Latency-driven BitTorrent

Marco Slot

Student number: 1397117

Specialization: Parallel and Distributed Computer Systems
Department of Computer Science
Faculty of Sciences
Vrije Universiteit

Thesis submitted for the Degree of Master of Science
at the
Vrije Universiteit

· July 2008 ·

Supervisors:
dr. Guillaume Pierre
dr. Paolo Costa

Abstract

In recent years BitTorrent has become a notorious contributor to Internet traffic. Not only is BitTorrent responsible for over one third of all Internet traffic, but an immoderate amount of it is expensive cross-ISP or even inter-continental traffic. Much of BitTorrent's long-distance traffic is due to its random selection of peers, which can cause connected peers to be at very different locations. This causes inefficient network usage and harms client-perceived performance.

In this paper we present the design and evaluation of latency-driven BitTorrent, our approach to bias BitTorrent communication towards nearby peers. Unlike previous approaches we do this without requiring any additional infrastructure or patches to client software. A small number of cooperating BitTorrent trackers can easily deploy our system. Evaluating our approach through simulation and Planet-Lab deployment we find average and median reductions in download time of up to 25%. We find we can maintain or improve our gains in a rapidly growing network by controlling the peer sample size. We also show we reduce traffic cost with 12% less traffic going over global (transit) networks and more traffic going over short network paths.

Contents

Chapter 1

Introduction

With the arrival of peer-to-peer networks the Internet traffic landscape has radically changed. Long-distance traffic has surged as clients actively exchange data with peers from all over the world. End-user ISPs, previously accustomed to download-oriented applications, are suddenly faced with large volumes of outgoing, global traffic. The impact is such that some fear the Internet will run into serious capacity problems within a few years [CNE].

The most significant traffic contributor today is BitTorrent, a popular peer-to-peer application for the distribution of large files. When participating in a BitTorrent network, clients exchange data with randomly chosen peers. This approach has some beneficial properties, but it also causes many neighbours to be geographically far apart. As a result BitTorrent clients generate high volumes of long-distance traffic. To individual clients overall performance is suboptimal as long network paths imply higher latency, packet loss, and more bandwidth bottlenecks. At the same time ISPs are forced to buy transit from global networks, driving up their operational costs. In effect both users and ISPs have a strong mutual benefit in improving the locality of BitTorrent traffic.

A simple example is shown in figures 1.1 and 1.2. In the first figure peers are connected to 3 randomly chosen neighbours. Many peers have intercontinental connections despite other peers being nearby. Especially striking in this example are the peers in Brazil, that are unlikely to be connected since the majority of peers are on other continents. In the second figure we connect the peers to the 2 nearest peers and 1 random peer. Intercontinental connections are strongly reduced, and peers are connected over shorter paths.

Several approaches to bias BitTorrent traffic based on network conditions and properties currently exist, but they are unlikely to be deployed at a large scale due to their difficult applicability. P4P [XKLS07] requires ISPs to provide detailed information on network conditions to peers, while Madhyastha et al. [MIP+06] make use of a large-scale PlanetLab based infrastructure. Such network *oracles* may become available to a wide audience at some point, but currently only exist for scientific purposes. Bindal et al. [BCC+06] and Ono [CB08] have clients determine the network latency to each other and thus require modifications to the client software. With the vast number of different BitTorrent implementations installed such solutions are unlikely to enjoy widespread adoption.

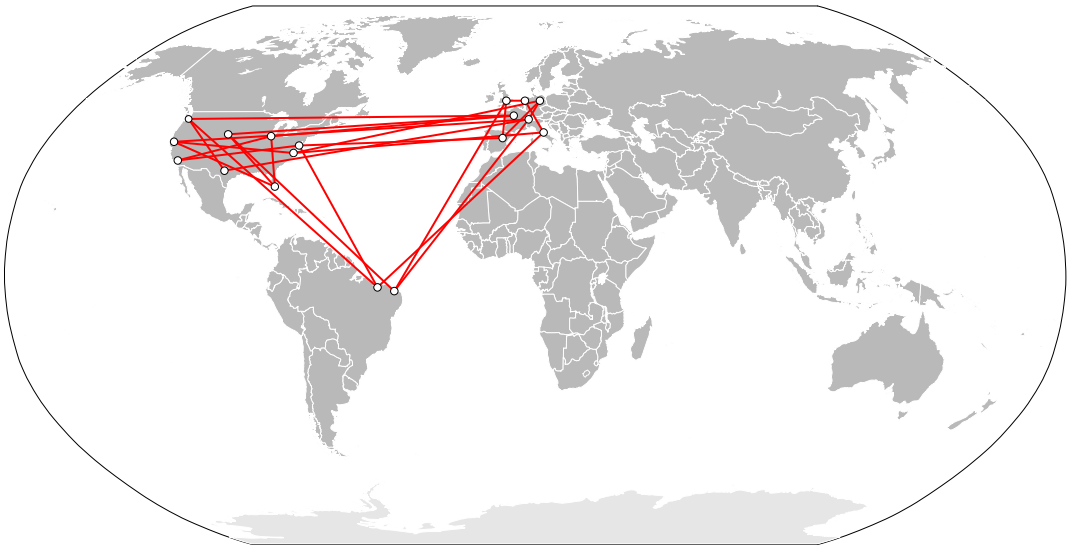

Figure 1.1: Random BitTorrent graph

To improve the flow of BitTorrent traffic for users and ISPs in an applicable way we must address a number of challenges. First, we need a way to bias traffic without modifying the client. We do this by means of biased peer selection by the tracker. If a BitTorrent client is provided with peers with better throughput rates it will itself prefer to exchange data with those peers. To achieve this the tracker needs to know about network conditions to be able to determine better peers. We find latency to be a good indication of both better throughput rates and lower traffic costs. Determining latency directly through round-trip-time measurements would result in $O(N^2)$ measurements and require client participation. Instead we use a technique known as coordinate-based latency prediction. With this technique we only need to measure the latency between the client and a small set of landmarks. From the measurements we can derive a set of coordinates in a multi-dimensional Euclidean space and predict latency by computing coordinate distance. Finally, we need a way to measure latency to landmarks without modifying the client. To this end the tracker lets BitTorrent clients connect to a small number of landmarks, which then measure the round-trip-time during the connection handshake. This allows us to localize BitTorrent traffic unilaterally from the tracker.

We evaluated our design through simulation with representative network data and Planet-Lab experiments. On average our optimizations reduce the direct traffic cost induced for a single BitTorrent swarm by 12% while decreasing the median download time for participants by 10-15%. In networks with a large number of seeds we find performance gains of up to 25%. We also find that by controlling the sample size we can maintain or even improve our gains in a rapidly growing network.

