the date on the horizontal axis, and the number of users continuously online since 16 January 2006.

3.5 Overhead of Swarm Discovery Solutions

In this section we will measure the bandwidth usage of the most used existing swarm discovery solutions, namely the central BitTorrent tracker and the Kademlia DHT as implemented in the Azureus BitTorrent client. We will compare their overhead with the overhead of our own implementation in Section 5.4.3.

3.5.1 Central BitTorrent tracker

The bandwidth usage of swarm discovery through the centralized BitTorrent tracker depends on its announce frequency, peerlist length and whether or not a peer uses compact format to receive peer information from the tracker. Announce frequency is defined as how long a peer waits between announcing itself at the tracker. Normally, the tracker decides how frequently peers can connect to it, to reduce tracker bandwidth and server load. Periods from 10–30 minutes are usual. Announcing more often than advised by a tracker is considered malpractice, but can give peers faster downloads.
After an announce, the tracker will return a peerlist, with the ip addresses, ports and peer IDs of peers in the swarm. Default tracker settings are to send peerlists of 50 peers (when available). The peerlist can be sent in *b-encoded* format (around 73 bytes per peer) or the more bandwidth efficient *compact format* (6 bytes per

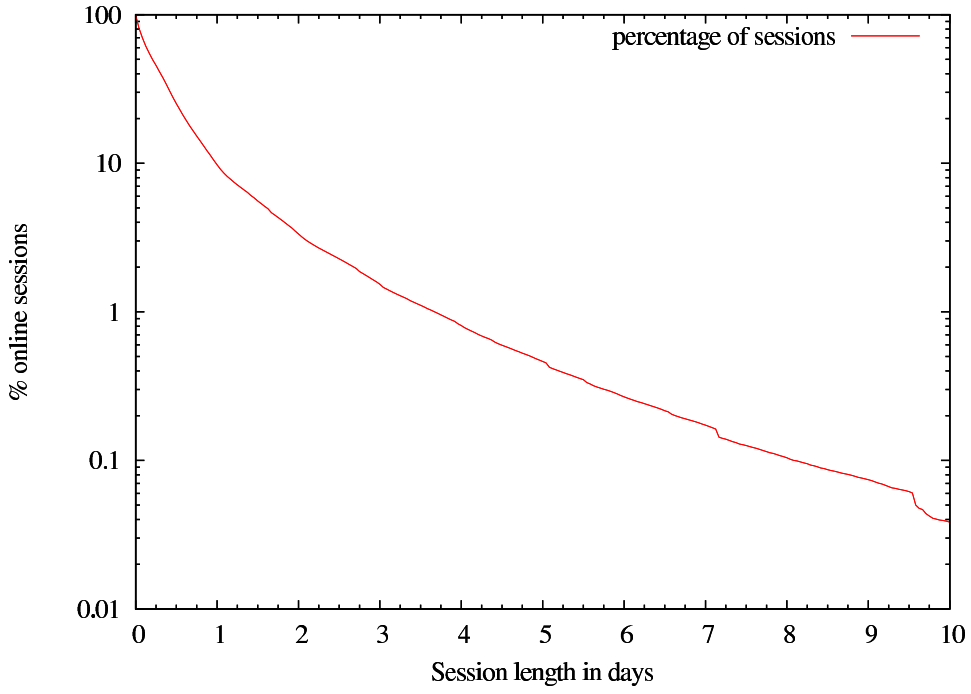

Figure 3.8: The percentage of sessions that are *longer* than a certain number of hours in the filelist.org community. Percentages are shown on a logarithmic scale.

peer). Table 3.1 shows the effective bandwidth usage of a centralized tracker with different announce frequencies and peerlist formats. The bandwidth usage is very low for each client, but for a tracker handling many big swarms, this can still create high bandwidth usage.

Announce frequency	Bandwidth usage (bytes/min)	
	b-encoded format	Compact format
10 minutes	406	76
20 minutes	203	38
30 minutes	136	25

Table 3.1: The bandwidth usage of the BitTorrent tracker communication.

3.5.2 Distributed hash tables

In this section we discuss the bandwidth usage of the distributed swarm discovery protocol of the Azureus BitTorrent client, implemented by the Kademlia DHT (see Section 2.3.3). Hence we are able to compare its bandwidth and efficiency with our own swarm discovery protocol in Section 5.4.3. These statistics are derived from the Azureus client, which has a statistics screen with detailed information about the operation of Kademlia. Table 3.2 shows our derived statistics after a connection with Kademlia for a period of 64 minutes on a fully connective computer with a public IP address.

Statistic	Sent	Received
Total packets	9494	2981
Packets/minute	148.3	46.6
Bandwidth (KB/minute)	12.10	4.75
Successful packets	1478 (15.6%)	
Successful FIND VALUE packets	0	69
Successful STORE VALUE packets	10	18

Table 3.2: The statistics of Kademlia DHT protocol usage for 64 minutes.

Most remarkable is the fact that only 15.6% of all outgoing DHT requests is successful. Furthermore, our results show that in our case, no actual peer information was found through the DHT in the measurement period, because no successful FIND VALUE requests where sent. This is in accordance with the fact that Azureus only starts using the DHT when a central tracker is unresponsive, which was not the case during our test. Other peers using the DHT have however managed to use our peer by storing and retrieving peer information by sending our peer STORE VALUE and FIND VALUE requests. The bandwidth usage for the DHT solution is around 100 times more than the central tracker bandwidth usage. It will increase when the DHT is actively used as swarm discovery solution, instead of only as a

backup. Still, it is a reasonable overhead when peers are bartering large amounts of data.

Chapter 4

A Decentralized Swarm Discovery Protocol

Design is not just what it looks like and feels like. Design is how it works.

— Steve Jobs

In this chapter we present our decentralized swarm discovery protocol, called LITTLE BIRD[1]. LITTLE BIRD is an epidemic protocol that exchanges swarm information with other peers in the swarm. Epidemic algorithms are recognized to be robust and scalable means to disseminate information in large networks. LITTLE BIRD cooperates with the BuddyCast protocol (presented in Section 1.3) to bootstrap initial swarm discovery.

In Section 2.4, we concluded that there is a tension between security and scalability in existing swarm discovery solutions. The solutions that have removed central components, like DHTs and peer exchange, lack a good protection against peers that try to misuse those protocols. In the design of LITTLE BIRD, we aim to combine scalability with resilience against attacks of malicious peers.

To add the security aspect to our scalable protocol, we have designed a quantitative measure of the contribution of a peer. When peers do not cooperate or even try to misuse the protocol, their level of contribution is reduced and these peers are omitted in further swarm discovery communication.

Because LITTLE BIRD combines scalability with incentives and security, we believe it can replace the current swarm discovery solutions. As we have implemented the LITTLE BIRD protocol for the Tribler network, we will refer to peers that support this protocol as Tribler peers.

In Section 4.1, the social-based design of the protocol is presented. We will present the theory behind epidemic data dissemination in Section 4.2. The components and the software architecture of LITTLE BIRD are discussed in Section 4.3. In

[1] Our protocol uses gossip dissemination to spread swarm information. When people are gossiping, they tend to say *A little bird told me...* when they do not want to give the source of the gossip. Therefore, we call our protocol after this little bird that is a source of gossip for many.

Section 4.4, we give our definition of the level of contribution of a peer and explain the indicators that we use to measure it. In Section 4.5 we describe how LITTLE BIRD meets the design requirements from Section 2.2 and present additional design details of LITTLE BIRD.

4.1 Social-based protocol

The overall design goal for LITTLE BIRD was to develop a social-based swarm discovery protocol, instead of a purely mathematical approach. In the DHT implementation of swarm discovery, choices are made based on mathematical structures. For instance, the choice of which peer is obliged to play the role of a tracker for a certain swarm is based on a mathematical measure of distance. The outcome of such a mathematical distance function often does not match the interest of the peer and therefore does not offer incentives to cooperate with the swarm discovery protocol.

In LITTLE BIRD, swarm discovery requests will be send to peers with interest in the particular swarm, namely peers active in the same download swarm, peers with a similar download history, and peers with similar content taste. The peers that manage the peerlist of a swarm, so that other peers can use them as a source of swarm discovery, are also peers that are downloading (or have recently downloaded) in that same swarm. All these peers have a semantical relation with the content that they make available, which makes them motivated to contribute to the protocol.

This design strategy lets LITTLE BIRD work like a social community. People solve their problems by cooperating with others that share mutual interests. The relations that are created are valuable to solve future problems. In LITTLE BIRD, peers with the similar download interests cooperate to make their favorite content available to each other through swarm discovery. These peers automatically meet new peers with interests in the content. When peers find each other to be contributive, they will prefer connections with each other during future bartering.

4.2 Epidemic information dissemination

Research into epidemic information dissemination [26] has been inspired by research on real epidemic diseases and their infection models. The transmission of an infection from one person to a group of other people is modeled by a computer that forwards a message to a set of other computers. Hence, information is disseminated reliably in the same way an epidemic would propagate within a population. This can be compared with the propagation of a rumor among people and is therefore also called *gossip dissemination*.

Using the behavior of epidemics, one can create application-level multicast in which each peer in a network can distribute a message to all other peers. This distribution is initiated by creating a message and sending it to a limited number of

peers. Upon reception of a message, a peer will forward it to f other peers, known as the *fanout*. Each period in which all peers forward a message is called a round. There are two basic models for epidemic data dissemination, namely the *infected forever* and *infect-and-die* model. In the infected forever model, peers that have received a message will permanently store it and forward it every round to f other peers. In the infect-and-die model, the message is only forwarded once, after which the peer erases the information (it dies).
When we use the epidemic protocol for swarm discovery, received swarm information is stored permanently, so we will focus on the infected forever model. This model has the fastest information dissemination. In the *infected forever* model, assuming that infectious peers try to contaminate f other peers in each round with a population size n, one has the following approximate formula for the expected fraction Y_r of infected members after r rounds [6]:

$$Y_r \approx \frac{1}{1 + ne^{-fr}}. \tag{4.1}$$

Equation 4.1 shows that epidemic information dissemination realizes exponential decrease of the fraction of peers that have not received a message.
Additional advantages of epidemic protocols are their scalability and robustness. The protocol is scalable because it can be implemented with a static fanout. Hence, the bandwidth overhead per peer is bounded. Redundancy of messages makes the system very robust against communication errors between peers, which will do no substantial harm to the fraction of peers that receive a message.
In LITTLE BIRD, the communication of swarm information through the swarm is designed as an epidemic protocol, but we eliminated message-forwarding from the protocol. Instead of forwarding new swarm information to other peers, these other peers have to take the initiative of sending a request to collect the new information. After the requesters have stored the new information, other peers can request them, so that the information spreads through the swarm. This approach preserves the scalability of epidemic data dissemination, while it adds additional security to the protocol. Peers can no longer flood the network with pollution. This is an important design decision from a security perspective.
This model assumes that there is a constant and limited population that is infected at each round. In swarm discovery, we are dealing with a dynamic population of joining and leaving peers. In the analogy of the model, this can be seen as infected people that disappear and are replaced by healthy individuals. Clearly, this will make the process of spreading swarm information to most swarm members more difficult and swarm coverage will be lower in practice than predicted in the theoretical model.

4.3 Architecture of LITTLE BIRD

The LITTLE BIRD software architecture consists of three components, namely the *peer acquisition*, *swarm database*, and *peer selection* component. These three

components are used to receive peerlists from other LITTLE BIRD-supporting peers, store the new discovered peers in a local database and select contributing peers for bartering. We have also created a graphical user interface to give the user more insight in the swarm discovery mechanism and its overhead. The remainder of this section will describe the three LITTLE BIRD components and our user interface in detail.

4.3.1 Peer acquisition

The peer acquisition component is responsible for the transfer of requests to and the reception of responses from other peers in the swarm. Peers use these messages to spread swarm discovery information information through the swarm. Our protocol uses two types of messages for this purpose. The message containing a request for new peers is called a *getPeers* message. A Tribler peer responds to a getPeers message with a *peerlist* message, containing a list of peer addresses and statistics. A detailed description of message formats can be found in Appendix A.3. A complete LITTLE BIRD request and respond cycle consists of five steps, which will be explained in turn.

Tribler peer selection (Figure 4.1a) The selection of peers to send a request to is based on the level of contribution (described in Section 4.4) of the known Tribler peers. Furthermore, the connection interval of peers has to be checked, so that peers can not be overloaded by getPeers requests. Finally, one or more Tribler peers are selected to send a getPeers message to.

Transfer of getPeers message (Figure 4.1b) A getPeers message is created and sent to the selected Tribler peer. In each getPeers message, a Bloom filter is in-

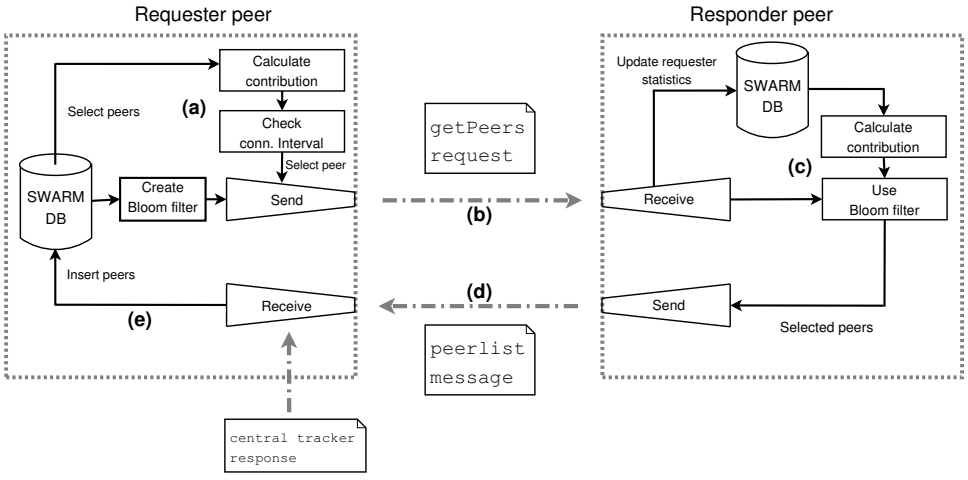

Figure 4.1: Schematic overview of the LITTLE BIRD peer acquisition of a requester peer and the help of a responder peer in five steps.

cluded that represents the currently known swarm peers of the requester, so that these peers are not received again (see Section 4.5.3).

Creation of peerlist (Figure 4.1c) The responder peer responds to incoming getPeers requests by sending a peerlist message to the requester. The peer will make a selection of the peers in its swarm database with the highest level of contribution, excluding the peers that the requester already knew and has sent in the bloom filter. A responder peer can choose not to respond to a getPeers request when it is overloaded by messages.

Transfer of peerlist message (Figure 4.1d) The peerlist is transfered back to the requester peer.

Storage of peers (Figure 4.1e) The peer information from the received peerlist message is stored in the swarm database.

We have formatted our peerlist message similar to the peerlist returned by a central BitTorrent tracker. Therefore, also responses from central trackers can be received and stored by the peer acquisition component.

4.3.2 Swarm database

Each Tribler application has a local database manager, which is used as a knowledge base for Tribler features. We have added a database, called *swarm database*, for the storage of peers in download swarms. The swarm database plays a central role in our swarm discovery design. The local view of a peer on a swarm is modeled from the statistics of all known peers, which are stored in the swarm database. Standard BitTorrent does not include such a local view on the swarm. Instead, BitTorrent discards network addresses received from the BitTorrent tracker after a connection attempt. With the addition of the swarm database we enable our client to literally learn from the past.
When a successful getPeers request has been transmitted, the resulting peer information from the received peerlist message is stored in the swarm database. These data are updated with statistics about connection results and bartering activity, extracted from the barter engine. The details about the statistics in the swarm database are presented in Appendix A.4. The peer statistics from the swarm database are used by the peer selection component to calculate the contribution of a peer (see Section 4.4). This level of contribution influences which peers are connected for bartering or swarm discovery and which peers are inserted in a peerlist message.
We have implemented a simple removal policy for the information in the swarm database. Swarm information is deleted after it has not been accessed for a certain period. This means that a peer has not used the local information itself and no request for it was done by other peers. More sophisticated removal policies can be applied if the size of the swarm database becomes significant.

4.3.3 Peer selection

Peer selection is the process of deciding to which peers (received through swarm discovery) a bartering connection will be attempted. The standard BitTorrent system does not contain a peer selection component. After periodical requests to the central tracker, connections will be made to all received peers. The connections are kept until one of the peers closes them, for instance when the Tit-for-Tat protocol or a lack of data does not allow data exchange (see Figure 4.3 top).
LITTLE BIRD has a more sophisticated approach to peer selection. Acquired peers are not directly connected, but stored in the swarm database. The peer selection component will choose which peers to select for bartering, based on the new and older peers in the swarm database. Hence, old acquired peers can be preferred over newer peers with a lower level of contribution. The selection process is depicted in Figure 4.2. The connection statistics, which for instance include if connection attempts have succeeded, are fed back into the swarm database, so that the level of contribution can be updated.

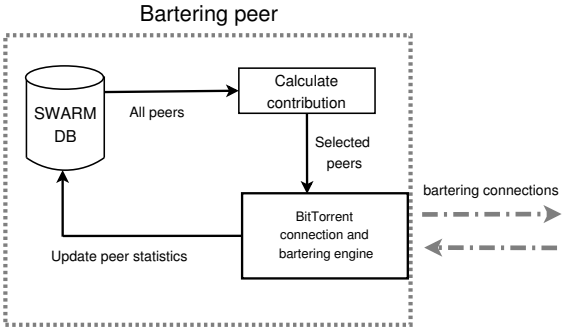

Figure 4.2: Schematic overview of the LITTLE BIRD peer selection component.

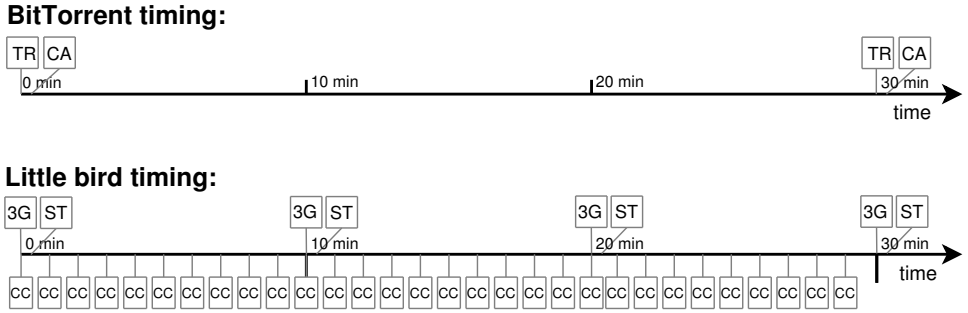

Figure 4.3: BitTorrent (top) only does a tracker requests (TR) approximately every 30 minutes and then connects to all received peers (CA). LITTLE BIRD (bottom) acquires peers by sending three getPeers requests (3G) every 10 minutes and stores the resulting peers (ST). Every minute connections are made to the most contributive peers (CC).

Figure 4.3(bottom) shows the timing of peer acquisition and selection in LITTLE BIRD. Above the time line the peer acquisition is depicted, which will send three getPeers requests every 10 minutes and store the acquired peers. Below the time line, we have shown that the peer selection component performs a peer selection step every minute. During a selection step, this component selects a set of peers and attempts to connect to them.