The contributions of this work are the following. We designed and implemented an infrastructure for latency-biased peer selection in BitTorrent with the dual goal of reducing long-distance network traffic and reducing download time for users. We implemented the system by modifying the popular BNBT tracker to use our peer selection algorithm and by building a latency measurement landmark for BitTorrent. To evaluate the design we

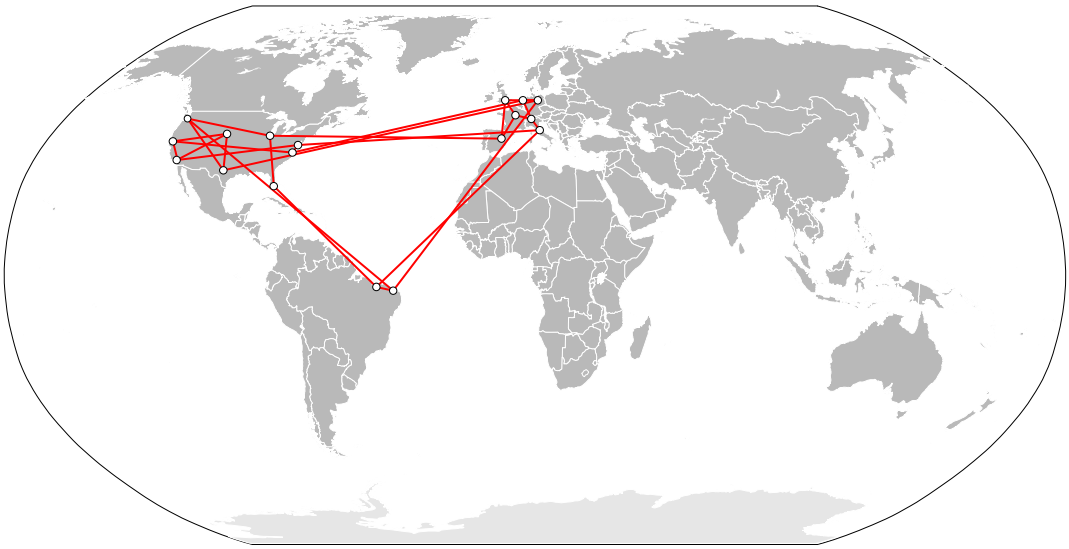

Figure 1.2: Latency-driven BitTorrent graph

implemented a simulator for Internet-wide BitTorrent networks using realistic datasets and conducted large-scale simulations. We also evaluated our approach through large-scale experiments on Planet-Lab. We have shown significant improvements in download time and network cost.

This thesis is structured as follows. In Chapter 2 we discuss the background material of our work. Chapter 3 analyzes various aspects of BitTorrent to study the viability and potential effectiveness of latency-biased peer selection. Chapter 4 outlines the design decisions and implementation of latency-biased peer selection and Chapter 5 presents the methods and results of our evaluations. Chapter 6 discusses other approaches to improving traffic locality in BitTorrent and Chapter 7 concludes the thesis.

Chapter 2

Background

This work is situated at the crossroads of several seemingly distant areas. Before we present our solution, it is important to understand a few details about them. Specifically we discuss the necessary details of BitTorrent, ISP traffic cost and Internet data throughput. We also describe coordinate-based latency prediction and latency measurement techniques which we use to implement our system.

2.1 BitTorrent

BitTorrent is a peer-to-peer protocol for distributing large files. In recent years BitTorrent has gained huge popularity as a way to cheaply distribute media and software over the Internet. Reports indicate that BitTorrent is now responsible for more than one third of Internet traffic [SM07, ZDN]. The main difference between BitTorrent and earlier peer-to-peer file exchange networks is the use of *tit-for-tat*, a cooperation strategy for the prisoner's dilemma [Axe84]. Players always start by cooperating and then do whatever the opponent does. This strategy has proven highly effective and is used by BitTorrent to prevent peers from downloading content without uploading to others, which would severely harm the capacity of the peer-to-peer network. A BitTorrent client is prepared to upload data to its neighbour, but only if the neighbour does so as well.

To publish content using BitTorrent users generate a meta-data (.torrent) file. At this point the content is separated into equal-sized pieces and hashes of the pieces are added to the .torrent file so clients can verify data they receive. The .torrent file also includes the URL of a tracker, which can be used to find peers in the network. A .torrent file provides clients with all the necessary information to participate in the network.

2.1.1 Tracker

To join a BitTorrent network, a client registers at a BitTorrent tracker. The tracker is a web application which typically tracks many BitTorrent networks simultaneously. In the request to the tracker, the client encodes at least the hash of the .torrent file, its peer identifier and its listening port number. The tracker adds the client to its internal state and gives the client a peer sample, which is a randomly chosen list of the participants of

the BitTorrent network. After receiving a peer sample the client will connect to each peer in the sample until its set of connections (*peer set*) reaches its maximum size.

The client periodically reinvokes the tracker in order to refill its peer set. The preferred re-invocation interval is given by the tracker itself.

2.1.2 Peer-to-peer communication

BitTorrent peers communicate by sending data and control messages over a single TCP connection. In addition to transferring data, peers notify each other about their piece availability, interest in each other's pieces and whether they are prepared to upload.

BitTorrent distinguishes between *seeds* and *leechers*. A seed is a peer with the complete set of files, either obtained through the BitTorrent network or by other means. If at least one seed is present then the BitTorrent network can provide the entire file. Seeds altruistically upload blocks of data to the peers achieving the highest download rate from them. Peers that have not fully downloaded the file are known as leechers. Leechers selfishly upload to the peers from which they have the highest download rate.

The act of granting download privileges to a peer is known as *unchoking*. Peers are actively informed about their current status and will start requesting blocks of data immediately after receiving an unchoke message. When choosing a piece to download the client generally picks the piece which occurs least frequently among its neighbours. This reduces the probability of having rare blocks, which would harm download time for everyone.

By periodically reassessing the upload rates and periodically uploading to random peers (*optimistic unchoke*) the leechers effectively implement a tit-for-tat based scheme which encourages them to upload data in return for high download rates. Bharambe et al. [BHP06] have found through simulation that this scheme allows BitTorrent to converge to near-optimal utilization of the global upload capacity.

2.1.3 BitTorrent network utilization

The default behaviour of BitTorrent trackers is to give peers a random sample from the entire network. The reason for this design decision given in the original BitTorrent paper is that random graphs have good robustness properties [Coh03]. It was later shown that BitTorrent does not form a random graph in practice, but is in fact robust to large numbers of failing nodes [HLB07].

Indeed, random neighbour selection makes the distribution of shortest path lengths approximately the same for all nodes. This is an important property for the flow of pieces into the network as the transfer time over the shortest path between a node A, which needs a piece, and the nearest node B, which has it, is the minimal time it will take node A to obtain the piece. Together with the rarest-first piece selection strategy, which levels discrepancies in the occurrence frequencies of pieces, random selection effectively maximizes the availability of pieces throughout the network. However, it does not take heterogeneity in channel capacity into account. Depending on network properties and current conditions, the transfer times between two arbitrary Internet hosts may vary heavily. Thus in a real network random selection may not lead to an ideal availability. It does not take advantage of higher throughput rates between specific pairs of peers.

Client	BitTorrent software installed on the user's machine
Peer	An arbitrary participant of a BitTorrent network
Node	See peer
Seed	Peer with a complete set of files
Leecher	Peer with an incomplete set of files
.torrent	Meta-data file for the content
Tracker	Web application which keeps track of peers in the network and returns peer samples
Peer sample	Set of peers returned by the tracker which the client uses to fill its peer set
Peer set	Set of peers to which the client is connected
Neighbour	Member of the peer set
Active peer set	Set of unchoked peers
Unchoke	Allow a peer to request and download blocks of data
Choke	Disallow a peer to request and download blocks of data
Optimistic unchoke	Periodically unchoke a random, choked peer
Piece	Part of the data which can be verified through hash
Block	Part of a piece transferred in a single message
Rarest-first	Piece selection strategy which prioritizes pieces which are rare among a peer's neighbours
Tit-for-tat	Cooperation strategy for the prisoner's dilemma mimicked in BitTorrent by periodically unchoking only the peers with the highest upload rate in combination with optimistic unchoke

Table 2.1: BitTorrent definitions

Finding the optimal configuration is similar to finding a multi-commodity maximum flow in an undirected graph, which is an NP-complete problem and requires full knowledge of the network. Instead we can make smaller optimizations with partial information of the network. Of course, we can only make such optimizations when some upload links are underutilized.

There are many situations in which link underutilization may occur. One is a fairly high capacity peer which is not fully saturated by its connections. The peer could increase its active peer set size to allow for more connections, but since the optimal setting depends heavily on the circumstances this is generally not done automatically, if at all. Another situation is a network with surplus capacity in the form of a large number of seeds. In this case peers will receive relatively few requests and are unlikely to fully utilize their upload bandwidth.

More importantly there is suboptimal behaviour of BitTorrent itself. Although BitTorrent has an effective method to achieve high block availability by prioritizing blocks that are rare among neighbours, that does not guarantee peers will always have blocks to offer or download. Especially in a rapidly growing network peers will often have to wait for blocks that are interesting to their neighbours.

A related problem is that downloads start slowly. Obtaining a piece without having anything to share can only be done in the short time interval between being optimistically unchoked and choked for not providing any upload or by downloading from seeds. This

situation can be partly improved if neighbours have better throughput rates. Not only does this allow the peer to obtain a piece earlier, it also increases the chance that the peer will be unchoked by seeds for longer periods.

All these situations can occur frequently, but not consistently or at a single peer. For this reason graph optimizations can only be effective when applied to a large number of peers in a single torrent. In that case many peers can achieve performance gains and will benefit the whole network.

We will refer back to the concepts explained in this section throughout this thesis. For this reason we summarize the BitTorrent terminology in table 2.1.

2.2 ISP traffic cost

The shift by home users from classical client-server communication to BitTorrent-like peer-to-peer systems has been costly for most ISPs. In most client-server applications the client sends a small request and receives a large reply. As data transport is usually paid for by the sending party this is relatively cheap for the end-user ISP. BitTorrent, on the other hand, encourages clients to upload data in return for better download rates. This has led to a massive increase in outbound data and thus traffic cost. As ISPs typically charge their end-users flat fees, this directly harms their business.

One measure ISPs have taken in an attempt to lower the cost is to throttle BitTorrent traffic by limiting the upload rate of BitTorrent packets [Prea]. Besides harming user experience, the net effect of throttling and traffic shaping on the total amount of traffic remains unclear. The tit-for-tat mechanism in BitTorrent will cause a decreased download rate when the upload rate is throttled. The amount of data uploaded by clients is therefore proportional to the amount of downloaded data, regardless of throughput. It also encourages users to switch to evasion techniques (e.g., encryption) that result in higher traffic than regular BitTorrent). Some providers are also experimenting with a shift to metered Internet [Preb], where customers are charged for the amount of traffic they generate.

To see how BitTorrent affects the traffic costs of ISPs it is important to distinguish between two different ways in which ISPs communicate with each other. Either their networks have a direct interconnection with each other, which is known as peering, or they purchase transit from an intermediate network. We now give an overview of these two mechanisms and discuss their costs with regard to BitTorrent.

2.2.1 Transit

> A Transit relationship is a business arrangement whereby an ISP provides access to the Global Internet.

[Nor00]: In this definition *Global Internet* means that the transit provider is a Tier-1 network, which means it can reach all destinations without itself purchasing transit. Tier-2 networks can only reach a relatively small number of other networks and thus rely on transit for full access to the Internet. ISPs for end-users are typically Tier-2 networks.

Transit is paid for by the upstream network per Mbps of upload capacity used. The transit provider takes short random samples to measure average throughput and determine

Required capacity (Mbps)	Transit cost per month	Peering cost per month
1	$425	$2000
2	$850	$2000
3	$1275	$2000
4	$1700	$2000
5	$2125	$2000
6	$2550	$2000
7	$2975	$2000
8	$3400	$2000
9	$3825	$2000
10	$4250	$2000

Table 2.2: Cost for traffic to another ISP

monthly fees. In some cases providers charge extra for long distance traffic, because it requires more network resources. An indication of transit costs from 2000 is given in table 2.2.

2.2.2 Peering

> Peering is the business relationship whereby ISPs reciprocally provide access to each other's customers.

[Nor00] In a peering relationship two ISPs have a direct physical interconnection, usually rented at an Internet Exchange point. When peering, ISPs agree to deliver packets to destinations within their own network, usually without additional cost. Peering is non-transitive and therefore not a replacement for transit. In terms of cost, peering has a much higher entry barrier than transit as it is not paid for by the Mbps, but by the connection. Whether peering is actually beneficial depends on the amount of data the ISPs exchange, a cost indication is given in table 2.2.

If two ISPs decide to interconnect then using the peering link is always cheaper than transit, but ISPs cannot control the destination of the traffic. Peering is typically only applicable when the destination is nearby as the networks have to be physically connected. In the case of BitTorrent the destination is one of a randomly selected sample from all peers in the network. This means they are very likely to be at a distant locations and require Tier-1 transit.

ISPs cannot easily change which peers exchange data without severely harming the user experience (e.g. blocking connections). However, peer-to-peer clients will also benefit from traffic going through peering relations. Due to the direct physical connection peering relations typically result in short paths. This means there is less potential for bandwidth bottlenecks, packet loss and latency. Peer-to-peer systems can on their turn detect these paths by measuring latency.

In consequence, we can reduce traffic costs for ISPs by letting more traffic go through lower latency paths. This will cause more traffic to go through peering relations rather than transit, which will imply significant cost savings for the ISP. It also makes establishing new peering relations in the future much more attractive. We expect that such shorter paths will also benefit the end-section user.

2.3 Internet data throughput

To study BitTorrent's performance, it is important to realise the properties of the underlying network. Bandwidth on a path between 2 hosts (A and B) always depends on the smallest of the intermediary bandwidth bottlenecks. For the Internet we can identify 4 different (categories of) bottlenecks: the upload link of A, the Internet path bandwidth, the download link of B and the TCP induced throughput. We do not consider the home network of the user as a limiting factor as mainstream 100Mbps/1Gbps networks typically have much higher capacity than the access link to the ISP. We look at each of the bottlenecks in more detail.

2.3.1 Access link

Access link limits are the most noticeable bandwidth bottleneck as they are the highest throughput the user will achieve with any application at any time. A peculiar property of these limits is that the download limit is usually much higher than the upload limit. This was done because classical Internet applications such as the Web are characterized by asynchronous traffic volumes with small requests going to the server and large files going to the client. End-user ISPs therefore chose reserve a larger share of the physical link for download. Peer-to-peer applications like BitTorrent do not have this characteristic as every peer is as much a server as it is a client. Due to the tit-for-tat mechanism a low upload limit will in fact limit the download rate. Whether a client is able to saturate its download link depends primarily on the amount of altruism (read: seeds) in the network, but we can expect the download link is not the main bottleneck.

In addition to the physical limits of the access links, users may impose limits in the BitTorrent client itself. This is done to prevent BitTorrent from fully saturating the capacity of the physical link, which would disturb other applications. The connection throughput is further limited by BitTorrent having multiple active connections, which means the per-connection upload limit is only a slice of the full upload link. Since many users have a fairly low capacity upload link to begin with, the access link is the number 1 bottleneck determining data throughput in BitTorrent. It is also the bottleneck that can not be widened. Thus any performance improvements will have to rely on underutilization of upload links.