The actual selection criteria are based on the contribution of peers and protection against dDoS attacks, which will be explained in Section 4.4 and Section 4.5.5 respectively.

4.3.4 Graphical user interface

Swarm discovery is a part of a P2P program that works in the background. In order to give the user more insight in the state and overhead of swarm discovery, we have created a graphic user interface for LITTLE BIRD. For each swarm that a user is active in, Tribler can show a *Torrent Details* window, with different kinds of information about the selected torrent and swarm. To this window, we have added an additional tab called *Swarm Discovery Info*. The tab is shown in Figure 4.4 and gives properties of both the distributed and centralized tracker. Statistics include the bandwidth overhead of incoming and outgoing requests, the number of peer with Tribler and LITTLE BIRD support, and timing information of the requests.

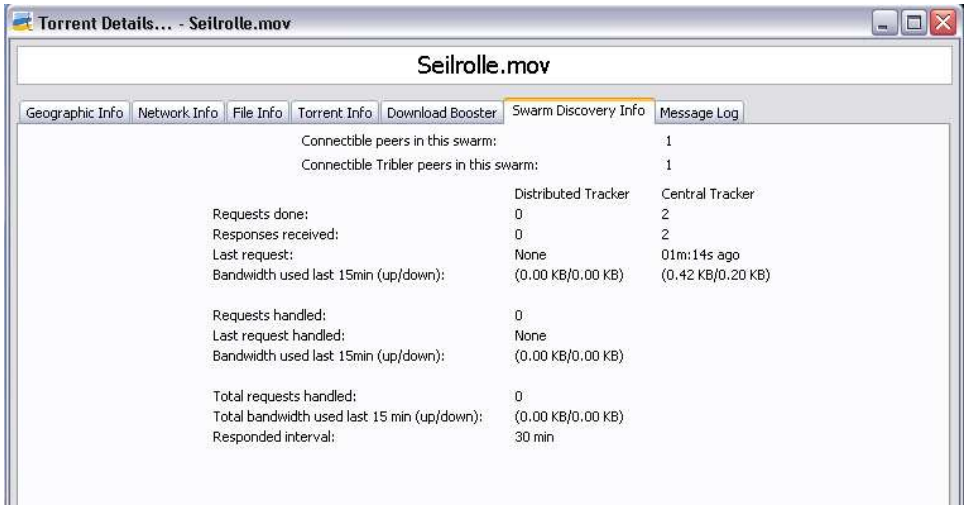

Figure 4.4: The Swarm discovery statistics window in Tribler gives information about swarm discovery by LITTLE BIRD and the BitTorrent tracker.

4.4 Contribution of a peer

The contribution of a peer is defined as the extent to which a peer cooperates with the BitTorrent file sharing protocol and the LITTLE BIRD swarm discovery protocol (if supported). Inconnective peers and peers that try to pollute the protocol will for instance have a contribution measure of 0.0, while a seeding Tribler peer with high bandwidth can have a contribution of 1.0.
We calculate the quantitative contribution measure for each peer in each swarm from its connection statistics in the swarm database. In the LITTLE BIRD protocol, this contribution measure is used for peer selection in different parts of the system, as can be seen in Figures 4.1 and 4.2. For instance, only the most contributive peers will be connected for bartering and only contributive Tribler peers will be used as a source for swarm discovery. A peer that is known in multiple download swarms will have a separate level of contribution for each swarm. The level of contribution will fall for a swarm that it leaves and be high for another swarm where it is still actively bartering.
The contribution $C(p)$ of a peer p with a connection history is defined as a weighted sum of the following four indicators:

1. Connectivity $C_c(p)$, weight 0.2
2. Bartering activity $C_b(p)$, weight 0.5
3. Swarm discovery activity $C_a(p)$, weight 0.1 (if p supports the LITTLE BIRD protocol)
4. Swarm discovery quality $C_q(p)$, weight 0.2 (if p supports the LITTLE BIRD protocol)

All four indicators are normalized to have a range of $[0, 1]$. The swarm discovery activity and quality measures are only calculated for peers that support the LITTLE BIRD protocol, so that the contribution measure C has a range $[0, 0.7]$ for BitTorrent

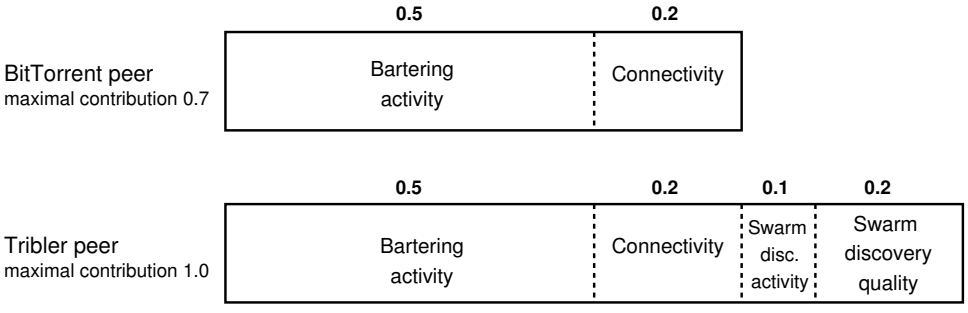

Figure 4.5: The quantitative contribution of BitTorrent peers and Tribler peers. The four weighted contribution indicators give the total level of contribution of a peer.

peers and a range $[0, 1]$ for Tribler peers (see Figure 4.5). The fact that Tribler peers are usually rated more contributive than other peers is valid, because a Tribler peers are not only bartering partners, but also a source for swarm discovery. These weighing factors are best guess values. Due to the limited time of our research we have omitted the evaluation and optimization of the weighing factors.
There is an exception to this calculation of the contribution C. When a peer is completely inconnective, this means that it has gone offline or has left the swarm. In this case, the other indicators cannot increase the contribution of the peer. Hence, if the connectivity C_c is close to zero, we will set the total contribution measure C to zero too.
C indicates the level of contribution for peers with a connection history that consists of at least one connection attempt. Peers without a connection history inherit their contribution $\bar{C}$ from their *source peers*, as explained in Section 4.4.5. Our definition of a source peer is as follows. When we receive a peerlist from a certain peer, we call this peer the *source peer* of all peers in the peerlist. Hence, all peers in the swarm database of a single client have a set of source peers. The set can also include the BitTorrent central tracker, if a peer was received from the tracker instead of a Tribler peer.
We will now give a detailed description of all four contribution measures (Sections 4.4.1–4.4.4) and then present the inheritance of contribution from source peers in Section 4.4.5.

4.4.1 Connectivity

Connectivity is a measure that indicates how well we can connect to a peer. Connectivity is calculated from the recent connection history of the peer. For a fixed period (set to three hours) a connection history is stored in the swarm database, containing the number of connection attempts c_a and the number of successful connections c_s. The ratio of successful connections over total attempts is used as the measure for connectivity. If no connections were attempted, the ratio is set to 1.0. The freshness of a peer, defined by the number of minutes ago that the peer was last seen online t_l, also influences our connectivity estimate. Hence, we will define the connectivity $C_c(p)$ of a peer p by

$$C_c(p) = \begin{cases} \frac{c_s}{c_a} \cdot \frac{1}{\max(1,t_l)}, & \text{if } c_a > 0 \\ \\ \frac{1}{\max(1,t_l)}, & \text{if } c_a = 0 \end{cases} \tag{4.2}$$

A successful bartering connection is here defined as a successful connection and BitTorrent handshake that is related to current download swarm. Therefore, peers that are online, but have left the swarm for which their connectivity is measured, also have zero connectivity.
The connection history of three hours has been chosen to give peers a "second chance". When a peer is found to be inconnective, it will be discarded and not connected again for three hours, until the record of the connection attempt is dropped

out of the connection history. The inconnectivity of this peer could be only temporary. After the short connection history is empty, the peers contribution level $\bar{C}$ will be calculated from inheritance again (see Section 4.4.5). If this contribution level is high enough, a second connection will be attempted.

4.4.2 Bartering activity

The primary form of cooperation in BitTorrent is to barter with other peers. So, bartering activity has a high weight in our contribution measure. We define the bartering activity of a peer as the amount of data it has uploaded to the local peer during the last hour. We chose this measure because it identifies peers that have a high-bandwidth Internet connection and enough pieces of the content to be interesting. The bartering activity measure will often give seeders a higher contribution. To get an up-to-date notion of the bartering activity of a peer we will only look at the data reception during the last hour instead of, for instance, the sharing-ratio of a peer.
Bartering activity $C_b(p)$ of a peer p is calculated by

$$C_b(p) = \frac{\log(\bar{b})}{\log(b_{max})}, \tag{4.3}$$

where $\bar{b}$ is the average bandwidth at which the peers uploaded to the local peer during the last hour and b_{max} the maximum bandwidth possible (set to 100 Mbit/s). Consequently, the logarithmic function will reward slow uploads with moderate bartering activity, while faster uploads gain bartering activity more slowly. For instance, a peer with an average upload speed of 1.86 KB/s has a C_b equal to 0.50 and an average upload speed of 100 KB/s gives a C_b equal to 0.73.

4.4.3 Swarm discovery activity

Tribler peers can increase their level of contribution by cooperating with the LITTLE BIRD protocol. This protocol cooperation is called swarm discovery activity. The swarm discovery activity of a peer is defined as the fraction of correct peerlists d that the peer has returned over the total number of received getPeers requests r. A peer is punished if it sends peerlist messages when they are not requested. Hence, the swarm discovery activity $C_a(p)$ for a peer p is defined as:

$$C_a(p) = \begin{cases} \frac{\max(0, d-d_e)}{r}, & \text{if } r > 0 \\ 1/2, & \text{otherwise} \end{cases} \tag{4.4}$$

where d_e is the number of unrequested peerlists. If a Tribler peer has never received a getPeers request, it will still have a swarm discovery activity of $1/2$. This approach gives unrequested Tribler peers already some extra level of contribution, so that they are preferred when creating a peerlist and propagate faster through the swarm.

4.4.4 Swarm discovery quality

If we have received a number of peerlists from a Tribler peer (if $d > 0$), swarm discovery quality is defined as the fraction of connective peers in those peerlists over the total number of peers in those peerlists. The swarm discovery quality indicator is used to unveil attacker peers that try to pollute the LITTLE BIRD protocol by sending peerlists with inconnective peers. In Section 4.5.5 this is explained in more detail. To calculate the connectivity of the peers in the peerlists, we use a more straight-forward definition of connectivity than in Equation 4.2, namely: A peer is connective if and only if it either has ever been connected or never a connection has been attempted. This simple definition suffices to unveil attackers that send polluting peerlists.
If the number of inconnective peers in the peerlists of some Tribler peer exceeds a fixed threshold T_a, the peer is immediately considered incontributive and receives a swarm discovery quality of 0.0. Tribler peers that have never sent a peerlist (if $d = 0$), receive full swarm discovery quality. The equation for swarm discovery quality $C_q(p)$ for peer p is given by

$$C_q(p) = \begin{cases} 1, & \text{if } l_r = 0 \\ 0, & \text{if } l_r - l_c > T_a \\ \frac{l_c}{l_r}, & \text{otherwise} \end{cases} \tag{4.5}$$

where l_r is the total number of peers received in peerlists, l_c is the number of connective peers in the peerlists, and T_a is the attacker threshold. The three possible sub-equations in Equation 4.5 distinguish a peer that has not sent us peers yet, an attacker peer that has sent too many inexistent peer addresses, and a regular peer that has sent us swarm information, respectively.

4.4.5 Inheritance of contribution

For a peer without a connection history, we do not have connection statistics in the swarm database to calculate the four presented contribution indicators. Still

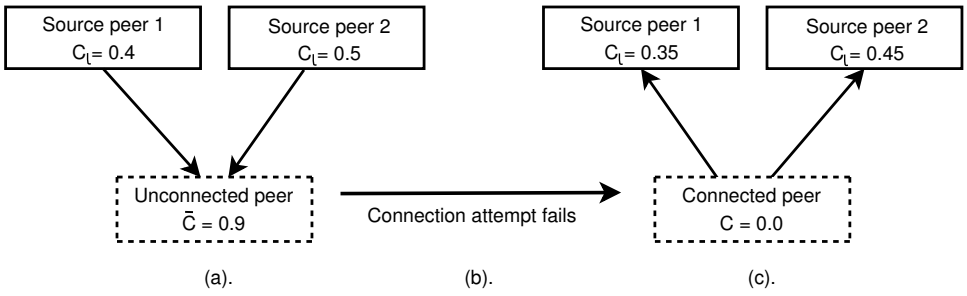

Figure 4.6: Dependency of contribution between an unconnected peer and its source peers. The contribution estimate is inherited from the source peers and the source peers are punished for the recommendation of a non-existent peer.

we need a contribution measure for this peer, to decide if connections to it will be attempted. Therefore, a peer without a connection history will inherit its level of contribution from its source peers (presented in Section 4.4). In the calculation of the contribution of the source peers, we do not take all four contribution indicators into account, but only those that describe the contribution to the LITTLE BIRD protocol (C_a and C_q). Had we taken the connectivity and bartering activity of the source peers into account, then the level of contribution of an inheriting peer would decrease when a source peer leaves the swarm, which does not make sense if that source peer has always cooperated to LITTLE BIRD and provided existing peers. Therefore, we define the level of cooperation to the LITTLE BIRD protocol C_l as

$$C_l = \frac{0.1 \cdot C_a + 0.2 \cdot C_q}{0.1 + 0.2}. \tag{4.6}$$

The weighing factors in Equation 4.6 have the same ratio as in the equation to calculate the level of cooperation from four indicators and give C_l a range of $[0, 1]$. The inherited level of contribution $\bar{C}(p)$ of peer p is defined by

$$\bar{C}(p) = \sum_{s \in S_p} C_l(s), \tag{4.7}$$

where S_p is the set of all source peers of peer p.

If we combine Equation 4.5 and Equation 4.7, we see that the level of contribution of an unconnected peer and that of its source peer can influence each other in both directions. This bidirectional dependency is shown in Figure 4.6. In Figure 4.6a, the contribution of an unconnected peer is calculated by the contributions of its source peers, according to Equation 4.7. Based on this contribution estimate, the peer is selected for connection, but it appears to be inconnective (Figure 4.6b). The inconnective peer will now have a contribution level of 0.0 and the contribution of the source peers is reduced (using Equation 4.5) because they have recommended a non-existent peer (see Figure 4.6c).

The consequence of these dependencies is that a connection attempt to an unconnected peer can influence the level of contribution of other unconnected peers through their mutual source peer. The peer selection component uses the contribution of the influenced unconnected peers to decide whether or not to connect to them as well. Hence, an initial connection attempt to a peer may influence if we want to attempt initial connections to other peers. The peer selection component is designed to connect to a limited number of unconnected peers per selection step. By selecting fewer unconnected peers per step, the connection attempts to other peers can be reconsidered based on the outcome of the currently executed attempts.

When our protocol is combined with standard BitTorrent, peers can also be acquired from the central BitTorrent tracker. In this case, the tracker is defined to be the source peer and the unconnected peer gets a reliability equal to 1.0.

4.5 Design requirements

In this section we will explain how we have complied with the design requirements presented in Section 2.2 in the design of the LITTLE BIRD protocol. Most of these solutions will be evaluated in Chapter 5.

4.5.1 Bootstrapping

Swarm discovery bootstrapping is the problem of finding initial peers in the download swarm that are compatible with the LITTLE BIRD protocol. These discovered initial peers are used to request more peers in the swarm and start the bartering process. In the LITTLE BIRD protocol, a bootstrapping peer only sends requests to peers that are semantically related to the swarm it wants to discover. Peers will only be requested to assist others in the discovery of a swarm, if they are currently or were recently active in that particular swarm. In Section 4.5.4, we explain how this gives the peers an incentive to cooperate with the LITTLE BIRD protocol.
To discover initial peers that are currently or have been recently active in the download swarm, LITTLE BIRD employs the data of the epidemic *BuddyCast* protocol. BuddyCast, presented in Section 1.3, is a distributed recommendation protocol for the Tribler network that uses preference lists to exchange the taste of peers between all online Tribler peers. Through the reception of preference lists, each peer builds a database with recommended torrents and a list of *recommender peers* for

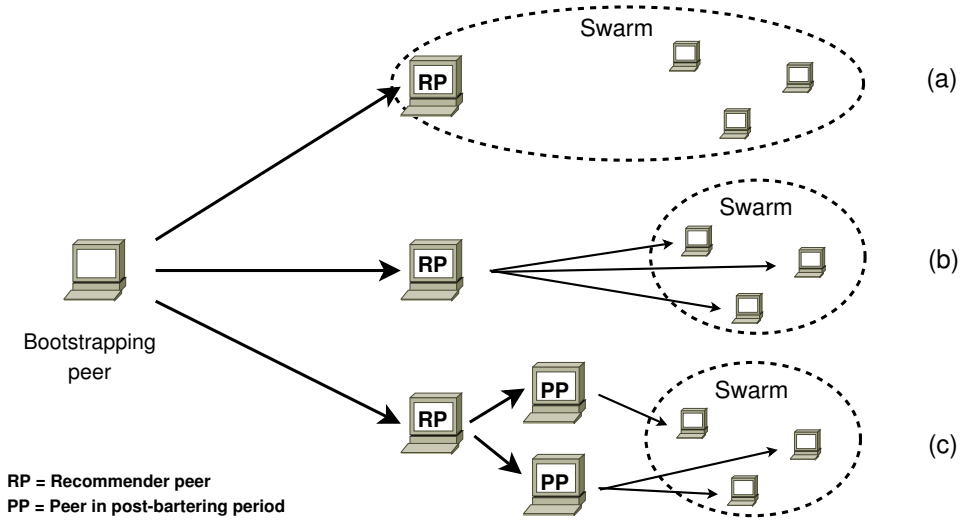

Figure 4.7: Three possible ways to discover a swarm through recommender peers. (a) The recommender peer is currently active in the swarm. (b) The recommender peer is in the post-bartering period, but knows peers active in the swarm. (c) The post-bartering recommender peer knows other post-bartering peers that help to discover the swarm.

each torrent. Recommender peers are the peers that have recently recommended a torrent. These peers are our first contacts to discover the swarms related to the recommended torrents. The BuddyCast protocol has the advantage that the received preference lists contain torrents that are in accordance with the local peer's taste. Therefore, its most favorite swarms will be the easiest to bootstrap and discover.

Recommender peers are either still active in the swarm or have already left. Both groups of peers have to able to handle a swarm discovery request. Therefore, we defined a period after a peer leaves a swarm in which the peer still stores the swarm information, called the *post-bartering period*. During the post-bartering period, set to 10 days, a Tribler peer will respond to getPeers request from bootstrapping peers with a peerlist containing its most recent view on the swarm. This view included the peers that were active in the swarm before the post-bartering peer left it.