2.3.2 Internet throughput

User-perceived throughput for a connection over the Internet is the result of a complex interplay of different network components. We will not go into the details of Internet routing, but we will point out a few important properties.

Most important for our purpose is the observation that there exists a weak, inverse correlation between latency and bandwidth [OCP+06, LSB+05]. The reason for this is that high latency is indicative of a heavily loaded router or a long network path. In both cases there is more potential for congestions which limit the bandwidth.

A second consideration is that longer network paths also imply higher loss rates. Since the Internet does not guarantee packet delivery any router can arbitrarily drop packets.

The more routers a packet visits on its path, the more potential for packet loss there is. This will directly affect the throughput of TCP.

2.3.3 TCP throughput

For a single connection TCP itself can be a bottleneck. Since every message in TCP needs to be acknowledged by a return message the maximum throughput of TCP is $\frac{W_{max}}{RTT}$ with W_{max} being the maximum number of sent, but unacknowledged bytes and RTT the round-trip-time between the two nodes. In standard TCP the window size is communicated in a 16-bit field and can therefore never be larger than 65536 bytes. When a packet is lost TCP will need to wait for the confirmation time-out and this will reduce throughput even further.

In modern operating systems the window size is no longer restricted to 2^{16} bytes. As early as 1992 it was clear that TCP throughput would become a bottleneck and an RFC [JBB92] describing window scaling was introduced. TCP window scaling uses previously unused bits in the TCP header to shift the window size left by at most 14 positions. This gives a maximum window size of 1GB, quite sufficient to transfer data over a TCP connection to the sun at 10Mbps. Although the results are not quite so dramatic when assuming even a small amount of packet loss, it would still be sufficient to make the network capacity the bottleneck in any case. However despite its early introduction window scaling has only recently been adopted by major operating systems. Linux supports window scaling since version 2.6.8 and Windows Vista is the first version of Windows to have it enabled by default. Although Vista is expected to grow, Windows XP is still the most used operating system by far, especially among home users [W3C]. In addition many problems exist with home routers which leads users to disable it [LWM].

For the purpose of BitTorrent we may improve performance by using connections with lower latency as this will lead to higher bandwidth and TCP throughput. However, this can only be done when upload links are underutilized.

2.4 Latency prediction

The simplest, most accurate way to find nearby peers is to actively measure round-trip-time between all of them. It is clear, however, that this approach does not scale well. In a peer-to-peer environment in which we need to know all distances the measurement complexity is $O(N^2)$. To this end several schemes have been developed which can predict latency between arbitrary hosts with reasonable accuracy using only a limited number of measurements.

2.4.1 GNP

GNP [NZ02] (Global Network Positioning) is a latency prediction technique which models latency in a Euclidean space. A set of coordinates is computed for every node and latency is estimated by computing the distance between coordinates. The advantage of coordi-

nate based latency prediction is the small number of latency measurements compared to relatively good accuracy ($O(1)$ in case of GNP).

Similar to physical positioning systems such as GPS, nodes measure their distance to a small number of landmarks. In the initial phase of GNP each landmark measures its round-trip-time to every other landmark. When all measurements are done the coordinates of the landmarks need to be computed. Since the measured network distances are unlikely to completely match a Euclidean space, this translates into a minimization problem over the sum of some error function ε. The error function used by GNP takes the square of the relative difference between the measured distance m_{H_1,H_2} between two landmarks and the predicted distance p_{H_1,H_2} based on the coordinates (see 2.1). The minimization method used by GNP is Simplex downhill [NM65].

$$\varepsilon(m_{H_1,H_2}, p_{H_1,H_2}) = (\frac{m_{H_1,H_2} - p_{H_1,H_2}}{m_{H_1,H_2}})^2 \tag{2.1}$$

The procedure to determine the coordinates for a regular node is similar to the initial phase. The round-trip-time between the node and each landmark is measured and coordinates are computed by minimizing the sum of the error function. Using this method GNP can predict over 90% of latencies within 50% relative error. This is generally sufficient to find nearby hosts.

2.4.2 Vivaldi

Coordinate based latency prediction is already available in one particular BitTorrent implementation. Azureus contains an implementation of Vivaldi [DCKM04], a decentralized network coordinate system. Unlike GNP, Vivaldi does not require landmarks, but relies solely on latency measurements between peers. Each peer continuously tries to minimize the error of its own position relative to the position of its neighbours based on the latency measurements. The accuracy of Vivaldi predictions is similar to that of GNP.

Although Azureus used to be the BitTorrent implementation with the largest market share, this is no longer the case [Tor]. On top of that the UDP-based protocol used to obtain the coordinates is plagued by firewall issues as we will see in chapter 3. Vivaldi is currently unsuitable for the purpose of applying latency-bias to a significant number of peers.

2.5 Latency measurement

Latency measurements are typically done by measuring the round-trip-time between the sending of a probe packet and the reception of a reply. This can be done in various ways depending on the protocols in use.

2.5.1 ICMP ping

For basic latency measurements ICMP Echo (ping) or TTL-limited packets are commonly used. ICMP (Internet Control Message Protocol) is intended for various types of notifications between Internet hosts. ICMP Echo messages are meant to verify that a router is

still online. Routers that receive an ICMP echo request immediately send back an ICMP echo reply. This also allows Internet hosts to use it to measure the round-trip-time of packets on the link to another router or host. Similarly latency can be measured by giving a packet a low maximum number of hops and when the maximum is reached the router replies immediately with failure message.

Using ICMP messages for round-trip-time measurement has several limitations when we need to rely on their results [Aue04]. Many routers and hosts will simply discard ICMP packets, making it impossible to perform latency measurements. In many networks ICMP messages, and small packets in general, also have different priority than normal packets causing inaccurate measurements.

2.5.2 BitTorrent messages

Even though packets are hidden at the application layer in many cases we can still give a reasonable estimate of round-trip-time through query-response mechanisms. To do this we must ensure that outgoing data is not buffered, but sent immediately. This does not happen if the response is sufficiently large to fill the maximum packet size of the Operating System. One such message type exists in BitTorrent. When a client requests a block it is typically answered with 16kB of data. This is much larger than the maximum segment size and thus causes the TCP buffer to send the data immediately. Unfortunately this would be a very cumbersome method for latency measurement. We can only request data when we are unchoked and this can take a considerable amount of time. Typically a peer is unchoked every 30 seconds and a client can have 50 peers to try. In a worst case scenario we would have to stay connected for 25 minutes to be able to do this type of latency measurement. In addition the appropriate moment to do latency measurements is when a client joins the network, but at this point it does not have any blocks to answer us with.

Some BitTorrent implementations also use a UDP-based protocol which in fact has an operation specifically designed to measure latency. However in the next chapter we will show that due to various reasons this can only be used to measure latency for a small fraction of the peers.

2.5.3 SYNACK/ACK measurement

An alternative technique to active probing is passive latency measurement, by means of the SYNACK/ACK technique [ASW+02]. This technique measures round-trip time by measuring the delay between packets exchanged during the three-way handshake of TCP as displayed in Figure 2.1. The reason for performing the measurements during the handshake is that the acknowledgement in the handshake may not be delayed. Consecutive acknowledgements may be delayed by up to 0.5 seconds to wait for the opportunity to piggy-back it on a data segment or acknowledge several incoming segments simultaneously.

To be able to use the SYNACK/ACK technique the client only needs to open a TCP connection to the measuring point. This is a reliable and fairly accurate solution, but the disadvantage is that the measurement server needs to be able to use raw sockets. In most situations this will mean running as root. Despite this disadvantage we expect many trackers to operate their own servers and be able to use this technique.