Through our initial group of recommender peers, there are three ways to finish the bootstrapping process. These possibilities are shown in Figure 4.7. Figure 4.7a shows the situation in which a requested recommender peer is still active in the swarm. This is the most fortunate situation that directly completes the bootstrapping process. A getPeers request to the recommender peer will directly give substantial swarm information and the bootstrapping peer can start bartering. This situation is likely to occur when BuddyCast preference lists spread quickly over all Tribler peers. Peers will then find out what others are downloading before those

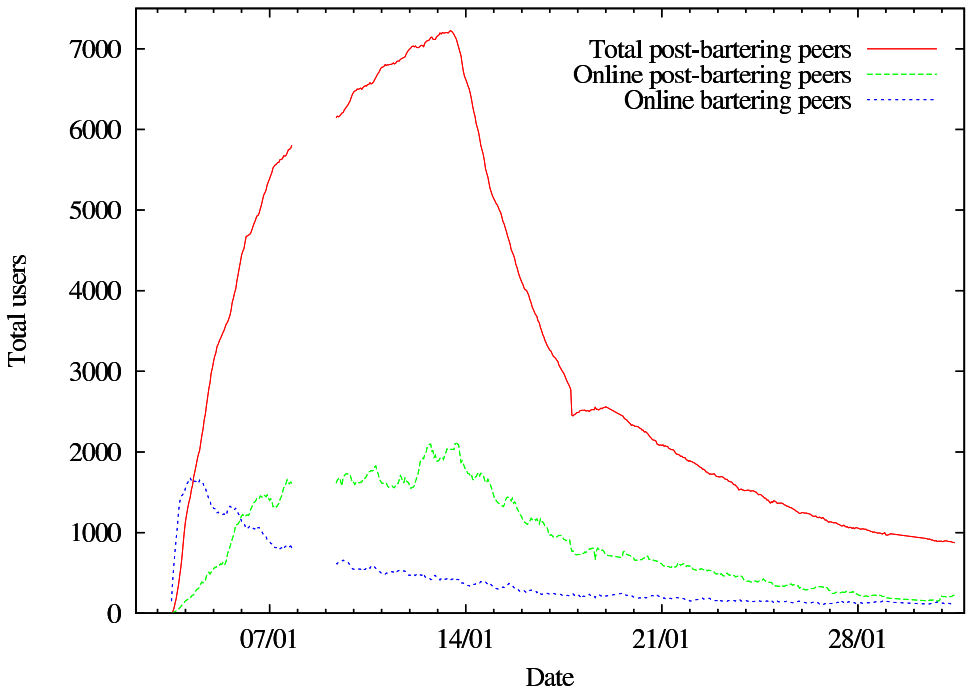

Figure 4.8: The current number of peers in the swarm, the number of peers in the post-bartering phase and the number of online post-bartering peers.

peers have finished their downloads.
In Figure 4.7b, the recommender peer has left the particular download swarm, but is still in its post-bartering period. Upon a request from the bootstrapping peer, it will return the list of peers that it knows from its bartering period. When a part of these peers is still active in the swarm, the local peer has finished bootstrapping. The active peers can be used for subsequent LITTLE BIRD requests. However, when none of the peers that were returned by the post-bartering recommender peer are active in the swarm, we have to do another discovery step (see the situation in Figure 4.7c). The bootstrapping peer can send getPeers requests to the inactive peers and see if one of them is in the situation that it still knows active swarm peers. A bootstrapping peer can continue to send swarm discovery requests as long as there are online peers that have recently left the swarm. It will either finally find active swarm members, or run out of peers to request.
The fact that Tribler peers will still respond to getPeers messages related to download swarms that they have recently left simplifies the bootstrapping problem. Instead of finding peers in a certain swarm, a peer has to find peers that are online and have recently been in this swarm. This is exactly the information that is provided by the BuddyCast protocol. In Figure 4.8 we show the total and online number of peers in the post-bartering period and the number of bartering peers of the swarm introduced in Chapter 3. The total number of peers in the post-bartering period has an explosive growth. This group of peers is basically all users that had interest in the content of the swarm. If we look at the online post-bartering peers of this swarm, we see that since two days after the swarm creation their number also is larger than the number of peers currently in the swarm. This proves that by engaging post-bartering peers in the bootstrapping process, the probability of quickly discovering a swarm has increased considerably.
After a Tribler peer has succeeded in bootstrapping and starts bartering in a swarm, it will not send requests to peers in the post-bartering period anymore. Requests are only send to other bartering Tribler peers, in order to keep the bandwidth overhead for post-bartering peers to a minimum.
Both for bootstrapping and for swarm discovery in general the online probability of peers in the community is very important. There may be many peers holding swarm information of a swarm that some peer wants to discover, but if most of these peers are currently offline, the swarm discovery still fails. In Section 3.4.4 we measured that in the filelist community, 40% of the users that were online on an initial day are still online during the 10 following days. Other research [4] indicates that in the *etree* community [25] this is 11.1%. These promising high online probabilities will realize reliable decentralized swarm discovery in active communities.

4.5.2 Swarm coverage

When a peer is bootstrapped, it knows at least one Tribler peer in the particular swarm. Then it will use this peer to maximize its swarm coverage and get to know a significant part of the swarm peers.

The LITTLE BIRD protocol is designed to let peers discover all Tribler peers in a swarm in a fast way. This can be realized because peers with a higher level of contribution are prioritized during the creation of a peerlist. The peers that are most valuable to a bartering peer are exchanged first and peerlists are a qualitative selection from the swarm instead of a random selection, which a BitTorrent tracker would give. Tribler peers, with their additional contribution level, are also prioritized. Hence, a peer receives a growing number of swarm discovery sources and can manage to get a high swarm discovery.

In the current setting, peers will send a maximum of three getPeers requests every 10 minutes. The number of requests that will be send can be limited by a lack of Tribler peers or load balancing rules (see Section 4.5.3). Every getPeers request will return a peerlist with a maximum of 100 peers, so a peer can receive up to 300 peers every 10 minutes.

In Section 4.5.3, we will explain how the use of Bloom filters prevents a requesting peer to receive peer information that it already has. Except for the efficiency improvement, the addition of Bloom filters in LITTLE BIRD also increases the exchange of new swarm information and thus increases swarm coverage.

In Section 2.2, we described the possibility of swarm partitioning if there is no central tracker, low swarm coverage, and high churn. In theory, a swarm could become partitioned when using our decentralized protocol. In practice, however, the request frequency of LITTLE BIRD is high enough to compensate for the churn measured in Section 3.4.2. The churn measurements show that during 10 minutes, at most 5% of the swarm leaves and is replaced by new peers.

To analyse the probability of swarm partitioning, a swarm can be modeled as a undirected graph. In this model, peers are vertices in the graph and for each two peers that know each other's network address, there is an edge in the graph. The swarm coverage of a peer can in this analogy be seen as the degree of the related vertex in the graph. A swarm is partitioned when the graph that can be modeled from it is not connected. A graph is connected if there exists a path between any two of its vertices.

We give two example graphs in Figure 4.9, of which one is connected and one is

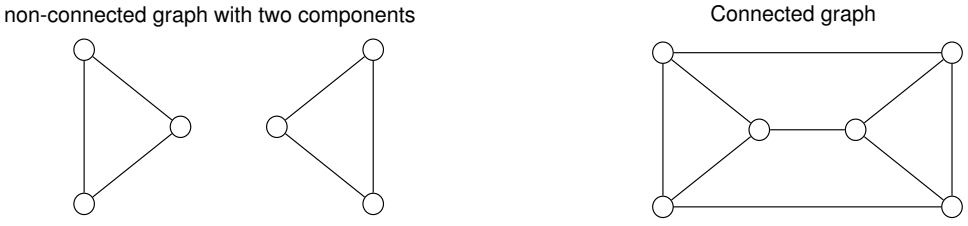

Figure 4.9: Two graphs with size vertices. The graph on the left is not connected and consists of two parts. All vertices have degree $d(v) = 2$. In the graph on the right all vertices have degree $d(v) = 3$ and must be connected, as stated by Theorem 4.1.

partitioned. We present Theorem 4.1 in order to conclude when swarm partitioning might occur.

Theorem 4.1 *A graph $G = (V, E)$ in which each vertex has a degree at least equal to $\lceil \frac{|V|}{2} \rceil$ is connected.*

Proof Assume that all vertices have degree at least equal to $\lceil \frac{|V|}{2} \rceil$ and that the graph G is not connected, but consists of $p > 1$ disjoint connected components $V_i \subset V, i = 1, \ldots, p$.
Then there exists a component V_j with $|V_j| \leq \frac{|V|}{p}$. Then, for every vertex k in V_j, the degree $d(k)$ of k satisfies $d(k) \leq \frac{|V|}{p} - 1 \leq \frac{|V|}{2} - 1$, which contradicts our assumption that every vertex has degree at least equal to $\lceil \frac{|V|}{2} \rceil$.

When we interpret Theorem 4.1 from a swarm perspective, it states that for each swarm in which all peers have a swarm coverage higher than 50%, the swarm can not be partitioned. Assume that the minimal swarm coverage is $(50 + c)\%$ in a swarm, then if $c\%$ of the peers in this swarm leave, the minimal swarm coverage will still be higher than 50% and the swarm can not partition. This means that for the measured 5% peers that leave between two subsequent LITTLE BIRD request rounds, the swarm can not become partitioned if the minimal swarm coverage of each peer is at least 55%.

4.5.3 Scalability and load balancing

The design of LITTLE BIRD was foremost focused on its scalability. Therefore, we chose for an epidemic protocol for the communication of swarm information, see Section 4.2. LITTLE BIRD is currently configured so that each peer sends at most three requests every 10 minutes. The average number of requests that each peer has to respond to is equal to the number of requests sent. The actual number of getPeers requests that a peer receives depends on the contribution of the peer. Peers with a high contribution level will be known by more peers in the swarm and are more frequently requested. In addition to contribution, peers prefer to send requests to a variated set of peers. A requesting peer values variation of peers, because if it requests the same peer quickly after a previous request, the peer is not likely to have new swarm information.
There could be a situation in which a small part of all peers is very popular and are seen as very contributive to the rest of the swarm. These peers will then receive more getPeers requests than the others. We have added a load balancing mechanism to LITTLE BIRD in order to prevent such an unbalanced load on peers. To balance the load over the total swarm, each peer maintains an interval length I (in minutes), which is included in peerlists. Similar to the central tracker protocol of BitTorrent, each peer can set this interval to the time that it does not want to receive requests from a requesting peer. Peers have to respect this interval, because otherwise their level of contribution from the point of view of the responder will

decrease. Furthermore, the requested peer will simply not respond, because with the secure identification in Tribler, it can verify the compliance of the requester with the transfered interval.
Peers will set I to dynamically manage the amount of overhead they want to spend on handling getPeers requests. The value of I is calculated by

$$I = \max(30, 2 \cdot \frac{r_{15}}{r_m}), \tag{4.8}$$

where r_{15} is the number of received getPeers messages in the last 15 minutes (for all swarms that the peer is active in) and r_m is the maximum number of getPeers messages we want to handle per minute. We will set r_m equal to 4 getPeers requests per minute, enabling the local peer to serve 120 requesters that each send a request every 30 minutes. On average, a peer can be in 12 swarms before it will set its interval I to a value greater than 30 minutes to stabilize the frequency in which it receives requests.
These limits give every peer enough freedom to send getPeers requests. When a download swarm is big, there are many candidates to send a getPeers message to, when it is small, all peers in the swarm will be received during the first request(s). The churn will be so minimal, that there is no need for requests more frequent than every 30 minutes.
The use of interval I only has effect on the frequency of subsequent requests of peers. When many peers send an initial getPeers request to an overloaded peer, the only option the peer has, is to ignore part of the requests..
The conventional BitTorrent centralized tracker returns a list of random peers from the swarm upon a request. When a swarm is small or when a peer already knows many peers, it is very likely to receive redundant peers. Since we have improved the 'memory' of a peer by the addition of the swarm database, the probability of receiving duplicate peers is even higher.
For this reason we have included the list of currently known peers in the getPeers message. This is implemented efficiently using a *Bloom filter* [12]. A Bloom filter is a dense data structure, which is used to store or test the membership function of a set [67, 13]. In our implementation, all known peers that a peer has tried to connect to are included in a Bloom filter, so that the receiver of the getPeers message can test which of his peers are still unknown to the requester. Only these new peers are included in the peerlist, which makes the communication much more efficient. The Bloom filter inclusion is already bandwidth efficient when only a few peers can be left out, because the Bloom filter uses only 16 bits per added peer in the getPeers message.
A Bloom filter is a probabilistic data structure, which introduces a false-positive probability when testing for members. In our case, when a false positive occurs, a peer will not be included in a peerlist, although it is new to the requester. The false-positive probability for a Bloom filter of 16 bits per elements is $4.6 \cdot 10^{-4}$, which is so low that the reduction in bandwidth is worth it.
The resulting overhead is shown in Table 4.1. We assume that an average peerlist

contains the information of 50 peers, because of the use of Bloom filters. In reality this may be less, because of the use of a Bloom filter, which we will describe below.

	Average (per torrent)		Maximum (for t torrents)	
Sending requests	3	24 KBytes	$3 \cdot t$	$24 \cdot t$ KBytes
Handling requests	3	24 KBytes	40	320 KBytes

Table 4.1: The overhead in number of requests and bytes needed for sending and handling distributed tracker requests per 10 minutes.

4.5.4 Incentive to cooperate

In Section 2.2 we have already explained that peers need to have incentives to cooperate. Without such an individual stimulus, there will be less cooperation in the total community [20, 4]. In our implementation we have realized this incentive through the definition of contribution.
A peer that responds to getPeers requests gains contribution from the perspective of the requesting peers, because their value of swarm discovery activity C_a grows. This will increase the probability that other peers attempt to barter with it in the future and thus increase its download speed. Cooperation with the LITTLE BIRD protocol also makes it more likely that new peers will connect to a peer. Hence, investing a little amount of bandwidth in order to send a peerlist is profitable for a peer even from a individualistic point of view.
When a peer is in its post-bartering period for a certain swarm, it has no direct incentive to help others in that particular swarm. This is caused by the fact that the contribution in our current design is defined for each peer in each swarm. Cooperation in a swarm that a peer has left will increase its contribution level for that swarm, but not for future swarms, where the peer can profit from it. It would be a good addition to LITTLE BIRD to combine the contribution levels of a single peer that is known from different swarms. This addition would give a peer in the post-bartering period an incentive to help bootstrapping peers, because this help will make it more trusted in future swarms.

4.5.5 Integrity and security

Now we have replaced a trusted centralized tracker by distributed swarm discovery, we have to implement a more strict filtering policy to ensure integrity of our swarm database. Received peerlists have to be handled as untrusted data. In Section 2.2 we introduced the pollution attack and dDoS attacks as most important attacks to focus on. We will show how LITTLE BIRD uses its contribution measure as protection against them. In Chapter 5 we will evaluate the effects of both types of attacks on LITTLE BIRD.
The design decision to place the initiative of swarm discovery on the side of the requesting peer is a first protection against this sort of attacks. A peer only accepts

a peerlist that it has explicitly requested, so attackers have to provoke it to send a getPeers request to them. This can only be done by realizing a high level of contribution through cooperation and bartering of content. The load balancing rules will make only one request per 30 minutes possible to each peer. If an attacker wants to receive more requests, it has to create multiple contributive identities.
When an attacker has managed to receive some getPeers requests from the swarm, it can return an erroneous peerlist. The receiving peers will add these peers to their swarm database, but this does not mean the received peers are trusted. When a connection to a received peer fails or when the peer appears not to be in the download swarm, the reliability of the peer is directly set to zero. In this case, the swarm database has the function of a blacklist and connections to this peer will not be retried. These peers are not given a second chance (as was explained in Section 4.4.1), because after three hours their level of inherited contribution will still be too low for reconnection.
These inconnectible peers will also never be included in peerlists. A responder peer will only include peers to its peerlist that it has had an outgoing connection to. This *check before you tell* strategy is in fact a very important design decision. Correct peer information may be disseminated slower through the swarm, but we can guarantee that inconnectible IP addresses are never forwarded by honest peers. While reliable peer information is spread through the swarm very quickly using the epidemic protocol, pollution is not spread at all. Hence, an attacker has to infect each peer individually.
When a portion of the peers from a peerlist can not be connected, the contribution of the source peer will be reduced. Hence, the peers that have received the erroneous peerlist from the attacker will in the future prefer other peers for swarm discovery. When the number of inconnectible received peers exceeds the threshold T_a, defined in Section 4.4.4, the source peer will be considered an incontributive peer (attacker) for this swarm and is never included in future peerlists. Consequently, both the attack peers as their polluted peer data will be will not be forwarded though the swarm. If the attackers are reliable bartering partners, bartering with them will continue.
During a dDoS attack, the attacker peers attempt to spread peer information consisting of a single IP address. The goal is to incite the incite the other Tribler peers in a swarm to connect to this address and overload the victim's computer (see Figure 2.1). Our contribution measure is designed in such a way that it can be calculated before we attempt a connection to a peer, because we would already participate in the dDoS attack by attempting connections to peers received from the attackers. This solution lets peers ignore the peerlists received from attackers and connect only to the more reliable ones.
We have also create an additional defense mechanism against dDoS attacks. Each peer selection step, connections will only be attempted to peers with different IP addresses. When a peer has received many peer addresses containing the same IP address, it will connect to only one of these addresses per minute.
With these two solutions, an attacker can only incite a Tribler peer to connect to

a single IP address once every selection step (executed every minute), until the attacker is found to be incontributive and it will not be requested anymore. From dDoS attack measurements [55] can be concluded that actual attacks use a packet rate of more than 1000 packets per second. With the current configurations, attackers would need 60,000 peers for such a minimal attack. Soon, these peers will conclude that the source peer of the inconnectible IP addresses does not contribute to the protocol and stop all together with connecting to the victim. The only way to continue the attack is by creating new identities and flood the thousands of peers again.

We conclude that the attack resilience against dDoS attacks in LITTLE BIRD is sufficient, because it costs more bandwidth for an attacker to incite other peers to help in the attack, than to make connections to the victim itself. Hence, using Tribler peers to execute a dDoS attack is not profitable.

Chapter 5

Evaluation

Don't be too timid and squeamish about your actions. All life is an experiment. The more experiments you make the better.

— Ralph Waldo Emerson

In this chapter we evaluate the performance of LITTLE BIRD. We have carried out an experiment to test LITTLE BIRD decentralized bootstrapping and two large-scale emulations to test the general performance of LITTLE BIRD and the performance under attack. To realize these large-scale emulations, we have developed CROWDED, a trace-based swarm emulation environment for the DAS-2 super computer. CROWDED makes it possible to use one of our swarm measurements from Chapter 3, and reproduce the swarm behavior by starting or stopping Tribler applications for each joining or leaving peer.
In Section 5.1, we present the hardware setups that we used for our three experiments. We describe the CROWDED emulation environment in Section 5.2. In Section 5.3, we evaluate the decentralized bootstrapping of LITTLE BIRD on a small scale. We demonstrate how LITTLE BIRD successfully realizes fully decentralized bootstrapping. In Section 5.4, we use the CROWDED environment to emulate a swarm of LITTLE BIRD-supporting peers and evaluate the general performance of the swarm discovery. We show that all peers manage to discover the majority of the swarm. In Section 5.5, we emulate a dDoS and pollution attack on a swarm and see if the protocol is resilient against misuse.

5.1 Hardware setup

We have used two hardware setups for our evaluations. The evaluation of the bootstrap functionality of LITTLE BIRD was carried out on a single computer. For this experiment we did not need significant computation power. The other evaluation experiments, presented in Sections 5.4 and 5.5, are conducted on the Distributed ASCI Supercomputer 2 (DAS-2).
The DAS-2 is a wide-area distributed computer of 200 Dual Pentium-III nodes [23]. The machine is built out of five interconnected clusters of workstations,

located at Vrije Universiteit Amsterdam, Leiden University, University of Amsterdam, Delft University of Technology and University of Utrecht. We chose to use the DAS-2 in order to have enough computing power at hand to emulate the measured BitTorrent swarms with the actual Tribler software. Hence, we can carry out a realistic analysis of how LITTLE BIRD performs in combination with BitTorrent. For the execution of these emulation we have create the CROWDED emulation environment.

5.2 CROWDED emulation environment

We have designed the CROWDED environment to emulate a complete swarm by executing the actual Tribler applications on a supercomputer. These Tribler applications support the LITTLE BIRD swarm discovery protocol, so that it can be evaluated. CROWDED needs as input data the swarm behavior measurements from the filelist community (see Section 3.4.3). It will then reenact the behavior of each peer in the swarm, by starting a Tribler application on each measured join-moment of the peer and stopping it when the peer was found to have left the swarm.
We have used two grid computing tools for the operation of CROWDED, namely *Koala* and *Grenchmark*. Koala is a grid scheduler [54] that has been designed at the PDS group in Delft. Koala offers simultaneous allocation of nodes in multiple clusters of the DAS-2 to an application, in order to flexibly use DAS-2 nodes independent of their cluster. Grenchmark [40] is a grid benchmarking tool, which we used for the initiation and timing of our emulations. This software is designed to launch many simultaneous jobs to a super computer grid in order to benchmark the job scheduler or other grid middle-ware. We have extended the workload submitter part of Grenchmark, so that it can send emulation commands to the CROWDED environment.
CROWDED can be compared to the Symptop simulation toolkit for P2P networks [73], which has been developed as a MSc. project in the PDS group. Symptop also runs P2P applications to emulate and analyse their behavior in a network. Its design was focused on the Gnutella and Overnet P2P networks. We decided to build the CROWDED environment, because Symptop lacks the input format and the output statistics that would be practical for our trace-based emulations. In Symptop, peers behave based on distributions of, for instance, join-rate and life time. We value our approach to do trace-based emulations in which peer behavior is not based on static distributions, but on measured behavior. The output statistics of CROWDED also enable us to extract detailed statistics about the LITTLE BIRD protocol, where Symptop's output is focused on raw measurement of connections and bandwidth.

5.2.1 Architecture

The CROWDED environment consists of the *main controller* and a series of *node listeners*, see Figure 5.1. The main controller receives start and stop commands

from the Grenchmark workload submitter in real-time. It has the task of routing these commands to the correct node listener. The node listeners, which run on computing nodes on the DAS-2, receive these commands and execute them. The node listener to start a new Tribler application on, is selected based on two rules. When a user joins the swarm for the first time, the related Tribler application is started on the node listener that runs the fewest users. This rule guarantees that the Tribler applications of all users are evenly divided over all DAS-2 nodes. When a peer joins the swarm for a subsequent session, it will be run on the same node as the previous sessions. The working directory of the peer has been stored on this particular DAS-2 node, so that the user can access its own swarm database, data

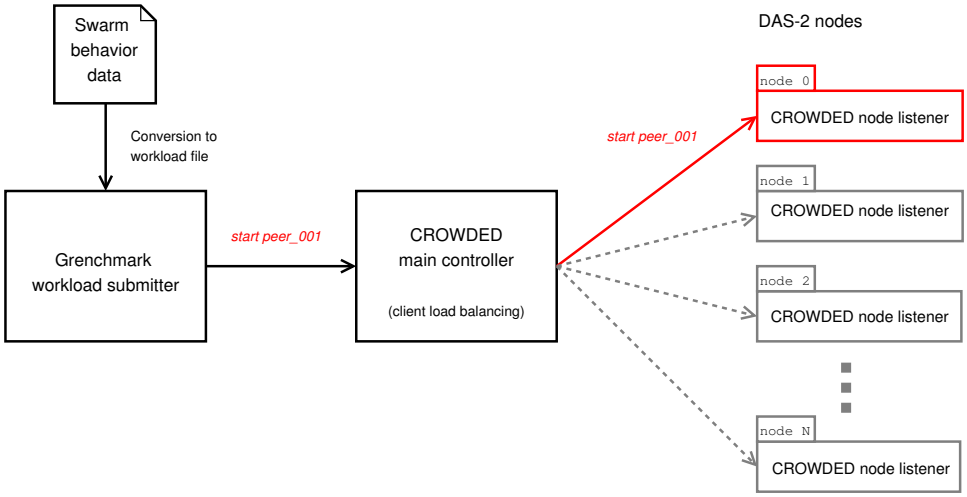

Figure 5.1: The CROWDED main controller handling a start peer command.

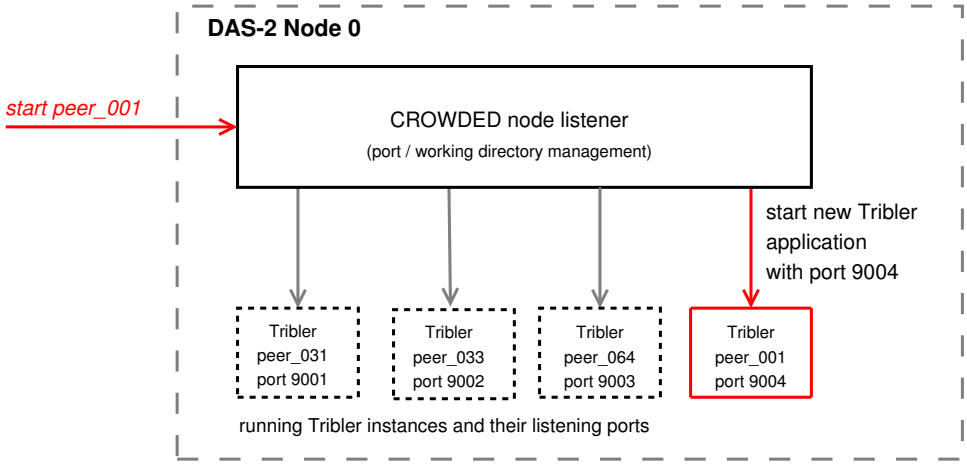

Figure 5.2: The CROWDED node listener handling a start peer command by starting a new Tribler application.

files, and other configuration.

An emulation experiment is executed as follows. First, the CROWDED main controller is started. Then, DAS-2 nodes are allocated on the super computer using the Koala grid scheduler and a CROWDED node listener is started on each of them. The necessary number of nodes depends on the size of the emulation. Each node listener handles the control of the node by preparing it for the experiment, running and stopping Tribler applications, and acquiring emulation statistics after the emulation has ended. During the preparation of the experiment, the node listeners copy the Tribler source code to the local node and notify their ready-state to the CROWDED main controller.

After all node listeners have notified the main controller that they are standing by to receive commands, the main controller starts the Grenchmark workload submitter and begins to receive commands. The commands are then routed to the correct node listeners. There are four commands that the main controller can send to a node controller:

- Start [peer name] - Start a Tribler application as a leecher under the node controller.
- Seed [peer name] - Start a Tribler application as a seeder under node controller.
- Stop [peer name] - Stop a Tribler application on this node.
- Kill - Stop all Tribler instances on this node, copy acquired statistics to the storage server and quit the node controller.

When a node listener receives a *start* command for peer p, it checks if it is the first session of peer p in the swarm. If it is, a working directory is created and Tribler is started. Otherwise, a Tribler application is started using the existing working directory from previous sessions. The working directory stores among other things the swarm database and (in)complete downloaded files. Hence, a peer will keep the knowledge stored in the swarm database over multiple sessions.

Figure 5.1 shows how the main controller receives the command *start peer_001* and routes it to node 0. The node listener on node 0 receives the command and directly starts a new Tribler applications (see Figure 5.2). A unique listening port number and working directory are associated with peer_001, so that it can be connected by other peers in the swarm.

The node listener starts all Tribler applications in *nice* cpu, which lets the operating system schedule their processes with a priority lower than that of the node controller process. This increases the responsiveness of the node controller and prioritizes the execution of commands above the operation of a Tribler application.

Computation node limitations

We created the CROWDED environment in order to run many Tribler programs in parallel on a limited number of super computer nodes. For an emulation of a swarm that has a certain maximal size, we like to know how many DAS-2 nodes we need. With the knowledge that CROWDED divides Tribler applications evenly over the available nodes, we need to know how many Tribler applications can run on a single node. To get a notion of this, we have measured the 1-minute load of a DAS-2 node under different numbers of running Tribler applications. The system load describes the amount of work that a computer system is doing [21]. The results are shown in Figure 5.3, which has a logarithmic vertical axis.
When a small number of Tribler application are running, the load is smaller than 1.0 and thus the processes of the Tribler applications are quickly scheduled which guarantees smooth operation. Above 13 Tribler applications on single node, the load starts to rise faster and execution of the Tribler application becomes slower. This situation does not resemble Tribler executing on a personal computer anymore. Therefore, we conclude that per node, a maximum of 13 Tribler applications can run under realistic circumstances. During the large-scale emulations of Chapter 5, where we let up to 13 Tribler application run per node, it showed that in this setting the cpu is fully used.

Experimental statistics

After we have performed an emulation in the CROWDED environment, the emulation statistics are gathered. To generate these statistics, each Tribler application writes all protocol details to a separate log file. From these log files the emulation is analyzed. The following statistics are extracted from the log files of each Tribler application:

- Session information. The online and offline times of all peers are stored in order to recalculate the actual swarm size and relate other statistics to the

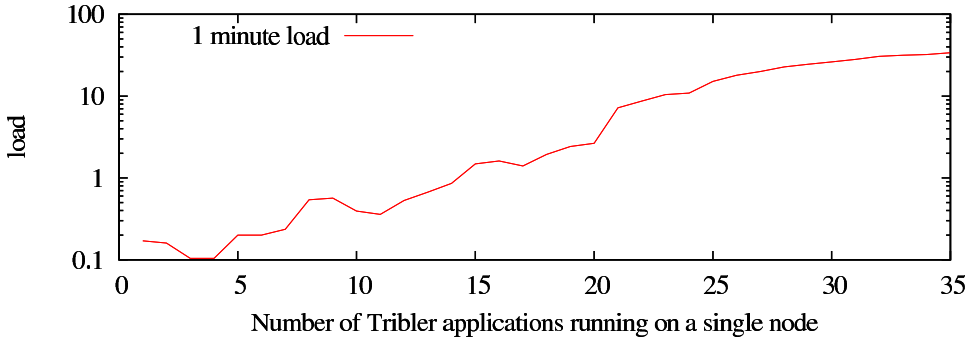

Figure 5.3: The 1-minute load of a DAS-2 node as a function of the number of Tribler programs scheduled on that node.

session length of peers.

- Swarm coverage data. Each peer stores a periodic list of all IP addresses that it knows in the swarm. During the analysis, these IP addresses are matched against the online information, resulting in the swarm coverage of each peer.

- LITTLE BIRD communication. The number and bandwidth of the outgoing and incoming LITTLE BIRD messages are stored. This gives an insight to the bandwidth usages and load balancing of the protocol on each moment.

- Security statistics. To test the resilience of the protocol against incorrectly communicating peers, we have done experiments in which a part of the peers tries to pollute the rest with false IP addresses. The exact number of pollution broadcasted and the number of connection attempts to these fake IPs is logged.

We combine the information from all log files to create statistics that describe the complete swarm. We will present these statistics in Chapter 5, where we have used CROWDED extensively.

5.2.2 Proof of concept

In this section we will perform a simple experiment in order to show the correct operation of the CROWDED environment in practice. The experiment consists of the creation of a workload file for the Grenchmark workload submitter, the emulation, and the presentation of the statistics. Normally, the workload file is created from our BitTorrent measurements. To keep the experiment simple, we will create this workload file by hand, containing a miniature swarm with three peers running on two nodes. Table 5.1 shows which join and leave times of the three peers were stored in our workload file.

	Join time (minute)	Leave time (minute)	Seeder/leecher
Peer 0	0	18	seeder
Peer 1	6	18	leecher
Peer 2	12	24	leecher

Table 5.1: The join and leave times in minutes after start of the emulation of the three peers in our small example swarm.

After we have started CROWDED with our workload file as the input, the three Tribler applications have joined and left the miniature swarm, reenacting the behavior that we defined. They use a central tracker for swarm discovery, because our focus lies on the operation of CROWDED. The emulation is finished after 24 minutes. The log files of the main controller and the Tribler applications show that the emulation has been executed as planned.

```
00:49:42 - Sending seed peer_000000 to node ('node308', 6050)
00:49:42 - node308: 1,  node319: 0,  TOTAL: 1

00:55:42 - Sending start peer_000001 to node ('node319', 6050)
00:55:42 - node308: 1,  node319: 1,  TOTAL: 2

01:01:42 - Sending start peer_000002 to node ('node319', 6050)
01:01:42 - node308: 1,  node319: 2,  TOTAL: 3

01:07:42 - Sending stop peer_000001 to node ('node319', 6050)
01:07:42 - node308: 1,  node319: 1,  TOTAL: 2
01:07:42 - Sending stop peer_000000 to node ('node308', 6050)
01:07:42 - node308: 0,  node319: 1,  TOTAL: 1

01:13:42 - Killing all nodes...
01:13:42 - Sending kill to node ('node308', 6050)
01:13:42 - Sending kill to node ('node319', 6050)
01:13:42 - Ready killing all nodes.
```

Log File 5.1: Snippets of the log file of the CROWDED main controller.

```
00:49:42 - Starting peer peer_000000 on 130.161.211.208:9000
01:07:42 - Stopping peer peer_000000

00:55:42 - Starting peer peer_000001 on 130.161.211.219:9000
01:07:42 - Stopping peer peer_000001

01:01:42 - Starting peer peer_000002 on 130.161.211.219:9001
01:13:42 - Stopping peer peer_000002
```

Log File 5.2: Snippets of the log files of the three Tribler applications.

Log File 5.1 shows snippets of the CROWDED main controller log file. Each line starts with a timestamp, followed by a logging message. These logging messages describe the transfer of commands to selected nodes and the number of Tribler instances running on each node. According to the log files, peer 0 was started on node308 and peers 1 and 2 on node319. We conclude that the CROWDED environment manages to distribute new Tribler applications evenly over the allocated nodes. Each of the three new Tribler applications is started on the node with the fewest running clients. The number 6050 indicates the listening port of the node listeners, which is used to communicate with the main controller. At the end of the experiment, all node controllers receive the *kill* command and the experiment is ended.

We will also look at the log files of the three Tribler applications that have been running. The session information from these log files is shown in Log File 5.2. From the log file snippets, we can see that the node controllers correctly assigned unique listening ports to the Tribler programs. Also, the start and stop times correspond with those defined in our workload file. The IP addresses 130.161.211.208 and 130.161.211.219 belong to node308 and node319 respectively.

In Figure 5.4, we show the swarm size of our miniature swarm and the download progression of the two leecher peers (peers 1 and 2). The swarm size follows ex-

actly the join and leave times of the peers that we stated in the workload file as we already saw in the logging data. This proves that the CROWDED environment manages to reenact swarm behavior by the controlled starting and stopping of peers. The download progression of peer 1 and peer 2 shows that during the experiment a genuine BitTorrent swarm is formed in which the peers can barter with each other. We will use the CROWDED experimental environment for the emulations in Sections 5.4 and 5.5.

5.3 Bootstrap evaluation

In this section, we show that the LITTLE BIRD protocol makes content and swarm discovery possible without a central BitTorrent tracker. We give an example situation in which a peer has created a torrent file without a tracker address and distributes this trackerless torrent through the BuddyCast protocol. Other peers discover the new content and perform the swarm discovery bootstrapping through the LITTLE BIRD protocol, as explained in Section 4.5.1.

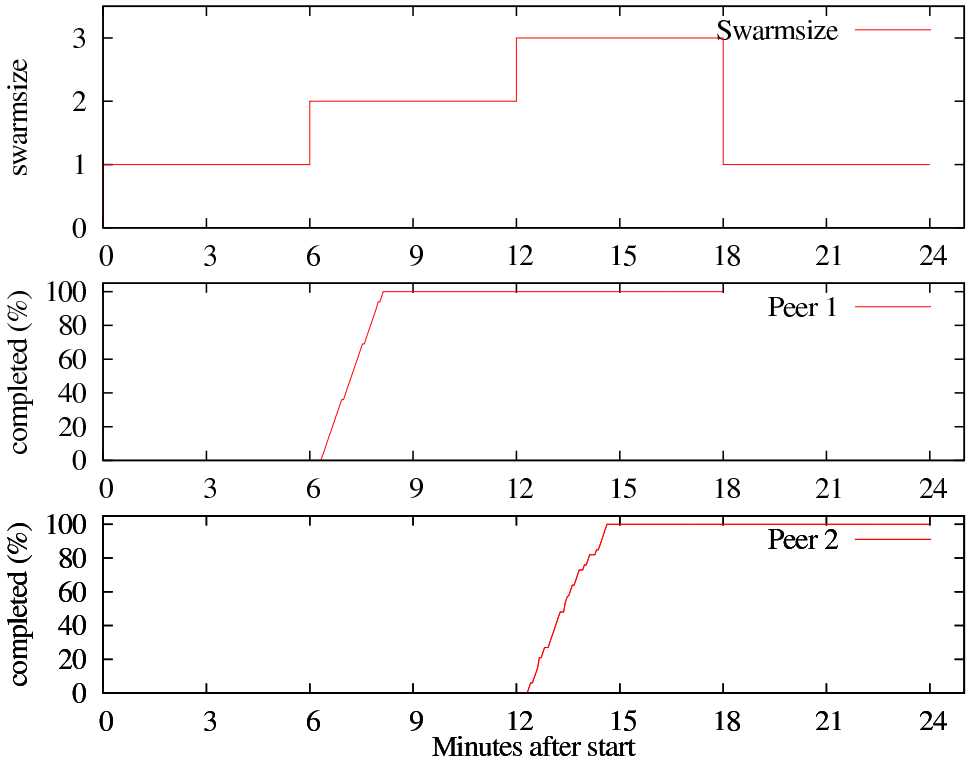

Figure 5.4: The swarm size of our example swarm (top) and the download progression of leechers peer 1 and peer 2 (bottom).

5.3.1 Trackerless content distribution

Trackerless content distribution starts with a content creator peer, which creates a torrent file without a tracker address. During the sessions of the content creator peer, this torrent file will be automatically be transfered to other peers using the BuddyCast recommendation protocol. Any Tribler peer that is online may receive a BuddyCast preference list including the newly shared trackerless torrent. For simplicity we have created the following setup.
In our setup, there are two swarms:

- Swarm A - The trackerless swarm related to torrent file A, the trackerless torrent of the content creator peer.
- Swarm B - Another swarm (with a central tracker). In this swarm the peers will have their initial contact.

Furthermore, we distinguish three peers:

- Peer 1 - The content creator Tribler peer.
- Peer 2 - An additional Tribler peer, joining swarm B.
- Peer 3 - An additional Tribler peer, joining swarm B after peer 2 has left.