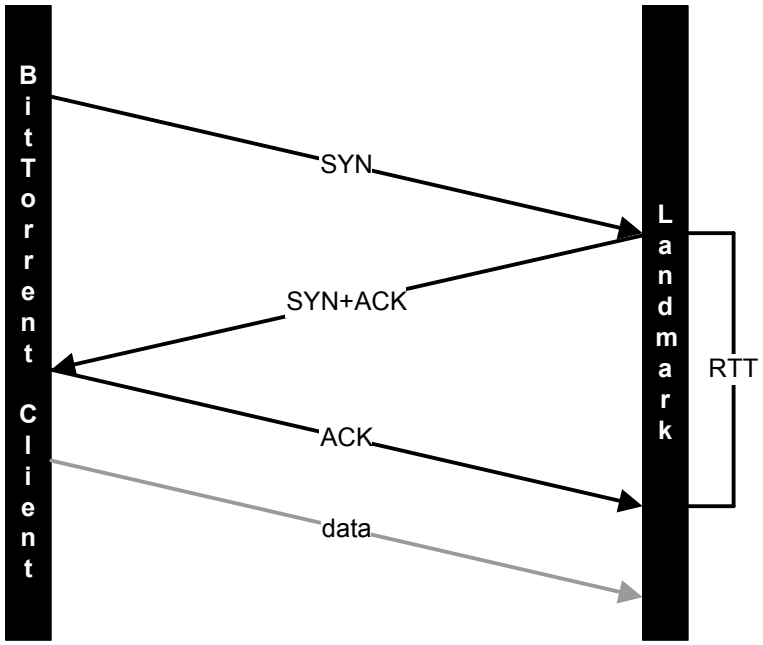

Figure 2.1: SYNACK/ACK latency measurement

2.5.4 Double segment/ACK measurement

In some environments raw sockets can only monitor TCP packets exchanged after the connection handshake. In this case we cannot make use of the immediate ACK in the TCP handshake to measure round-trip-time, but the RFC [rfc89] which specifies the delayed ACK mechanism for Internet hosts also gives an exception:

> A TCP SHOULD implement a delayed ACK, but an ACK should not be excessively delayed; in particular, the delay MUST be less than 0.5 seconds, and *in a stream of full-sized segments there SHOULD be an ACK for at least every second segment.*

In this context full-sized means that the segments are exactly as big as the Maximum Segment Size (MSS) for TCP. The MSS is typically set at 1460 bytes, thus we will receive an immediate ACK if we send a burst of at least 2920 bytes. To trigger such a burst the outgoing TCP buffer must have room for 2*MSS more bytes to prevent it from blocking. Since there is generally no practical way of checking whether this is the case, the technique should only be used at the start of a connection.

For BitTorrent this type of latency measurement can be implemented by letting the measuring point send a large number of keep-alive messages (a single byte with value 0) in addition to its handshake as shown in Figure 2.2. As this is valid BitTorrent traffic it will not cause the client to drop the connection prematurely. The disadvantage of this technique is that it has a much higher traffic overhead than SYNACK/ACK measurements.

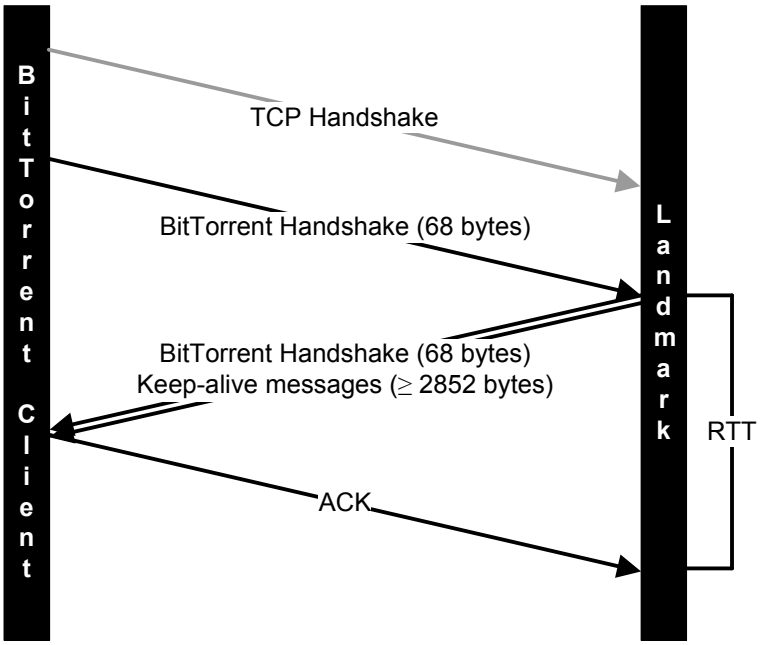

Figure 2.2: Double Segment/ACK latency measurement

Chapter 3

BitTorrent analysis

In this chapter we analyse various aspects of BitTorrent to study the viability of biased peer selection as an effective way to reduce download time and network cost. We also study the potential of the BitTorrent DHT protocol for latency measurement and prediction.

3.1 Utilization

BitTorrent has been shown to converge to near optimal upload utilization in simulations by Bharambe et al. [BHP06]. In this case optimizations will have little effect on the download time and thus be of no benefit to users. The question is whether it is also the case in a real network environment. To learn more about upload utilization in a real network we studied the dataset gathered by Izal [IUKB+04].

The Izal dataset is a log of all requests to a BitTorrent tracker for a single torrent over 5 months. Every time a client invokes the tracker it reports the number of bytes it has uploaded. Using the timestamps this allows us to compute a mean upload rate over a time segment. For all hosts that stayed for more than 4 hours we computed the upload rate over time as a percentage of the maximum upload rate achieved by that host. In Figure 3.1 we show the mean of the percentages of all hosts in their first 4 hours.

The figure shows that, on average, clients take up to 40 minutes to reach their maximum throughput rate. The reason why downloads in BitTorrent start slowly is quite clear: it takes some time before the client has something to share. Until the client gathers a full piece it can only get data by being optimistically unchoked as it cannot be unchoked providing a high upload rate. Nonetheless this process apparently takes a very long time. If peers can obtain pieces faster by having neighbours with better throughput rates this period can be shortened. If the neighbours are also seeds this will lead the peer to be choked less, further improving performance.

The absolute value of the utilization is far from optimal, indicating further room for performance gains. However, it does not follow from the data to what extent this is inefficiency and to what extent it is bandwidth being allocated to other, simultaneous BitTorrent downloads.