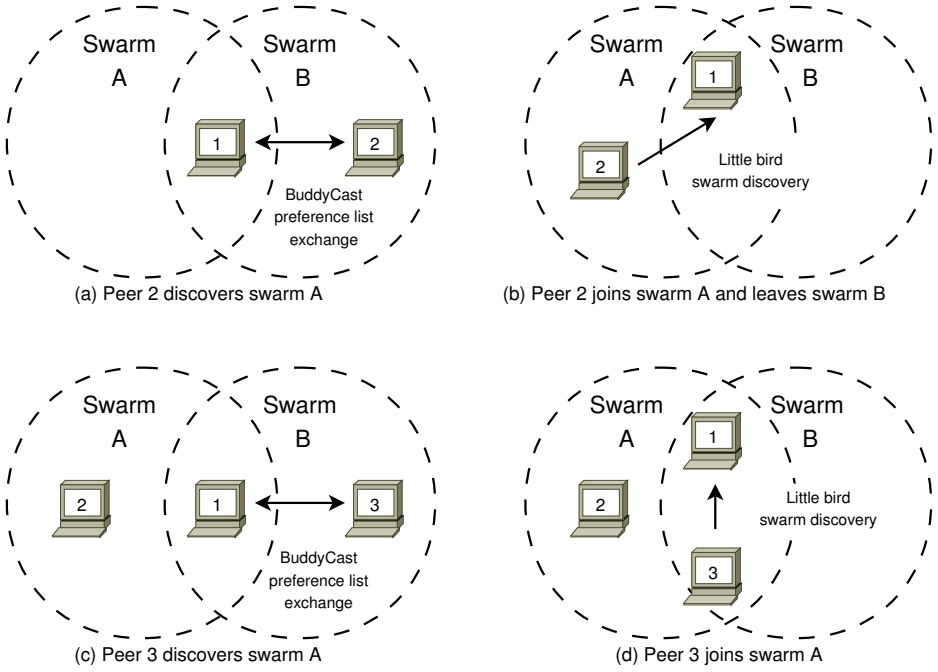

Figure 5.5: Peer 2 and peer 3 discover the content of trackerless swarm A through a BuddyCast exchange with peer 1. When they decide to join the trackerless swarm, they receive the peerlist of swarm A from peer 1 through the LITTLE BIRD protocol.

```
secover: dns task send message BUDDYCAST ('130.161.158.27', 9002)
metadata: read torrent Seilrolle.mov.torrent
"Seilrolle.mov.torrent": "seeding" (100.0%) - 1P0s0.000D u2.8K/s-d0.0K/s u64K-d0K

secover: dns task send message BUDDYCAST ('130.161.158.27', 9003)
metadata: read torrent Seilrolle.mov.torrent
"Seilrolle.mov.torrent": "seeding" (100.0%) - 2P0s0.099D u14.5K/s-d0.0K/s u5248K-d0K
```

Log File 5.3: Snippets of the log file of peer 1.

The following set of events are a typical way in which peers 2 and 3 could discover the content and the swarm members of trackerless swarm A. The four steps below are show in Figure 5.5.

Initially, peer 1 will be the only peer in swarm A, because it is the only peer that has a copy of the content and knows about its existence. Peer 1 is also bartering in swarm B, where it finds peer 2. Because both peers support the BuddyCast system, they decide to exchange preference lists (see Figure 5.5a). Peer 2 will hence find out about the content that peer 1 has injected and downloads torrent file A. We assume that the newly injected content matches the taste of peer 2 and it is recommended. Peer 2 decides to download the new content.

Because torrent A has no tracker associated with it, LITTLE BIRD will discover the swarm, as explained in Section 4.5.1. The information from BuddyCast is copied to the swarm database and peer 1 is found to be an active peer in swarm A. When peer 2 sends a getPeers request to peer 1, peer 1 responds with a peerlist containing only itself, because there are no other peers in swarm A. Peer 2 has now joined swarm A and starts bartering with peer 1. Because peer 2 has finished downloading in swarm B, it leaves that swarm and is only active in swarm A (see Figure 5.5b).

Peer 3 also finds peer 1 during the bartering process in swarm B, and they exchange preference lists (see Figure 5.5c). Peer 3 decides, just like peer 2, to download the content of trackerless swarm A. When it sends a getPeers request to peer 1, it will receive a peerlist with all peers in swarm A (in this case peer 1 and peer 2). Peer 3 can start bartering in swarm A after connecting to peer 2, and its swarm discovery is finished (see Figure 5.5d).

Note that swarm B was only added to this setup to make sure these three peers would exchange preference lists with each other. In practice, a Tribler peer exchanges its preference lists with peers in all swarms that are active. This leads to faster propagation of knowledge through the network and hence faster content discovery. To reduce the complexity of this emulation, we left all other online Tribler peers outside the scope, so that peers 1, 2 and 3 would only BuddyCast with each other.

5.3.2 Emulation results

To evaluate if the content and swarm discovery using BuddyCast and LITTLE BIRD works correctly, we have performed the events described in Section 5.3.1 with three

```
secover: dns task send message BUDDYCAST ('130.161.158.27', 9001)
DisTrackerClient(Seilrolle.mov): Found 1 torrent owners (1 not yet in swarmDB)
DisTrackerClient(Seilrolle.mov): known_peers 130.161.158.27:9001
"Seilrolle.mov.torrent": "1:17:00" (0.1%) - 0P1S0.000D u0.0K/s-d10.9K/s u0K-d64K
```

Log File 5.4: Snippets of the log file of peer 2.

```
secover: dns task send message BUDDYCAST ('130.161.158.27', 9001)
DisTrackerClient(Seilrolle.mov): Found 2 torrent owners (2 not yet in swarmDB)
DisTrackerClient(Seilrolle.mov): known_peers 130.161.158.27:9002,130.161.158.27:9001
"Seilrolle.mov.torrent": "0:39:44" (2.4%) - 1P1S0.113D u8.0K/s-d21.7K/s u368K-d1280K
```

Log File 5.5: Snippets of the log file of peer 3.

actual Tribler applications on a single machine. Swarm B is implemented using a torrent with a central BitTorrent tracker.
For our experiment, we use a text-mode Tribler version. Instead of users that click to download a certain recommended torrent, we move torrents from the recommendation directory to the download directory to download a recommended torrent.
In the log files of the three clients we can see the whole bootstrap process being executed. We have extracted the important lines from each log file and show them in Log Files 5.3, 5.4, and 5.5. The IP address in these log files (130.161.158.27) belongs to the host on which we conducted the experiment. The peers can be identified by their listening port, which is set to 9001 for peer 1, 9002 for peer 2, and 9003 for peer 3. The trackerless torrent injected by peer 1 is called *Seilrolle.mov.torrent* and forms swarm A.
In Log File 5.3, the first three lines indicate the communication with peer 2. Firstly, preference lists are exchanged by sending a BuddyCast message to peer 2. Then the torrent file of swarm A is sent to peer 2 by the meta-data handler. On the third line, the bartering statistics of peer 1 are shown. The interesting parts of this line for this analysis are:

- SeilRolle.mov.torrent - indicating that the torrent of swarm A is bartered.
- 1P0s - indicating that peer 1 is bartering with 1 leecher and 0 seeders.
- u2.8K/s-d0.0K/s - the upload and download speed of the bartering process.

We conclude that peer 1 is bartering in swarm A with peer 2. These steps are repeated during the communication with peer 3, shown in lines 4–6. Eventually, peer 1 is bartering with both peers in swarm A (indicated by the string '2P0s' in line six).
In Log Files 5.4 and 5.5 we see the same actions from the side of peers 2 and 3. Firstly, the transfer of a BuddyCast message, which results in the discovery of swarm A and the reception of torrent *Seilrolle.mov.torrent*. When the peer 2 chooses to download this torrent, it finds one *torrent owner*. We used the name torrent owner in our log files when referring to recommender peers. Peer 2 finds

that recommender peer 1 is still active in swarm A and starts bartering with it. Log File 5.4 line 4 shows its bartering statistics indicating that it is bartering with 1 seeder (peer 1) and 0 leechers. Peer 3 even finds two recommender peers (peers 1 and 2) and Log File 5.5 line 4 shows that it is bartering with 1 seeder (peer 1) and 1 leecher (peer 2).
We conclude from these results that the content and swarm discovery bootstrapping have been executed as predicted. BuddyCast manages the exchange of content and recommender peers between swarms, while the LITTLE BIRD protocol handles the exchange of swarm discovery information within the swarms. The combination of the two realizes decentralized bootstrapping without any need of the BitTorrent tracker.

5.4 General performance evaluation

We have evaluated our LITTLE BIRD swarm discovery protocol in depth using the CROWDED emulation environment presented in Section 5.2. With CROWDED we can evaluate how our protocol will perform in a real life swarm. In this section we present the details of our experiment and to what extend the protocol meets the design requirements from Chapter 2.

5.4.1 Emulation properties

For our protocol evaluation, we chose a swarm with realistic properties in size and churn, under the restriction that it could be emulated on the DAS-2 cluster in Delft

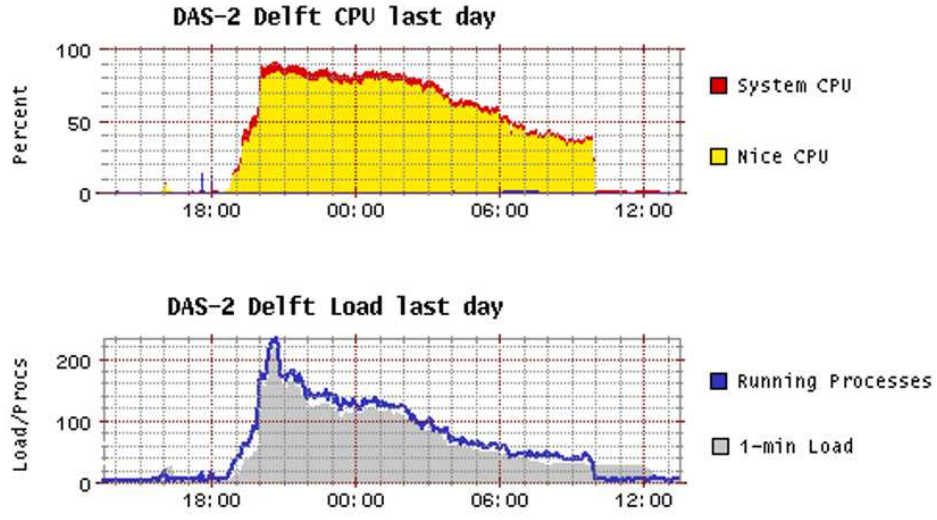

Figure 5.6: Statistics on the Delft DAS-2 cluster from Ganglia: The cpu usage of the cluster (top), and the load of the cluster (bottom).

according to the limitations described in Section 5.2. The processing power of the DAS-2 nodes restricts us to emulate BitTorrent swarms with a maximum size of 300 peers. The usage of the DAS-2 supercomputer also limits the time within which our experiments have to be conducted. We chose a swarm with 1430 unique peers and a maximum size of 305 peers, which we will emulate the first 2.5 days after creation. This is the most interesting period in the life-cycle of the swarm including the flash-crowd period, maximal swarm size, and very high churn. We have reduced the emulation time by conducting the emulation under accelerated time. Therefore, we have changed the swarm behavior and protocol configuration, so that all events happen five times faster.
We have evaluated the content discovery bootstrapping step already in the experiment in Section 5.3.1. In this experiment we will therefore skip this step and directly give each peer the contact information of a single peer in the swarm when it wants to join. This is done by a centralized BitTorrent tracker. In practice, users would receive these contacts using swarm discovery bootstrapping. When a peer has received the single peer from the centralized tracker and has sent a successful getPeers request to it, it is considered bootstrapped and included in the statistics.
We will focus on the performance of LITTLE BIRD in these emulations, and not so much on the download progression of the peers in the swarm. Still we have made a distinction between peers that start as a seeder and those that start as a leecher. The nine peers that were online on the moment that our scraper software initially found this swarm will join as seeders, while all subsequently joining peers will join as leechers.
Figure 5.6 shows the load and cpu usage of the complete DAS-2 Delft cluster during this emulation experiment, as generated by the cluster monitoring system Ganglia [32]. For this experiment 28 supercomputer nodes were used. The load graph shows that the load of the cluster follows the size of the swarm (compare to Figure 5.7). The maximum load was around 250, which is a load of almost 9 per allocated node on which 11 Tribler applications were running. This is higher than expected from Figure 5.3, because there is more activity in the Tribler applications during the flash-crowd period than during the load experiment.
The cpu usage graph shows the 'nice cpu' usage of the Tribler instances and some system cpu usage. In total, we use up to 90% of the cpu of the cluster, with 90% of the total number of nodes allocated. Hence, the processing power of the allocated nodes is almost fully used during the experiment. Both the load and cpu usage metrics show that these emulations use the full capacity of the DAS-2 cluster.

5.4.2 Swarm coverage

Swarm coverage is the part of the swarm that a peer knows, as explained in Section 2.2. We will evaluate if the LITTLE BIRD protocol manages to give peers a large enough share of the swarm in order to barter fast and flexibly.
First, we will look at the progression in swarm coverage of a single peer p. We chose one of the peers that has joined soon after the creation of the swarm and

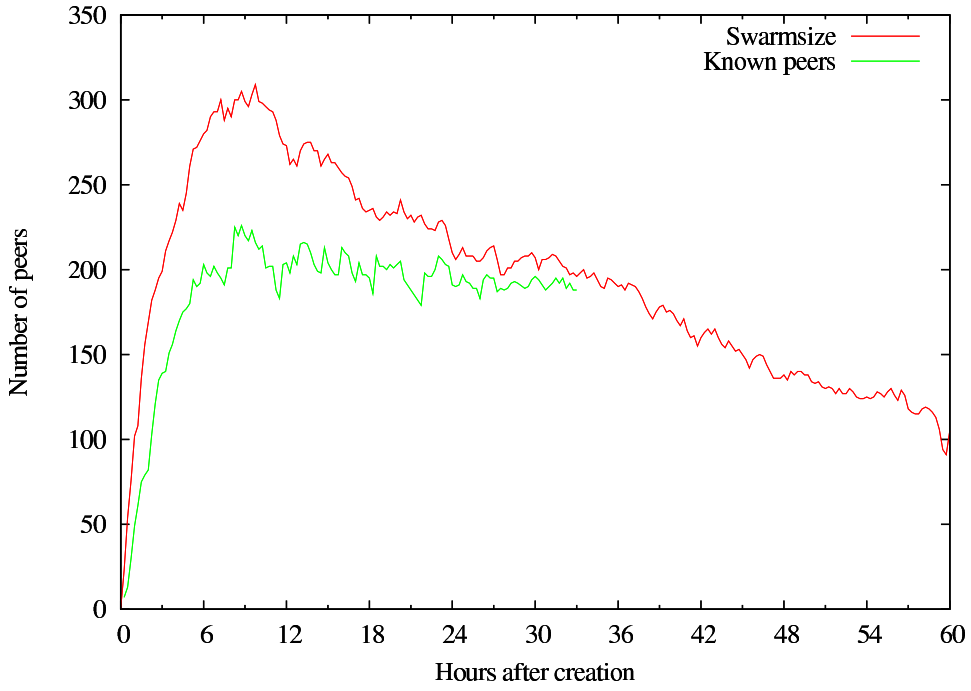

Figure 5.7: The swarm coverage of one of the initial peers in the swarm.

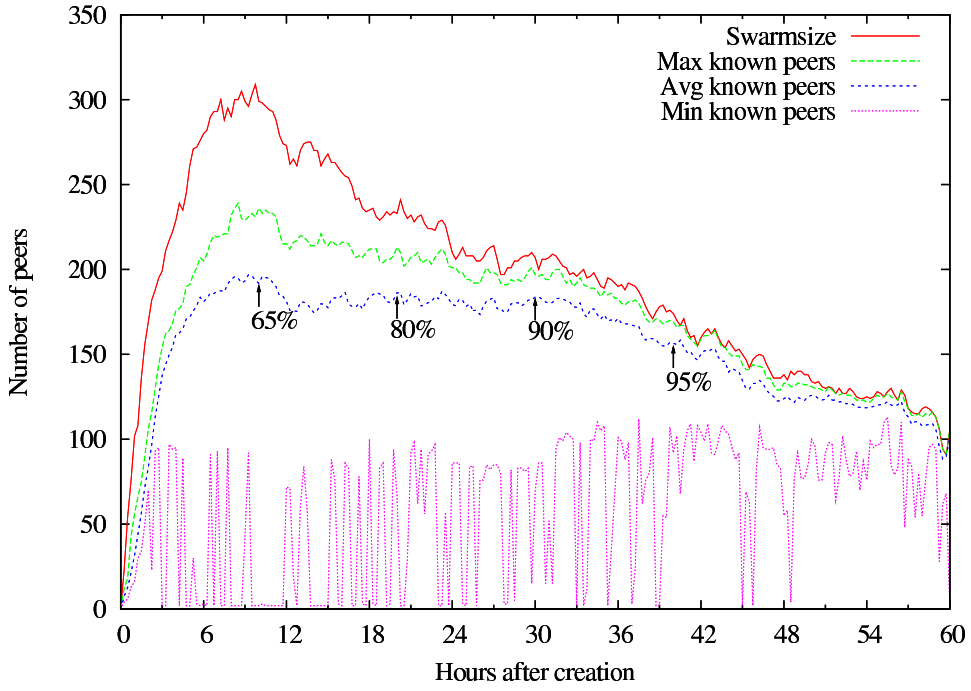

Figure 5.8: The average swarm coverage of all the peers in the swarm.

have shown its swarm coverage in Figure 5.7. During the flash-crowd period the number of known peers of peer p rises quickly with the swarm size, until it knows around 220 online peers. Then it becomes harder to find more peers, because the leave-rate begins to increase so that more peers in the swarm database of peer p become out-dated. Its number of known peers remains stable until peer p leaves the swarm after a session of 33 hours. It's swarm coverage has increased to almost 100% during this session.

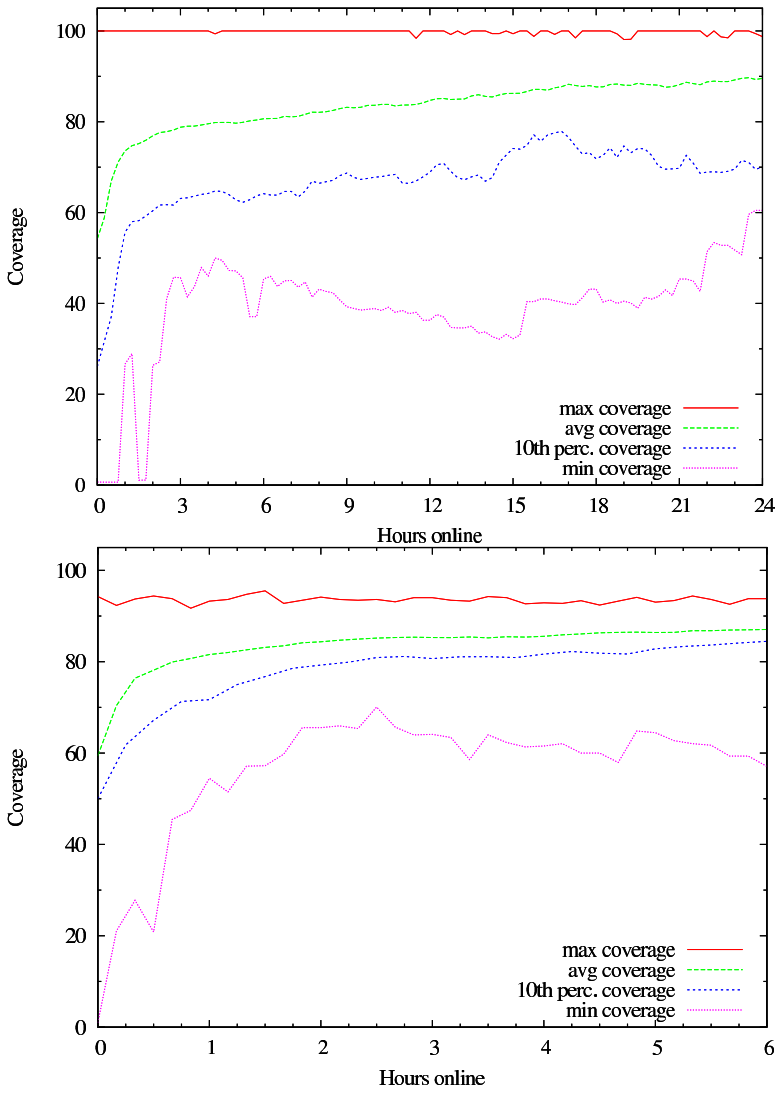

Figure 5.9: Statistics of the swarm coverage of all the peers as a function of their online time in the swarm. The top figure shows data of the accelerated emulation where heavy load reduced the quality of swarm discovery for some peers. The bottom figure shows the data of the shorter emulation without accelerated time.

When we combine the swarm discovery information from all peers in the swarm, we can analyze the behavior of the swarm coverage in the total swarm. Figure 5.8 shows the minimum, average and maximum known peers of all peers in the swarm over time. The average swarm coverage is given as percentage of the swarm size after 10, 20, 30, and 40 hours. We can conclude that the average coverage in the range of 65–95% of the swarm is more than enough for downloading. After 40 hours there are even some peers that have discovered the complete swarm.

At first sight the minimum number of known peers in the swarm seems insufficient for BitTorrent download, because it swings up and down between 0–100 known peers. In reality, the peers that cause the minimum to be very volatile are all peers that have just joined the swarm. When, after a couple of LITTLE BIRD requests, their swarm coverage rises, the minimum will rise with it. A short moment later, a new joining peer will draw the minimum swarm coverage down again. This way, the minimum number of peers keeps swinging up and down. The minimum value swings up to around 100 peers, because that is the maximum number of peers received from a single request (maximal peerlist size). We conclude that peers that have a low swarm coverage have joined the swarm recently.

To give more insight in how the swarm coverage depends on the online time of peers, we created Figure 5.9 (top), which shows the swarm coverage of all peers in the swarm as a function of the time that they have been online. For each online time, the minimal, 10th percentile, average, and maximal swarm coverage are plotted.

The average swarm coverage is already high (55%) when peers have just joined the swarm and increases quickly in the first hours. The average rises up to 90% for peers that have swarm sessions longer than a day. The maximum swarm coverage lies close to 100% for all online times, indicating that many peers manage to discover the whole swarm. There are even peers that have a 100% swarm coverage directly after joining the swarm. The swarm database realizes this instant coverage through the storage of peer information from previous sessions in this swarm, and gives the peer a full view without additional requests. This shows the added value of the swarm database, which increases the performance of Tribler by giving it a 'memory'.

The minimum swarm coverage in Figure 5.9 (top) shows that all peers that are longer online than two hours have effectively discovered the swarm. Some peers that have been shorter online still have no swarm coverage, but this is only a very small subset (21 out of the total 1430 peers need an hour; 1 peer needs 2 hours). When we look at the log files of these peers in detail, we find that they have become victims of communication errors and therefore could not discover a reasonable portion of the swarm in a short time. Only after repetitive attempts the peers manage to request half of the swarm peers.

These communication errors occur because we have increased the load on the Tribler applications by executing the protocol under accelerated time. Some peers become temporarily irresponsive by heavy load and ignore a received getPeers request. Here we see that our approach to carry out realistic emulations, also gives

detailed feedback about weaknesses in the current implementation of LITTLE BIRD. In Section 5.4.3, we will give a more detailed description of this implementation issue.
When we conduct the first 12 hours of this same emulation without accelerated time, shown in Figure 5.9 (bottom), we see that all peers easily discover the swarm. The minimal swarm coverage and 10th percentile are significantly higher in this graph and there are no peers that need additional getPeers request to discover a significant part of the swarm. We conclude that LITTLE BIRD would perform far better in reality than it does in the accelerated experiments presented in this chapter, where the protocol is tested under a heavy load.
The average swarm coverage is reduced by the fact that peers only broadcast new discovered peers when they are sure that they are connectible, which is one of our important security design steps that ensures that pollution is not spread throughout the swarm. This *check before you tell* strategy can form a bottleneck for the speed at which new IP addresses are communicated inside the swarm and thereby lower the average swarm coverage. The accelerated time of our emulations increases this effect. Still, with this limitation, LITTLE BIRD provides high quality swarm coverage. With this high swarm coverage all peers in the swarm have managed to download the file from the nine initial seeders. The only exceptions were 32 peers that had such short sessions, that they had no time to use LITTLE BIRD.
Considering the swarm coverage data resulting from our experiments, we can say the following about swarm partitioning. Figure 5.9(bottom) shows that all peers that are longer online than one hour have a swarm coverage higher than 55%. In Section 4.5.2, we stated that when all peers in a swarm have a coverage higher than 55%, partition is impossible. There we assumed that less than 5% of all peers in the swarm leave between subsequent request steps, which is realistic considering Figure 3.5. We conclude that the part of the swarm consisting of peers that are long online, will never partition using LITTLE BIRD. Recently joined peers could in theory partition from the swarm, but with the measured high swarm coverage, partition is very unlikely.

5.4.3 Scalability

To see if our protocol is scalable and efficient, we have measured the LITTLE BIRD related communication of all peers in the swarm. Each getPeers request and peerlist reply and their bandwidth usage were measured to see if LITTLE BIRD is scalable and if the load of getPeers requests is reasonably balanced over all peers.
In Figure 5.10 (top), we show the average number of getPeers requests that were answered per peer in the swarm as a function of the time. The average number of requests that were sent per peer is equal the number of answered requests. LITTLE BIRD is configured to send three getPeers requests per 10 minutes. Hence, we would expect that the average peer sends and handles 0.3 requests per minute. Figure 5.10 (top) shows that during the start of this emulation, peers had this predicted average send rate of almost 0.3 request per minute. However, after three hours, the

request rate falls and remains lower during the remainder of the experiment. Analysis of the log files show that the reduced request rate is not caused by the decision of peers to send fewer requests, but by an implementation issue that causes peers send their requests less frequently. Under the heavy load of many Tribler applications on a single DAS-2 node, and the accelerated execution of the LITTLE BIRD protocol, the peer selection component slows down significantly. In Log File

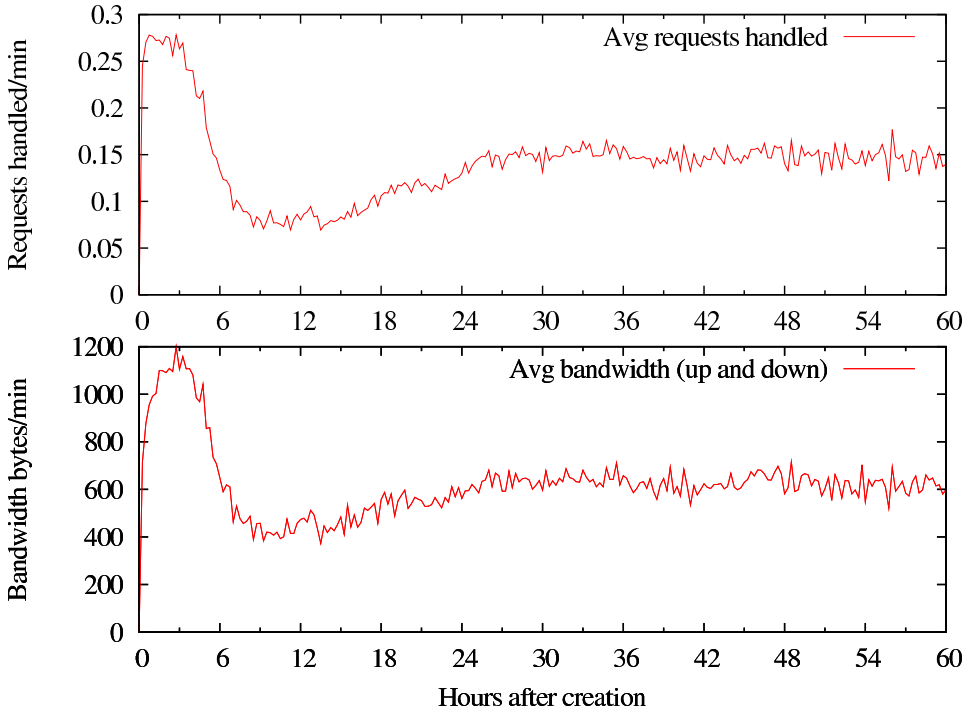

Figure 5.10: The average number of getPeers requests a peer has handled per minute (top), and the average bandwidth usage of a peer (bottom) as a function of the time.

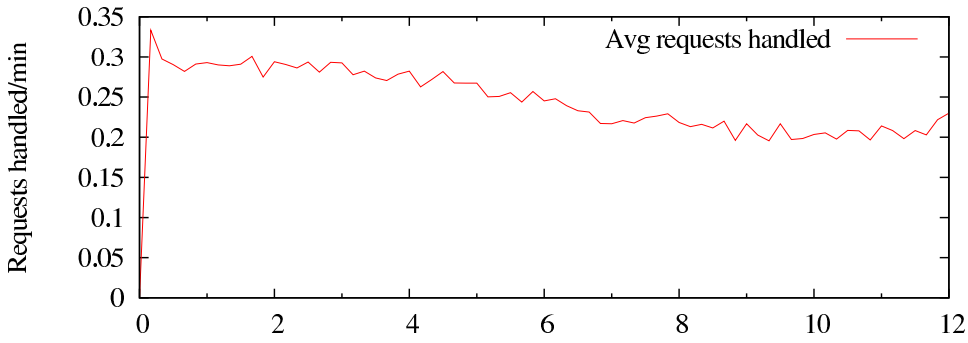

Figure 5.11: The average number of getPeers requests a peer has handled per minute when the swarm is emulated without accelerated time.

```
1161725716.115 - Peer selection step 430
1161725734.105 - Done 3 getPeers requests
1161725739.398 - Selecting peers to connect took 5.292721 seconds
1161725753.869 - Peer selection step 431
1161725819.137 - Selecting peers to connect took 65.267959 seconds
1161725831.917 - Peer selection step 432
1161725896.694 - Selecting peers to connect took 64.776068 seconds
1161725913.017 - Peer selection step 433
1161725937.324 - Selecting peers to connect took 24.307455 seconds
1161725956.040 - Peer selection step 434
1161725969.786 - Selecting peers to connect took 13.746594 seconds
1161725986.755 - Peer selection step 435
1161725999.623 - Selecting peers to connect took 12.544147 seconds
1161726016.912 - Peer selection step 436
1161726025.022 - Selecting peers to connect took 8.110421 seconds
1161726041.394 - Peer selection step 437
1161726055.328 - Selecting peers to connect took 13.933817 seconds
1161726069.977 - Peer selection step 438
1161726079.406 - Selecting peers to connect took 9.428656 seconds
1161726097.741 - Peer selection step 439
1161726105.876 - Selecting peers to connect took 8.134667 seconds
1161726118.474 - Peer selection step 440
1161726130.343 - Done 3 getPeers requests
```

Log File 5.6: Snippet of the log file of a Tribler application showing the slow execution of peer selection under heavy load. The delay between the first and second transmission of getPeers requests is here 396 seconds instead of the planned 120 seconds.

5.6, a snippet of a log file of one of the peers during the heavy load period is shown. Because we accelerated the emulation by a factor five, every 12 seconds a selection step should be executed (see also Figure 4.3). During every 10th selection step, getPeers requests are sent. In our implementation, LITTLE BIRD sleeps 12 seconds between the finishing of one connection step and the start of the following step. When the execution of a connection step takes significant time, the connection step frequency falls. The frequency with which getPeers requests are sent also falls, as this is executed every 10th connection step. This delay effect explains the average request rates of Figure 5.10.

When we repeat the first hours of this emulation without time acceleration, the average number of requests that are answered behaves as in Figure 5.11. Under these more relax circumstances, the request rate lies much closer to 0.3 requests per minute, but still there is some delay when executed under heavy load. We have shown in Figure 5.9 that this more realistic request frequency results in a higher swarm coverage.

We conclude that the robust properties of LITTLE BIRD ensure reliable swarm coverage even when the request frequency is lowered by heavy load. For future use, a less computation-intensive peer selection mechanism should be implemented, so that LITTLE BIRD is not only fully scalable in its communication, but also in the computation time of its peer selection component.

We will now compare the bandwidth usage of LITTLE BIRD with that of the central BitTorrent tracker and DHT solutions, presented in Sections 3.5.1 and 3.5.2. Figure 5.10 (bottom) shows that LITTLE BIRD uses at most 1,200 bytes per minute. When the send frequency is not lowered by a heavy load, as was the case in this emulation, the bandwidth usage will be close to this 1,200 bytes per minute during the whole experiment. Therefore, we will assume that a normal bandwidth usage for LITTLE BIRD is 1,200 bytes upload and 1,200 bytes download per minute per swarm.
The bandwidth usage of the popular swarm discovery solutions is shown in Table 5.2.

Swarm discovery solution	Bandwidth up+down (bytes/min)
BitTorrent tracker	76
LITTLE BIRD	2,400
DHT	16,850

Table 5.2: The bandwidth usage of LITTLE BIRD and two popular existing swarm discovery solutions.

The table shows that the bandwidth overhead of LITTLE BIRD lies between that of the BitTorrent tracker and the DHT solution. Important to mention is that the bandwidth usage of the BitTorrent tracker scales linearly with the number of concurrent swarms in which a user is bartering. LITTLE BIRD has also a linear scaling, although the total bandwidth utilized by answering getPeers requests for all swarms is limited by our load balancing mechanism. When the DHT solution is used for swarm discovery, only a single DHT is needed for multiple swarms. The P2P network implementation and DHT configuration determine how DHT bandwidth overhead scales with the number of swarms. We have no detailed measurements of this behavior.
The bandwidth consumed by the LITTLE BIRD protocol is mostly used for the secure handshakes through the Secure Overlay of Tribler. For instance a typical getPeers message consists of 2.4 KB handshake bandwidth and 80 Bytes of actual message bandwidth. After this handshake, the peerlist message is returned without an additional handshake. These secure handshakes allow us however to (re)identify peers, allowing us to make LITTLE BIRD secure swarm discovery.
We conclude that LITTLE BIRD realized reliable swarm discovery with a bandwidth usage seven times smaller than that of a DHT solution.

5.4.4 Load balancing

We have seen that the average bandwidth usage of LITTLE BIRD is small and limited. We will now evaluate if this bandwidth is evenly balanced over all active Tribler peers.
LITTLE BIRD load balancing works by dynamically setting an interval I that a peer has to wait before sending a subsequent request, as described in Section 4.5.3.

According to Equation 4.8, a peer will increase its interval period I above the minimum when receiving more than four requests per minute. Figure 5.10 (top) shows that in our experiment, an average peer on a certain moment never receives that much requests. There could be, however, peers that receive much more requests than the average. To analyse how well the requests are balanced over all peers, we have calculated for each peer the average number of handled requests per minute over its whole session in the swarm. The better the load balancing of LITTLE BIRD the more these averages of all peers should be similar.

Figure 5.12 shows the probability mass function of these average request frequencies of all peers. The average peer has handled 0.14 getPeers requests per minute during its session. The figure shows that the request frequencies of all peers are concentrated around the average value. Only 10% of all peers have received more than twice as much requests as the average peers with a maximum of 0.58 requests per minute, which is about four times as much as the average peer. Analysis shows that these peers that have received a relative high number of getPeers requests, have short online sessions. Peers prefer to send requests to recently joined peers, because it is likely that they can provide new swarm information.

In this single swarm situation, there are no peers that received more than four getPeers requests per minute during a substantial time. Hence, our load balancing mechanism has only occasionally been active during our experiment and is not

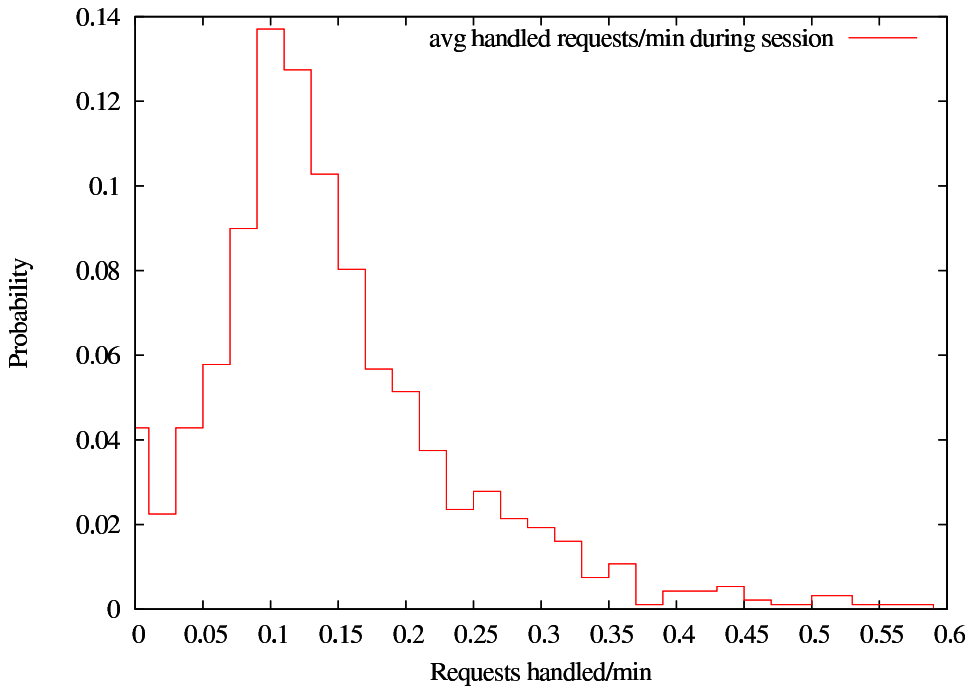

Figure 5.12: The probability mass function of the average number of getPeers requests per minute handled by a peer during its session.

needed when a peer is only active in a single swarm. However, when a peer is active in multiple swarms, the number of requests per minute will increase. Furthermore, peers will receive post-bartering requests for swarms which they have recently left. In these situations, the load balancing rules will help overloaded peers.
We conclude that the LITTLE BIRD requests are evenly divided over the peers in a swarm, resulting in a well balanced load. When the number of swarms in which a user is downloading is low, the load balancing mechanism will not play an active role in LITTLE BIRD. Otherwise, it can limit the number of requests that overloaded peers receive. We will have to evaluate the load balancing mechanism in a multiple swarm setting in order to evaluate it in more detail..

5.5 Attack resilience evaluation

In Section 4.5.5 we focused on making our protocol resilient against pollution and dDoS attacks. In this section we evaluate this resilience by conducting a new emulation in which we have configuring a part of the peers to misbehave in the protocol. These *attacker peers* will do a dDoS and pollution attack on the distributed Tribler swarm.