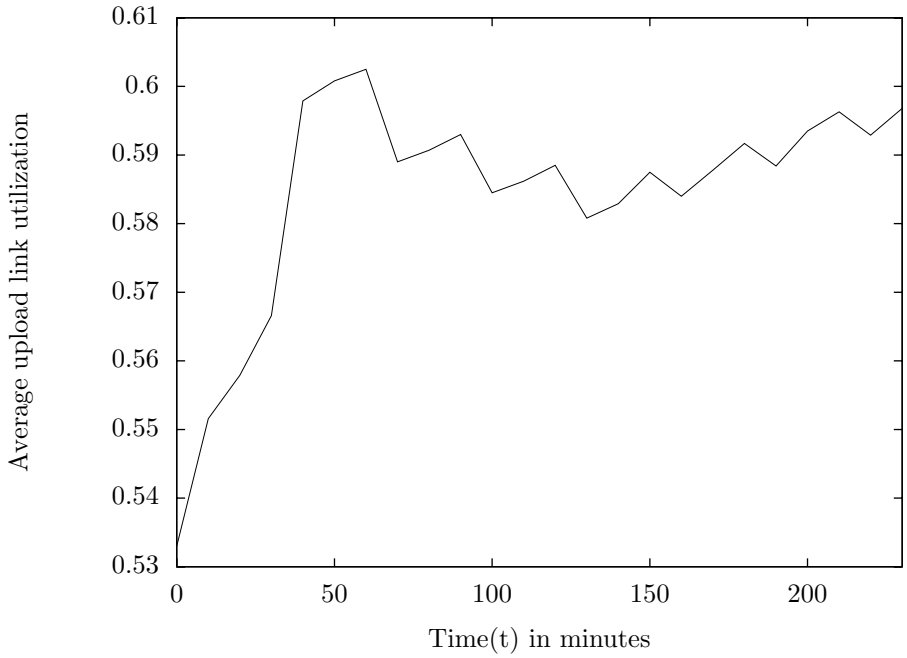

Figure 3.1: Upload utilization over time

3.2 Network size

Peer selection bias can only be effectively applied to networks with a sufficiently large number of peers. If the network is smaller than the peer set size (usually 50) clients will simply have all peers in their peer set. To investigate whether biased peer selection can be useful in practice we need to know what percentage of BitTorrent clients are actually in large networks. To this end we screen-scraped the number of seeds and leechers from the 150,000 most recent torrents on thepiratebay.org, the most popular BitTorrent tracker. In the sample we found that 50% of nodes are in torrents of size 82 or larger. The complete distribution is shown in Figure 3.2. This means we can do relevant selection for approximately half of all BitTorrent clients. In addition, we can expect the networks they are in to generate the majority of traffic as many smaller networks may be idle.

3.3 DHT

Many BitTorrent implementations, among which all the most popular, support some form of Distributed Hash Table (DHT). The DHT is essentially a peer-to-peer network on top of BitTorrent and its primary function is to provide peer samples when no tracker is available. Two different DHT protocols exist: Regular DHT [Loe] and Azureus DHT [Azu]. In both protocols clients communicate through UDP messages and have a ping operation which can be directly used to measure round-trip-time. The Azureus DHT is also used to measure and disseminate Vivaldi coordinates.

To test whether the DHT is a viable option for latency measurement and prediction

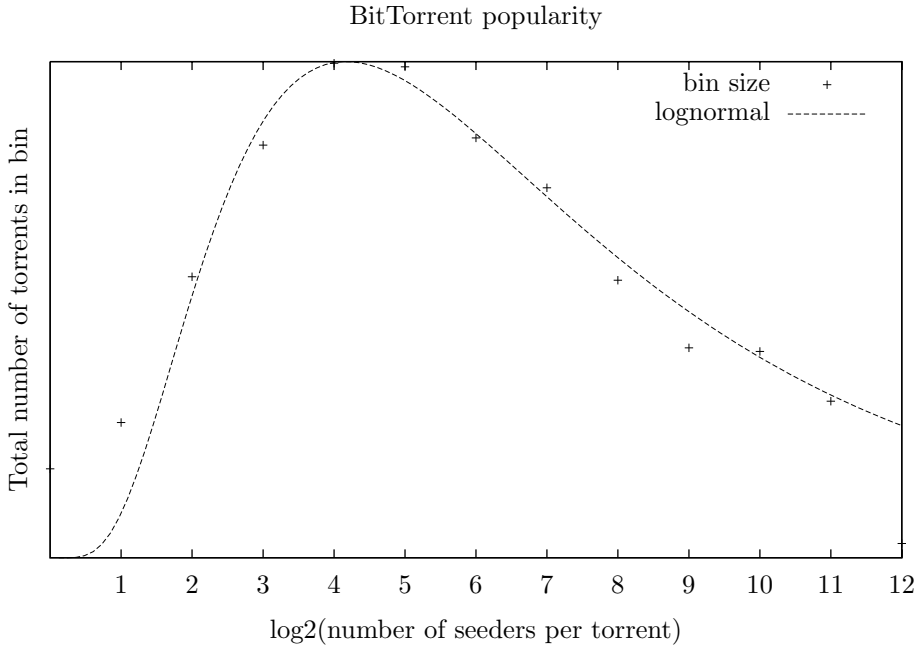

Figure 3.2: BitTorrent swarm size distribution (number of peers in networks with $2^x - 1 < size \leq 2^x$)

we built a primitive BitTorrent client supporting the regular DHT ping operation, the Azureus DHT ping operation, and the handshake of the regular BitTorrent protocol.

DHT messages should typically be sent to the same port the client is listening on for TCP connections. A client can learn this port in 3 ways: from the original peer sample received from the tracker, from a PORT message received through the standard BitTorrent protocol, or from the source field of a UDP message it receives. The latter option is only useful if the outgoing message is bound to the port on which the client waits for requests.

First we attempted to contact nodes directly by taking samples from a popular public BitTorrent tracker. Our client would first attempt to send the appropriate ping packet (Azureus or normal) to the port returned by the tracker and retried on the appropriate port if a PORT message or a DHT request arrived. In all samples at most 30% of the nodes responded to any form of ping. One potential reason could be that many clients are behind NAT routers. A NAT router rewrites IP addresses and port numbers to allow multiple hosts to access the Internet with a single public IP address. When a UDP message is sent from inside a network through a NAT router the router will temporarily keep the outgoing port open. Replies arriving at the advertised port will then be forwarded to the host inside the network.

The second method we tried is to add the testing client to various popular torrents by registering it at the tracker and waiting for connections. In this case the peers being measured initiate the connection and the NAT will accept ping messages. The measurement client would learn the port to use from DHT requests it receives or a PORT message. This method was even less successful. We found that only a fraction of clients use the PORT

message and most of those that did send DHT requests did not respond to ping on the source port. More importantly, more than half the clients did not appear to use any form of DHT. At this time we find the DHT ping to be unsuitable for obtaining a sufficiently large view of the latencies in a whole torrent. These findings also exclude Vivaldi coordinates as a viable option for latency-bias if we wish to apply to a considerable part of the torrent network as these can only be obtained through the Azureus DHT.

Chapter 4

Design and Implementation

There are a number of important design decisions that distinguish this work from other solutions. In this chapter we explain these decisions and discuss implementation details.

4.1 Overview

To reduce long-distance traffic in BitTorrent we aim to bias it towards shorter network paths. As we discussed in Chapter 2 the effectiveness of this optimization in terms of download performance depends on its application to a large number of peers in the network. Required modifications to the client software therefore have no benefit until they are widely adopted. With no single dominant BitTorrent implementation [Tor] such modifications are unlikely to be deployed at a sufficiently large scale. Instead we control the peers to which clients connect using standard BitTorrent protocols.

The tracker is the primary mechanism to obtain peers in BitTorrent. Other distributed peer sampling mechanisms also exist, but these are typically only used to complement the tracker. To bias peer samples returned by the tracker towards nearby peers the tracker needs to be made aware of network conditions. One option is to let the tracker query a network oracle [WSH99, XKLS07, MIP+06]. However, such infrastructure has only been deployed for scientific purposes and it remains to be seen to what extent it will have a commercial or public role. If we want to deploy our system today we need to perform our own measurements and estimations.