5.5.1 Attack scenarios

We have repeated the emulation of our example swarm in the same setting as in Section 5.4, but we have set 10% of the peers to be attackers. This means that these peers will return an *attacker* IP *list*, instead of a normal peerlist message, upon reception of a request. We have created two scenarios for the emulation of a pollution and dDoS attack. In the *pollution attack* scenario, an attacker IP list contains 100 random inconnectible IP addresses, ports and permanent identifiers. In the *dDoS attack* scenario, an attacker list is filled with a single inconnectible IP address combined with 100 random ports and permanent identifiers. This is the IP address of the victim computer at which the dDoS attack would be aimed.
In reality it would be a serious attack if 10% of all peers are malicious and try to pollute the system. Our results will show if the LITTLE BIRD protocol still manages to operate in a situation with this much pollution.
The reception of inconnectible peer information is especially harmful when a peer has just joined and has a small swarm coverage. Therefore, we use the following setting in this emulation: When a peer concludes that all of the peers it knows in the swarm are attackers, it will not send requests to these attackers anymore. Instead, it will bootstrap the swarm again using another single peer address and try to discover cooperating peers.

5.5.2 Security and integrity

We will first evaluate if LITTLE BIRD can still function with this amount of attacker peers. Peers should be able to continue their downloads as normal while under

attack. Then we will look to what extend the attackers were disguised by peers in the swarm.

We compare the average coverage of our three scenarios (no attacker peers, pollution attack, and dDoS attack) in Figure 5.13. The figure shows that during the initial flash crowd period, the swarm coverage of the scenarios does not differ significantly. After three hours, the average number of known peers in the swarms with attackers remains stable, while it still rises in the healthy swarm. This difference is never made up and the average swarm coverage remains lower in the attacker emulations.

There are two reasons for this lower swarm coverage. The first reason is that it is harder for peers to discover new peers when they receive malicious information from the attackers. A second reason is that, because an attacker peer always responds to a request with a peerlist filled with many fake IP addresses, the swarm database sizes of all peers grow. The swarm database works as a blacklist in this situation and enables peers to remember which IP addresses not to connect. However, we have seen in Section 5.4.3 that in our implementation a very full swarm database decreases the performance of LITTLE BIRD in our heavy loaded emulations. Therefore, the attack is extra harmful, as it slows down the requests rate of the peers. When we conducted an emulation of a pollution attack without acceler-

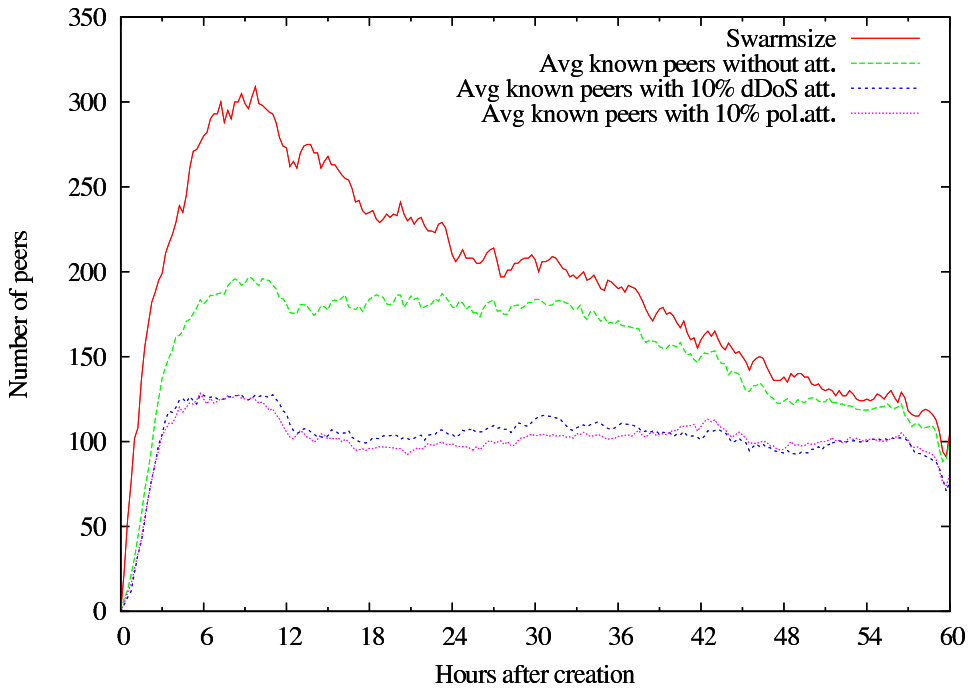

Figure 5.13: The average swarm coverage of peers in the swarm for our three scenarios: no attackers in the swarm, 10% of the swarm is a dDoS attacker peer, and 10% of the swarm is a pollution attacker peer.

ated time, we found the average swarm coverage to be higher.

Although swarm discovery is much harder when 10% of the sources are unreliable, still LITTLE BIRD manages to let most of the peers discover a sufficiently large part of the swarm without problems.

We will now look to what extend the peers in the swarm have identified correctly which peers are unreliable. We have measured the number of IP addresses that each attacker spreads through peerlists in order to evaluate the successfulness of attacker peers. Also we measured to how many of these IP addresses connections were attempted (see Figure 5.14). If the contribution mechanism in LITTLE BIRD operates effectively, peers in the swarm should conclude that the attacker peers are not contributive and reduce the number of requests to them. If peers have still sent a request to one of the attacker peers, they should distrust the IP addresses received from attackers and prefer IP addresses of existing peers.

Figure 5.14 shows that during the flash crowd period, many requests to attacker peers are sent. On this moment, new peers join the swarm and have no objections against sending requests to the attackers. Peers only attempt connections to a part of the received fake IP addresses, because LITTLE BIRD uses a defensive connection strategy. Honest peers will not forward inconnectible IP addresses to

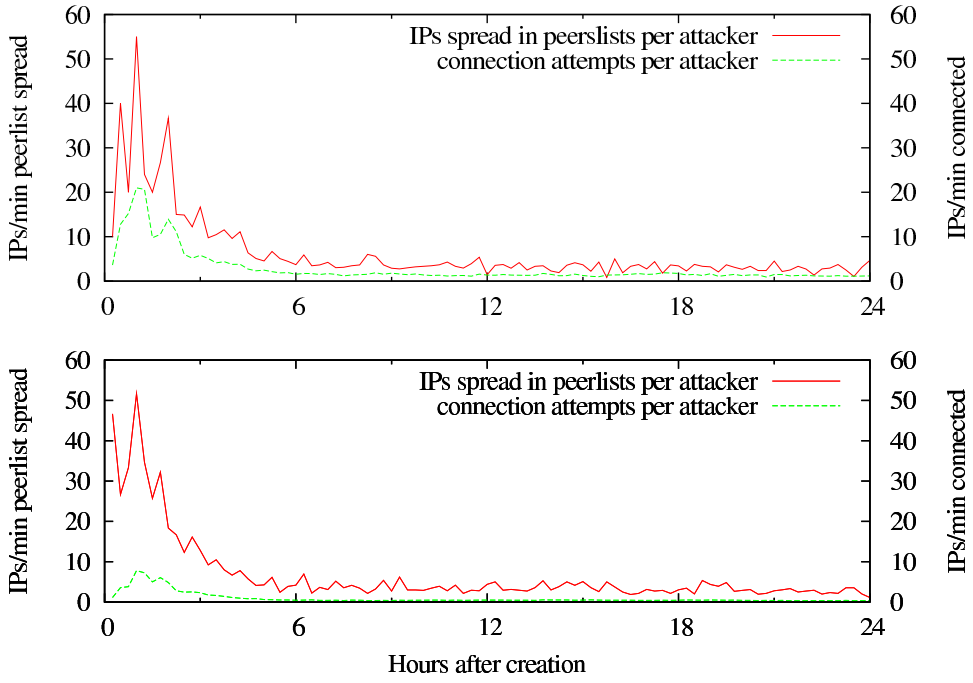

Figure 5.14: Both for the pollution attack (top) and the dDoS attack (bottom), the following information is plotted: The number of fake IPs spread in peerlists per attacker per minute and the number of connection attempts to these IPs from the other peers in the swarm per attacker per minute.

```
Contribution of 130.161.211.208:9033:
(Conn 0.20*0.70 (3/3), Bart 0.50*0.00, DT_act 0.10*1.00 (1-0/1),
DT_qu 0.20*0.48 (48/100)) = 0.3367
130.161.211.208:9033 is an attacker, with >50 unconnectible ips. Will not request it.
```

Log File 5.7: Snippet of the log file of a Tribler application during the selection process of peers to send a getPeers request. A peer with network address 130.161.211.208:9033 is found to be an attacker.

other peers, because of the *check before you tell* strategy (see Section 4.5.5). In the pollution attack scenario, peers have attempted to connect to 51.1% of the total IP addresses received from attackers. For the dDoS attack, this was only 17.4%. This shows that is costs more effort for a dDoS attacker to send fake IP addresses to a swarm than it would cost to connect to them itself. LITTLE BIRD is more resilient against dDoS attacks, because we have added an additional rule not to connect to more peers with the same IP address in a connection step, explained in Section 4.5.5. This rule creates an upper-bound of connecting to one fake IP address per minute during a dDoS attack. In our experiment, for each attacker, there are approximately 9 contributive peers. Therefore, per attacker, there are never more than 9 connection attempts per minute in the dDoS scenario.

The log files of peers in this experiment show that they have effectively disguised attackers (see Log File 5.7). An attacker is disguised when the number of received inconnectible IP addresses rises above the attacker threshold T_a, presented in Section 4.5.5. When this has not yet happened, an attacker peer is considered less contributive than other peers, but peers will still request it. Because peers defensively attempt connections to IP addresses received from attackers, the number of observed inconnectible IP addresses often stays below the threshold. Hence, LITTLE BIRD does not show a significant reduction in requests to attackers after some time.

We conclude that LITTLE BIRD is capable of realizing decentralized swarm discovery of reasonable resilience against large scale pollution and dDoS attacks. In our example setting of 10% peers that try to stop the others from discovering the swarm, LITTLE BIRD still succeeded to deliver swarm discovery. Furthermore, connection attempts to fake IP addresses are omitted when peers consider the source peer to be incontributive. Further research could optimize LITTLE BIRD, so that attacker peers are better recognized and isolated from a swarm.

Chapter 6

Conclusions and Future Work

In this chapter we give a summary of the problem that we solved and state our conclusions. Then we propose future research that can be conducted in the area of swarm discovery.

6.1 Summary and conclusions

Peer discovery, that is, finding the addresses of network members, is a general problem in P2P networks. In P2P file sharing networks, peers that download the same file have to be discovered, called *swarm discovery*, before content can be shared with these peers. In the BitTorrent system, swarms are discovered using a central tracker. This client-server solution lacks scalability and reliability. Other BitTorrent swarm discovery solutions that are more scalable lack effective security and incentive mechanisms. In order to solve this problem, we have designed and implemented a decentralized swarm discovery solution called LITTLE BIRD, with as design goals scalability and security. Furthermore, LITTLE BIRD provides peers with incentives to cooperate in the swarm discovery process.
For the evaluation of LITTLE BIRD, we have created an experimental environment called CROWDED, which enabled us to conduct large-scale trace-based emulations of swarms on the DAS-2 supercomputer. From our evaluation results, we can state our most important conclusions:

- LITTLE BIRD is *effective*. With LITTLE BIRD, peers discover a sufficiently large part of swarms to barter efficiently. All peers in our evaluation discovered more than 50% of a swarm of 305 peers within one hour. Hence, LITTLE BIRD keeps the peers in a swarm well enough connected to make swarm partitioning impossible under realistic circumstances.

- LITTLE BIRD is *scalable*. The bandwidth usage of a peer needed for LITTLE BIRD communication is 2.4 kiloByte per minute, which is a factor seven lower than the popular DHT swarm discovery solutions. The theoretical foundation of our epidemic protocol guarantees a constant message rate.

Therefore, bandwidth usage does not increase with the swarm size. The LITTLE BIRD requests are reasonably balanced over all peers, preventing that some peers have a much higher bandwidth usage than others.

- LITTLE BIRD is *decentralized*. Content can be shared and published without the need for any centralized components. LITTLE BIRD combined with the BuddyCast recommendation protocol manages decentralized content discovery and swarm discovery. The availability of download swarms is increased by caching swarm information during a period after a peer has left a download swarm.

- LITTLE BIRD is *secure*. Pollution attacks only slightly reduce the performance of LITTLE BIRD. The defensive design of LITTLE BIRD makes it unprofitable to misuse the protocol to launch a distributed denial-of-service attack against a victim computer.

6.2 Future work

During our research we have concluded that the following technical improvements should be made to LITTLE BIRD:

- The definition of the quantitative contribution of a peer should be enhanced, so that the contribution is shared over multiple swarms. Currently, the contribution is calculated for each peer per swarm, which does not give a peer incentives to help others after it has left a swarm. When a peer would gain contribution in future swarms, this incentive is created.

- Peer selection in LITTLE BIRD should be re-implemented so that the computation time no longer depends on the size of the swarm database. This could, for instance, be accomplished by storing peers in a sorted data structure and saving statistics incrementally.

- The attack resilience of LITTLE BIRD can be optimized, by developing a more sophisticated mechanism to distinguish attacker peers from contributive peers.

We believe that the functionality of LITTLE BIRD can be extended so that it can be used to discover a variety of social communities. LITTLE BIRD can then be a part of a larger mechanism for social-motivated file sharing and community management in the Tribler application. The strength of such an integrated system lies in the interaction between Tribler components. The following ideas may help to extend the effectiveness of LITTLE BIRD through interaction with other Tribler components.

Swarm discovery selection When multiple swarm discovery sources are available for a swarm (for instance, both a BitTorrent centralized tracker and LITTLE

BIRD), a peer should intelligently choose the most scalable solution and use unscalable solutions as backup. Currently, a mechanism to wisely choose the right swarm discovery source is not included in Tribler.

LITTLE BIRD **for Tribler video** The Tribler application will support the sharing of streaming video and video on demand in addition to simply sharing files. It would be interesting to extend the LITTLE BIRD protocol to realize the discovery of the swarms for these new P2P media.

Swarm pre-discovery The swarm discovery bootstrap mechanism could be optimized by pro-actively discovering highly recommended swarms before a user actually needs them, called *swarm pre-discovery*. This will cost only moderate overhead and will increase the probability of successful bootstrapping, because directly after a preference list is received from a recommender peer, it is probably online and active in the swarm. With swarm pre-discovery, not only taste buddies and content are automatically discovered, but also the favorite swarms of the user.

Additional BuddyCast information A new version of the BuddyCast protocol should include information about when recommender peers were still active in the swarms that they recommend. This enables users to estimate the probability of swarm discovery success with the help of a recommender peer.

Firewall puncturing Swarm coverage could be improved by having better knowledge of the connectivity of peers. Currently, peers that are discovered through incoming connections do not have to be connectible themselves due to firewalls or NAT traversal. Therefore, a peer does not distribute the addresses of incoming connection peers to other peers through our protocol. Only after having started an out-going connection to the peer, will it be included in the swarm database. This outgoing connection might never be established if the incoming connection is retained.

Bibliography

[1] Another bittorrent client (abc). http://sf.net/projects/pingpong-abc.

[2] K. Abener and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. of the 10th Int. Conf. of Information and Knowledge Management*, Atlanta, GA, 2001.

[3] E. Adar and B. Huberman. Free riding on gnutella. Technical report, Xerox PARC, August 2000.

[4] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on cooperation in bittorrent communities. In *P2PECON '05: Proc. of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 111–115, New York, NY, USA, 2005. ACM Press.

[5] Azureus java bittorrent client. http://azureus.sourceforge.net/.

[6] N.T.J. Bailey. *The mathematical theory of infectious diseases and its applications*. Griffin, 1975.

[7] Beautifulsoup, python html parsing library. http://www.crummy.com/software/BeautifulSoup.

[8] Bitcomet, a c++ bittorrent client. http://www.bitcomet.com/.

[9] Bittorrent multi-tracker specification. http://www.bittornado.com/docs/multitracker-spec.txt.

[10] Bittorrent trackerless dht protocol. http://www.bittorrent.org/Draft_DHT_protocol.html.

[11] Bittorrent trackerless introduction. http://www.bittorrent.com/trackerless.myt.

[12] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Comm. of the ACM*, 13(7):422–426, 1970.

[13] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proc. of the 40th Annual Allerton Conference on Communication, Control and Computing*, pages 636 – 646, Illinios, USA, 2002.

[14] J. Brustoloni. Protecting electronic commerce from distributed denial-of-service attacks. In *WWW '02: Proc. of the 11th int. conf. on World Wide Web*, pages 553–561, Honolulu, Hawaii, USA, 2002.

[15] Cerri, Ghioni, Paraboschi, and Tiraboschi. Id mapping attacks in p2p networks. In *Proc. of the Global Telecommunications Conference, n2005. GLOBECOM '05. IEEE*, pages 1785–1790, 2005.

[16] N. Christin, A.S. Weigend, and J. Chuang. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *EC '05: Proc. of the 6th ACM conference on Electronic commerce*, pages 68–77, New York, NY, USA, 2005. ACM Press.

[17] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[18] Clientcookie, python http-cookie library. http://wwwsearch.sourceforge.net/ClientCookie/.

[19] B. Cohen. Bittorent protocol. http://www.bittorrent.org/protocol.html.
[20] B. Cohen. Incentives build robustness in bittorrent. In *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.
[21] Computer load. http://en.wikipedia.org/wiki/Load_(computing).
[22] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. of the 13th USENIX Security Symposium*, August 2004.
[23] Distributed asci supercomputer 2. http://www.cs.vu.nl/das2/.
[24] emule, a p2p filesharing client for the kad and edonkey networks. http://www.emule-project.net.
[25] Etree, bittorrent community tracker. http://bt.etree.org.
[26] P. Eugster, R. Guerraoui, A.M. Kermarrec, and L. Massoulie. From epidemics to distributed computing. *IEEE Computer*, 37(5):60–67, May 2004.
[27] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *EC '04: Proc. of the 5th ACM Conf. on Electronic commerce*, pages 102–111, New York, NY, USA, 2004.
[28] Filelist.org bittorrent community. http://www.filelist.org.
[29] Flickr, photo sharing. http://www.flickr.com.
[30] Freeband / i-share. http://www.freeband.nl.
[31] Friendster, social community. http://www.friendster.com.
[32] Ganglia cluster monitoring system. http://ganglia.sourceforge.net.
[33] P. Garbacki, A. Iosup, D.H.J. Epema, and M. van Steen. 2fast: Collaborative downloads in p2p networks. In *Proc. of 6th IEEE Int. Conference on Peer-to-Peer Computing (P2P2006)*, Cambridge, UK, September 2006.
[34] Gnutella protocol development. http://www.the-gdf.org/wiki/index.php.
[35] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge. Incentives for sharing in peer-to-peer networks. In *Proc. of 3rd ACM Conf. on Electronic Commerce*, Tampa, FL, Octobre 2001.
[36] S. Gotz, S. Rieche, and K. Wehrle. *Peer-to-Peer Systems and Applications*. Springer, 2005. Chapter 8.
[37] Hyves, social community. http://www.hyves.nl.
[38] I2p, an anonymizing network layer. http://www.i2p.net/.
[39] Icq instant messenger, 2005. http://www.icq.com.
[40] A. Iosup and D.H.J.Epema. Grenchmark: A framework for analyzing, testing, and comparing grids. In *the 6th IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGrid'06)*, 2005.
[41] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *in Proc. PAM'04*, Antibes Juan-les-Pins, France, April 2004.
[42] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proc. WWW 2003*, Budypest, Hungary, May 2003.
[43] F. Kargl, J. Maier, and M. Weber. Protecting web servers from distributed denial of service attacks. In *WWW '01: Proc. of the 10th int. conf. on World Wide Web*, pages 514–524, Hong Kong, 2001.
[44] Sharman networks ltd. kazaa media desktop. http://www.kazaa.com.
[45] D. Kügler. An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks. In Roger Dingledine, editor, *Proc. of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.
[46] A. Legout, G. Urvoy-Keller, and P. Michiardi. Understanding bittorrent: An experimental perspective. Technical report, INRIA, Sophia Antipolis, November 2005.