As we discussed in chapter 3 the existing protocols for latency measurement and prediction in BitTorrent are not sufficiently widespread to estimate latencies between a majority of peers. Instead we let trackers use a GNP [NZ02] infrastructure to assign n-dimensional coordinates to every peer. This means the tracker operator needs to find or setup a small set of geographically dispersed landmarks. Landmark measurements need to be collected by the tracker in order to compute a coordinates in a Euclidean space. In the next sections we outline the design and implementation details of the landmark and tracker component.

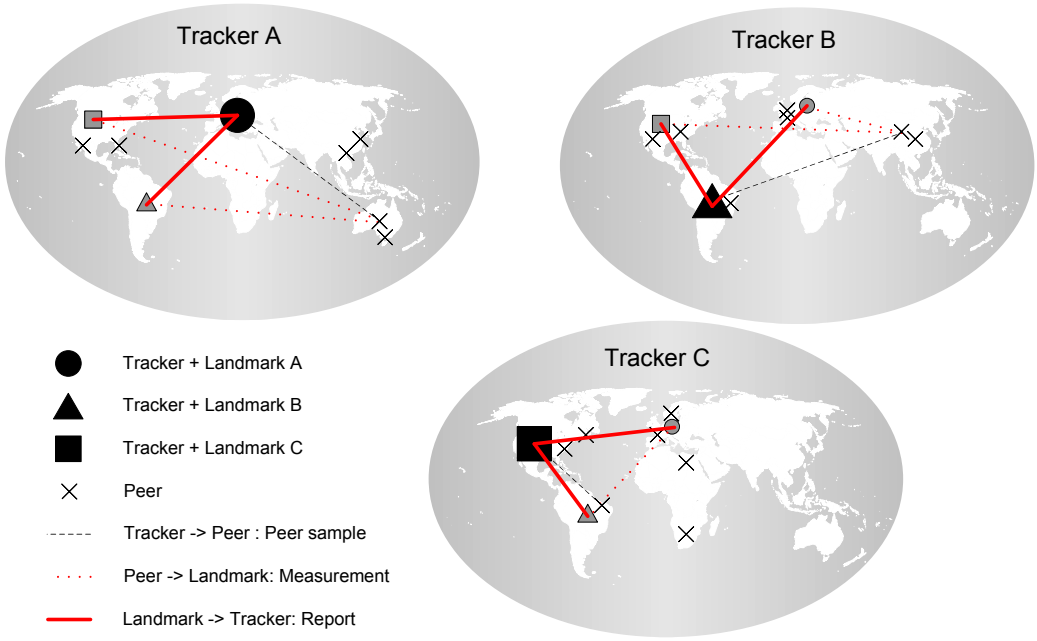

Figure 4.1: Overall architecture for cooperative 2D latency prediction

4.2 Landmark

GNP requires the latency between each peer and small set of landmarks to be measured. In our design this means the landmark must measure round-trip-time to the peer and report it back to the tracker. To be able to perform latency measurement to all peers we use the SYNACK/ACK technique [JD02]. The SYNACK/ACK technique requires the peers to make a TCP connection to the landmark. In order to achieve this we have the tracker add the addresses of the landmarks to initial peer sample returned to the client. Upon receiving the sample the client will immediately connect to the landmark, expecting it be another peer.

Once a peer establishes a connection the landmark has already measured the latency in the TCP handshake. In principle the landmark can immediately close the connection, but to have a graceful disconnect and to identify the peer and the tracker it was sent by the landmark also performs the initial BitTorrent handshake in which it receives the peer identifier and the torrent hash. Afterwards the landmark closes the connection.

To obtain accurate measurements the landmark must respond immediately when a client tries to open a connection. The landmark uses a pre-fork architecture and should have sufficient number of worker processes to handle all clients simultaneously.

Trackers can connect to the landmark and will receive reports of measurements. Each report includes the torrent hash, peer identifier, IP address, TCP port and the measured delay. In the current implementation reports are sent to all connected trackers, but this can easily be turned into a publish-subscribe system in which trackers subscribe only to specific torrent hashes. To prevent unauthorized use the landmark must implement some form of authentication. Currently this is done through a per-landmark password.

We expect groups of (6-8) independent trackers will provide each other with landmarks for mutual benefit. The additional upload/download overhead is only 68 bytes per client and clients are only measured once. Landmarks are stateless and have a marginal footprint. An overview of such a cooperative latency prediction architecture is shown in Figure 4.1.

4.2.1 Tracker

The tracker uses coordinate based latency prediction to bias its peer samples towards nearby nodes. The advantage of coordinate based latency prediction is that it only requires a small number of latency measurements and it fits our requirement of not have to rely on any client modifications or additional infrastructure. In principle any coordinate based system which uses a small number of static landmarks can be used, in this thesis we use the original GNP [NZ02], because it is well-known and widely documented [SPPvS08].

Any tracker implementation can be extended to implement our approach by making two additions. The first addition is our latency-prediction module, which can be added to the implementation as a separate component. The second addition is the peer selection algorithm which is implementation-dependent. We have implemented this system on the BNBT tracker.

4.2.2 Latency-prediction module

The latency-prediction module is the component responsible for gathering latency measurements from landmarks, computing GNP coordinates and making them available to the tracker. It is implemented as a separate thread and, outside (de)initialization, it provides 2 simple operations to the application: *get_coordinates(peer_id)* and *compute_distance(coordinates,coordinates)*. The module tries to maintain a connection to its landmarks at all time. When a client requests a peer sample the tracker will first lookup its coordinates using get_coordinates. If no coordinates are found the tracker returns as a peer sample the addresses of the landmarks and some random peers to allow the client to start downloading.

Under normal circumstances the client should connect to the landmarks immediately after receiving them in the peer sample. The landmarks then report the measured latency back to the prediction module and once it has obtained d+1 measurements it can compute d-dimensional coordinates for the peer. If it receives more than d+1 measurements (i.e. there are redundant landmarks) it will recompute the coordinates in hopes of better accuracy.

If for some reason the prediction module does not succeed at collecting a sufficient number of measurements before the peer asks for another peer sample the procedure can be repeated. However, most likely not enough landmarks are available, so the tracker should return a normal random sample.

The number of random peers a client receives in the initial sample should be small to prevent the client from saturating its peer set before receiving nearby peers. The consequence of this is that the peer may have a poor download rate until the second invocation. The invocation interval returned by the tracker in the initial sample should

therefore be very low and may be higher afterwards. In this case the gain of latency-driven BitTorrent will easily make up for the lost time.

4.2.3 Peer selection algorithm

When a peer contacts the tracker and its coordinates are known, the tracker uses the peer selection algorithm. To prevent parts of the network from being disconnected the peer samples are not fully biased. The selection algorithm takes a bias parameter from 0.0 to 1.0 which means the peer samples will consist of $n = bias * sample\_size$ biased peers and $sample\_size - n$ random peers.

A naive algorithm selects the n closest nodes. However, this algorithm causes large variance in the average download time. The source of the variance we found in simulations was the small number of random peers. We believe the reason for this is that the nodes form well-connected islands. A peer would usually prefer talking to its closest neighbours over talking to other neighbours. It would therefore take relatively long for a piece of data to be forwarded to another island as the peer will first upload to its nearest neighbours. This hurts the overall distribution of pieces unless the islands are at more or less equal distance from the seed. Whether this is the case depends on the small number of random neighbours in the peer samples. An additional problem with the naive approach is that peers will refuse to accept new connections if their peer set is full. If the tracker only returns the nearest peers the client will not find new peers to connect to, except those that were randomly chosen.