[47] J. Li, J. Stribling, T.M. Gil, R. Morris, and M.F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *IPTPS '04: Revised Selected Papers from the 3th International Workshop on Peer-to-Peer Systems*, page 875. Springer Berlin, 2004.

[48] J. Liang, R. Kumar, Y. Xi, and K.W. Ross. Pollution in p2p file sharing systems. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume vol. 2, pages 1174 – 1185. IEEE, march 2005.

[49] Limewire official homepage. http://www.limewire.com.

[50] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM Press.

[51] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[52] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[53] Mininova, bittorrent community and search engine. http://www.mininova.org.

[54] H.H. Mohamed and D.H.J. Epema. Experiences with the koala co-allocating scheduler in multiclusters. In *Proc. of the 5th IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2005)*, may 2005.

[55] D. Moore, C. Shannon, D.J. Brown, G.M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, 2006.

[56] Msn (microsoft network) messenger. http://join.msn.com/messenger/overview/.

[57] A. Nandi, T.-W. Ngan, A. Singh, P. Druschel, and D.S. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Proc. of Middleware*, Grenoble, France, November 2005.

[58] N. Naoumov and K. Ross. Exploiting p2p systems for ddos attacks. In *InfoScale '06: Prec. of the 1st int. conf. on Scalable information systems*, page 47, New York, NY, USA, 2006.

[59] BBC News. File swappers fight back, May 11, 2003. http://news.bbc.co.uk/1/hi/technology/3013065.stm.

[60] Orkut, online social community. http://www.orkut.com.

[61] Azureus peer exchange. http://azureus.aelitis.com/wiki/index.php/Peer_Exchange.

[62] The piratebay, bittorrent community and tracker. http://www.thepiratebay.org.

[63] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *IPTPS '05: Proc. of the 4th International Workshop on Peer-to-Peer Systems*, 2005.

[64] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M.J.T. Reinders, M. van Steen, and H.J. Sips. Tribler: A social-based peer-to-peer system. In *IPTPS '06: Proc. of the 5th International Workshop on Peer-to-Peer Systems*, 2006.

[65] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *SIGCOMM '04: Proc. of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 367–378, New York, NY, USA, 2004. ACM Press.

[66] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[67] J. Roozenburg. A literature survey on bloom filters. Research Assignment, November 2005.

[68] A.I.T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[69] J. Shneidman and D. Parkes. Rationality and self-interest in peer to peer networks. In *Proc. of 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[70] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proc. of the 2001 conference on Applications, Technologies, Architectures, and Protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[71] Tribler bittorrent client. http://www.tribler.org.

[72] Tribler streaming media client. http://www.tribler.org/test_streaming.

[73] G. van der Ent. Symptop: A simulation toolkit for peer-to-peer networks. MSc thesis, Delft University of Technology, June 2005.

[74] V. Vishnumurthy, S. Chandrakumar, and E.G. Sirer. Karma: A secure economic frame-work for p2p resource sharing. In *Workshop on Econ. of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[75] J. Wang, J.A. Pouwelse, R. Lagendijk, and M.J.T. Reinders. Distributed collaborative filtering for peer-to-peer file sharing systems. In *Proc. of the 21st Annual ACM Symposium on Applied Computing*, april 2006.

[76] Wikipedia. Online collaborative encyclopedia. http://www.wikipedia.org.

[77] D.K.Y. Yau, J.C.S. Lui, F. Liang, and Y. Yam. Defending agains distributed denial-of-service attacks with max-min fair server-centric router throttles. *IEEE/ACM Trans. Netw.*, 13(1):29–42, 2005.

[78] Youtube, online video streaming and sharing service. http://www.st.ewi.tudelft.nl/koala.

[79] N. Zennström and J. Friis. Skype. http://www.skype.com.

Appendix A

LITTLE BIRD Specifications

In this appendix we give the specifications of the LITTLE BIRD protocol. In Section A.1, we give an overview of the LITTLE BIRD source code. In Section A.2, the LITTLE BIRD execution arguments are described. In Section A.3, we describe the formats of the LITTLE BIRD messages. In Section A.4, we give the swarm database architecture and the statistics of peers that are stored.

A.1 LITTLE BIRD source code

In this section, we will give an overview over the python source code that we have added to Tribler in order to implement the LITTLE BIRD protocol. We used the source code of Tribler version 3.3.4 as a basis for our implementation. Although in this thesis we have referred to LITTLE BIRD as a decentralize swarm discovery protocol, in the naming of the source code files, we called our implementation a distributed tracker, abbreviated to *disTracker*.

Tribler/DisTracker This is the directory where all source code resides that is related to the LITTLE BIRD protocol.

Tribler/DisTracker/DisTracker.py The main file of LITTLE BIRD. This file contains three classes:

- DisTracker class - Parent class of the other two classes containing basic functionality.
- DisTrackerClient class - Manages the peer acquisition and peer selection (see Section 4.3). For each active torrent, a DisTrackerClient instance is made.
- DisTrackerServer class - Manages the responses to incoming getPeers requests. There is only one instance of this class.

Tribler/DisTracker/DisTrackerHandler.py This file contains the Secure overlay message handler for the LITTLE BIRD protocol. The actual message handling is delegated to the DisTrackerServer class.

Tribler/DisTracker/SwarmReliability.py This file calculates the level of contribution of peers, as described in Section 4.4.

Tribler/DisTracker/DistrackerDB.py This file performs some specific swarm database tasks.

Tribler/DisTracker/bloom.py This file contains classes to create and use Bloom filters.

Tribler/DisTracker/TrackerStats.py This file collects statistics of the operation of the LITTLE BIRD protocol. They are used on the statistics panel.

Tribler/DisTracker/TrackerPanel.py The swarm discovery statistics window, described in Section 4.3.4.

Tribler/DisTracker/DisTrackerAttacker.py We created this file for testing, to manage the creation of malicious peerlist messages and simulate a pollution and dDoS attack on the LITTLE BIRD protocol.

Tribler/Cache/CacheDBHandler.py In this file we added the SwarmDBHandler class, that manages all data storage related to LITTLE BIRD.

Tribler/Cache/cachedb.py In this file we added the four swarm databases described in Appendix A.4.

We performed many other small changes to the Tribler source code, which are all accompanied by comments describing their relation to LITTLE BIRD.

A.2 Tribler execution arguments

To add the LITTLE BIRD functionality to Tribler, we added the following execution arguments to Tribler. They were mostly used for the evaluation process of LITTLE BIRD.

- -distracker <**on**|**off**> Start Tribler with or without LITTLE BIRD support (defaults to *on*).

- -distracker_accelerate <**arg**> Start Tribler with the LITTLE BIRD protocol accelerated. This increases the getPeers request frequency and all other LITTLE BIRD timing by a factor *arg* (defaults to 1.0).

- -disable_central_tracker <**on**|**off**> Disable all connections to a BitTorrent tracker to let Tribler depend only on the LITTLE BIRD protocol (defaults to *off*).

- -distracker_test <**on**|**off**> Disable connections to a BitTorrent tracker after bootstrapping, i.e., when the local peer knows more than one reliable Tribler peer in a swarm (defaults to *off*).

- -distracker_attacker <off|dDoS|pollution> Upon a request, return an erroneous peerlist instead of a LITTLE BIRD peerlist. The *dDoS* option will return a peerlist with 100 peer addresses, created by taking a single random IP address combined with random ports. The *pollution* option gives 100 random IP addresses and ports (defaults to *off*).

A.3 Message formats

In this section, we give a technical description of the LITTLE BIRD message formats. There are two types of messages, the *getPeers* message, which is a request for peers in a swarm, and the *peerlist* message, the response containing peer information. This section is set-up in a similar format as all Tribler design notes [71].

A.3.1 Getpeers message

If a Tribler application wants to request which peers are in a certain swarm, it will send a getPeers message. Such a request is done over the Secure overlay to a Tribler peer. A getPeers message has the following parameters, which are sent in a b-encoded dictionary:

- info_hash: The 20 byte SHA1 hash of the torrent/swarm that the requester wants a peerlist of (same as BitTorrent tracker).
- numwant: The number of ip/port tuples of peers that the requester wants in the peerlist (same as BitTorrent tracker).
- port: The listening port of the requester, so that it can be added to the swarm database of the responder.
- compact: Maps on 0,1,[empty] (same as 0). Indicates if the peer information should be returned in the compact format.
- bloom_filter: A b-encoded dictionary containing a Bloom filter with all ip/port tuples currently known to the requester. These peers will be excluded from the peerlist. This field can also be left empty, which results in a list of random ip/ports of peers in the swarm, just like the BitTorrent tracker.
  - num_elements - the number of values in the Bloom filter (integer)
  - num_hash - the number of hash values (integer)
  - hash_type - string denoting the hashed method of elements ('SHA1-concat')
  - salt - string that is hashed with the elements.
  - data - the bit array of the Bloom filter in string format.

- event: Maps on started, completed, stopped, [empty]. Presents the event of the requester (same as BitTorrent tracker, currently unused in LITTLE BIRD).

A.3.2 Peerlist message

A Tribler application will respond to a getPeers message by sending a peerlist message to the requester. The permid field will be added for Tribler peers only, giving their permanent identifiers. When an error occurs, a dictionary with the key 'failure reason' is given that maps on a readable string denoting the reason of failure (same as BitTorrent tracker). Otherwise, a peerlist message consists of the following b-encoded dictionary:

- info_hash - The 20 byte SHA1 hash of the torrent/swarm that the requester wants a peerlist of (needed due to asynchronous communication through the Tribler Secure Overlay).
- interval - (integer) The number of minutes that the requester should wait before its next request (same as BitTorrent tracker).
- active - (either 0 or 1) Indicates if the responder is actively bartering. Inactive responders are either bootstrapping or in the post-bartering period (see Section 4.5.1).
- peers - A list of dictionaries per peer, that have the following keys:
  - peer id - The self-selected ID of the peer
  - ip - The ip address of the peer
  - port - The listening port of the peer
  - permid - (optionally if this peer is found to be compatible with Tribler protocol). Maps to the permid of the Tribler peer.

If the request had the compact parameter set in its getPeers message, the peers will be returned in the following compact form:

- peers - A string of concatenated peer informations, with for each peer:
  - ip address - (4 chars)
  - port - (2 chars)
  - permid_length - (1 char). This byte indicates the length in bytes of the following permid. If no permid is included, it is 0x00
  - permid - (permid_length chars) The permanent identifier of the Tribler peer.

A.4 Swarm database

In this section, we will describe in what format the swarm peer statistics are stored in the swarm database. All swarm discovery related data storage goes through the SwarmDBHandler class. Under this database handler, there are actually four databases, namely the SwarmDB, SwarmpeerDB, SwarmpeerprobsDB, and SwarmsubmittersDB. These databases are implemented in the cachedb.py file. We will describe each of them below.
Most of the stored statistics have a particular purpose in the LITTLE BIRD protocol. To clarify this, we define six categories in Table A.1 for which peer statistics can be stored. In the description of the databases below, the category is indicated between brackets (for instance [1]).

Statistics category	Used for:	Explained in Section
[1]	Contribution/connectivity	4.4.1
[2]	Contribution/Bartering activity	4.4.2
[3]	Contribution/Swarm discovery activity	4.4.3
[4]	Contribution/Swarm discovery quality	4.4.4
[5]	Contribution inheritance	4.4.5
[6]	Load balancing	4.5.3

Table A.1: Statistics categories to link peer statistics in the swarm database to the LITTLE BIRD protocol.

A.4.1 SwarmDB

This database stores which peers are in which swarms.

Key

info_hash - the identifier of the swarm.

Value

Set[(ip, port)*] - A set of the network addresses of peers in the swarm.

A.4.2 SwarmpeerDB

The SwarmpeerDB database stores the non-swarm specific properties of a peer.

Key

(ip, port) - the network address tuple of the peer.

Value

A dictionary with the following properties of the peer:

- peer id - The BitTorrent identifier of the peer.
- permid - The Tribler permanent identifier of the peer (optional).
- dt_bit - A flag indicating that this peer probably supports LITTLE BIRD (if permid is set, it is sure).
- so_bit - A flag indicating that this peer probably supports Tribler secure overlay (if permid is set, it is sure).
- down_last_hour [2] - List of (bytes, timestamp) tuples, containing the amount of downloaded data of the last hour.
- submitters [5] - List of permanent identifiers of the source peers of this peer (see Section 4.4). If we received this peer from a centralized tracker, the string 'ct' is included in the list.

A.4.3 SwarmPeerPropsDB

The SwarmpeerPropsDB database stores the swarm specific properties of a peer.

Key

(ip, port, info_hash) - A tuple of a network address combined with a swarm identifier. This identifies a peer in a particular swarm.

Value

A dictionary with the following properties of the peer:

- connected [1] - List of timestamps of last 3 hours of successful connections.
- connection_attempts [1] - List of timestamps of last 3 hours of connections attempts.
- active - This peer is active in the swarm (needed to barter or give in peerlist).
- last_seen [1] - Timestamp of last connection.
- last_tried [1]- Timestamp of last connection attempt.
- requests [3] - Number of swarm discovery requests the local peer has send to this peer.
- responses [3] - Number of responses to swarm discovery requests.

- error_resp [3] - Number of erroneous responses (very late or unrequested).
- last_request [6]- Timestamp of last swarm discovery request that the local peer has sent.
- last_incoming_request [6] - Timestamp of last incoming request from this peer.
- last_received - Timestamp of last reception of this peer.
- next_request [6]- Timestamp indicating when the next swarm discovery request is wanted according to the interval.
- next_incoming_request [6] - Timestamp indicating when this peer can request us again.

A.4.4 SwarmSubmittersDB

The SwarmSubmittersDB is used to store which peer a source peers has sent in peerlist messages.

Key

permid - The permanent identifier of this Tribler peer.

Value

Set[(ip, port)*] [4] - Set of (ip,port) tuples of peers that this Tribler peer has sent.

Appendix B

Overview of Measurements and Software

To make our Filelist BitTorrent measurements available to other researchers in the P2P field, we have produced a short and clear overview of the measurement data and software, and where they reside.

B.1 Overview of swarm measurements

B.1.1 Scraping software

Location: superstorage3.das2.ewi.tudelft.nl:/home/jelle/filelist
Main class: main.py
This scraping software was used to generate the raw filelist data on which our measurements are based. It uses the ClientCookie library [18] to log-in the Filelist.org website and the BeautifulSoup library [7] for HTML-parsing. Furthermore, it uses a configurable number of download threads to speed up website download. The downloaded pages are compressed and stored in a directory structure described in the next section.

B.1.2 Raw Filelist measurement data

Location: superstorage3.das2.ewi.tudelft.nl:/home/jelle/filelist_tracedata
This is the raw measurement data generated by our scraping software. Some details about this measurement data collection were already given in Section 3.3.2. The raw data of the Filelist community consist of *torrentlists* and *swarm details*, stored in the */torrentlists* and */torrents* directories respectively.

Torrentlists These lists describe all active torrents and the number of seeders and leechers in the related swarm. In the torrentlists directory the following files are stored:

- YYYYMMDD.zip - For each day that we have scraped, a zip-archive of all compressed torrent list scraped on the particular day in HTML format. The filename is a timestamp formatted as shown.

Swarm details For each swarm that was active in the Filelist community during our measurements, a directory was created, named after the info-hash of the particular swarm. In total 4,027 swarm directories are stored. In each swarm details directory the following files are stored:

- torrent.zip - The compressed *.torrent* file of the swarm
- info.txt - An information file, containing the creation date and torrent name.
- YYYYMMDD.zip - For each day that we have scraped statistics of this swarm, a zip-archive of all compressed swarm member lists of the particular day in HTML format. The filename is a timestamp formatted as shown.

B.1.3 Swarm churn files

Location data: superstorage3.das2.ewi.tudelft.nl:/home/jelle/filelist/swarm/churnFiles
Python tool: superstorage3.das2.ewi.tudelft.nl:/home/jelle/filelist/swarm/createChurnFiles.py

From the raw measurement data, we created for each swarm a file that describes its join and leave rates, refered to as a churn file. In the churnFiles directory for each scraped swarm, a churn file is stored. In a churn file, the join and leave rate are stored for each *churn period* between two subsequent swarm measurements. Churn files have the following format: on each line the following values separated by tabs:

- timestamp - Human readable timestamp.
- begin_time - Timestamp in seconds since epoch-format of the begin of the churn period.
- end_time - Timestamp in seconds since epoch-format of the end of the churn period.
- begin_time_rel - Timestamp in seconds since swarm creation of the begin of the churn period.
- end_time_rel - Timestamp in seconds since swarm creation of the end of the churn period.
- joins - Number of joining peers in the churn period.
- leaves - Number of leaving peers in the churn period.
- joins_min - The number of joining peers per minute.

- leaves_min - The number of leaving peers per minute.
- joins_min_smooth - A smoothed version of joins_min.
- leaves_min_smooth - A smoothed version of leaves_min.
- swarm_size - The size of the swarm in the beginning of the churn period.

Per swarm a data file and plot of its size and churn.

B.1.4 Peer behavior files

Location data: superstorage3.das2.ewi.tudelft.nl:/home/jelle/filelist/swarm/peerFiles
Python tool: superstorage3.das2.ewi.tudelft.nl:/home/jelle/filelist/swarm/createPeerFiles.py

From the raw measurement data, from each swarm we have created a directory named after its info_hash. In this directory, we have create a peer behavior file for each peer that has been active in the swarm. The files have the following format: on each line the following values separated by tabs:

- time_since_epoch - Timestamp in seconds since epoch-format.
- timestamp - Human readable timestamp.
- online - (0 or 1) Indicates if this user was online at this moment.
- connectible - (0 or 1) Indicated if this user was connectible through an open listening port.
- up - (kiloBytes) The amount of data uploaded since connected to the swarm.
- up_rate - (kb/s) The average upload speed of this user, calculated from its tracker requests.
- down - (kiloBytes) The amount of data downloaded since connected to the swarm.
- down_rate - (kb/s) The average download speed of this user, calculated from its tracker requests.
- ratio - The sharing-ratio of this user since connected to the swarm.
- complete - (%) The percentage of the content that this user has downloaded.
- connected - (minutes) The time that this user is connected to the swarm.
- idle - (minutes) The time since the last tracker request.
- client - A string indicating what BitTorrent client this user is using.