Our improved algorithm selects n peers randomly from the 25% closest peers and selects the remaining peers randomly from all peers. This algorithm gives better and more stable results.

In situations where some peers are far away from everyone or overloaded the latency-bias could result in these peers being explicitly connected. Since this would be counter productive the algorithm applies a threshold of 500ms for peers that are considered to be nearby.

4.2.4 Adaptive sample size

One issue concerning our approach is clustering in a growing network. BitTorrent clients will quickly try to find as many peers as possible and thus saturate their peer set. When new, nearby peers join the network, the existing peers will not disconnect from their more remote neighbours. This can severely reduce the effect of latency-biased peer selection.

To counter this effect we can give clients smaller peer samples while the network is growing. We can also adapt the size of the sample to the size and growth of the network. We will evaluate these mechanisms in Chapter 5.

The implementation of the landmarks and the modified BNBT tracker are available at `http://www.few.vu.nl/~marco/`.

Chapter 5

Evaluation

Evaluating the performance of any BitTorrent optimization is hard, because we cannot easily repeat a large-scale scenario that is representative for real BitTorrent networks. Simulation may often show poor accuracy with regard to reality, while actual deployment does not allow repeatable experiments. We evaluate our scheme using both (carefully designed) simulations and deployment on PlanetLab.

5.1 Simulation

BitTorrent is typically used to distribute large media files, which are mostly popular among home users. As a result a typical BitTorrent scenario involves mostly asynchronous (ADSL) Internet links and a heterogeneous network environment. Since our optimizations are targeted at BitTorrent networks with a large number of users, we have no practical or reliable way to have a representative set of users repeat an experiment multiple times for comparison. Instead we built an event-driven BitTorrent simulator with a fine-grained network model using realistic data.

5.1.1 Peer set

To create a representative set of peers to use in our simulations we use data provided by iPlane [MIP+06]. iPlane is a service providing loss and latency predictions for arbitrary Internet paths. Predictions are made by modelling the Internet as a Point of Presence (PoP) graph and measuring the properties of individual links using common tools such as traceroute from PlanetLab nodes. The predictions made by iPlane are made available through a Web Service. In addition to loss and latency measurements iPlane periodically participates in popular BitTorrent swarms to obtain additional measurements of edge nodes. The measurements include access link bandwidths, obtained by monitoring TCP packet traces.

In our experiments we take a random sample from the set of peers for which iPlane measured the access link bandwidth. We obtain latencies and loss rates for the prefixes in our sample from the iPlane web service. The simulator treats these values as the actual latency and loss rates.

5.1.2 BitTorrent model

The simulator implements the basic algorithms described in the BitTorrent specification [bitb, the], including: choking, optimistic unchoking, strict piece priority, request pipelining, rarest first piece selection. For simplicity it does not model end-game mode, which is otherwise unrelated to latency effects.

5.1.3 Networking

When simulating the underlying network we need to capture any effect that can be of influence on the performance when applying latency bias, or take a pessimistic approach towards our optimizations when the effect is very hard to model. The simulator models propagation delay, TCP throughput and upload capacity sharing. The design of these simulations are outlined in the following sections.

Delay

Most messages in BitTorrent do not affect performance since they carry asynchronous notifications. There is, however, one exception. When a client A is unchoked by its neighbour B it immediately starts sending requests. When client B receives the request it immediately starts uploading a piece message. In between the unchoke operation and the first data arriving there is a delay of at least $2 * latency(A, B) + latency(B, A)$ and this can directly influence performance if it occurs frequently. Further requests are queued at B and only become relevant if the upload rate of B is sufficiently high to honour all requests within a single round-trip-time. This is rarely the case.

To capture the effects of request delay the simulator delays the arrival of a packet going from A to B by at least $latency(A, B)$. The arrival may be delayed further if the message is preceded by piece messages that are still being uploaded. In that case the message is delivered after completion of the piece messages.

Throughput

To simulate data transfer the simulator computes the current throughput for every active connection, and appropriately schedules the arrival of messages. To compute the data throughput between two arbitrary nodes A and B we again distinguish between 4 potential bottlenecks: the upload limit of A, the download limit of B, wide-area bottlenecks and TCP throughput.

The upload capacity of a node A determines the maximum cumulative throughput of all traffic coming from A. In the simulation every active connection gets an equal slice of the upload capacity, unless it is limited by another bottleneck in which case the remainder is redistributed among the other connections. The download capacity of B can also be a bottleneck, but due to the tit-for-tat mechanism in BitTorrent the cumulative download rate achieved by a node is never much higher than its upload capacity. Since most users have ADSL style connections and download limits can be an order of magnitude higher than upload limits we do not model download capacity. We use the access link capacity from the iPlane dataset as the upload capacity.

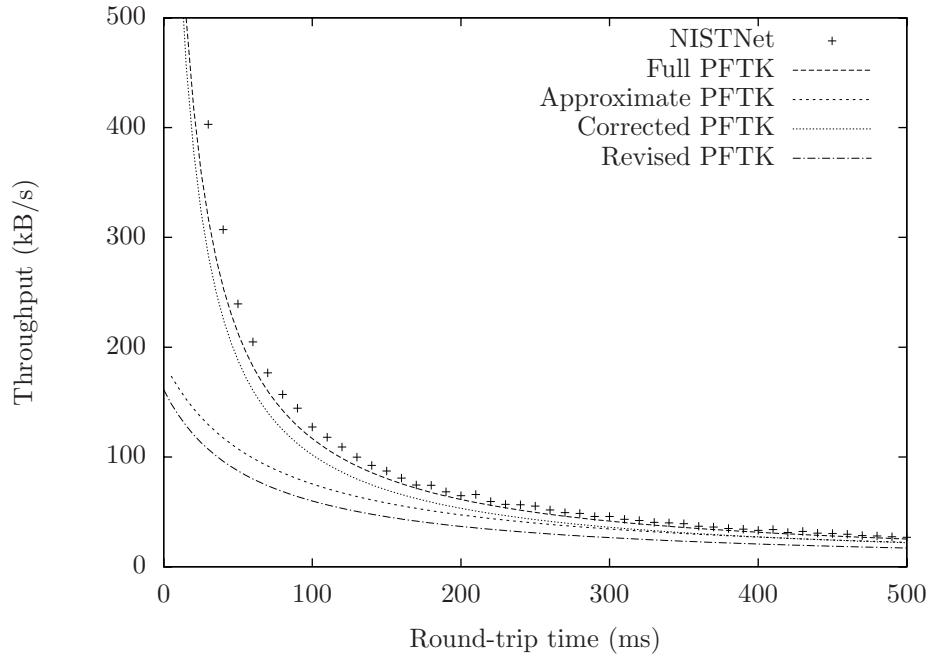

Figure 5.1: TCP throughput for 1% loss

On the path through the wide-area network (Internet), routers may experience congestions. These congestions can be a smaller bottleneck than the access link or the logical TCP limit and thus be the limiting factor on throughput. Although latency-biased peer selection can reduce the number of congested paths by selecting shorter paths, accurate data on this matter is hard to obtain. Instead we take a pessimistic approach with regard to the results of our evaluation and assume the wide-area network is never the bottleneck. We show that we can achieve significant improvements without factoring in wide-area congestions.

Finally the throughput of TCP can be a bottleneck for a single connection. Several models for TCP throughput exist. Padhye et al. [PFTK98] modelled TCP using two different formulas, the full formula (often referred to as the PFTK model after the initials of the authors) and a 'light-weight' approximation. The full model was later corrected [CBAT] and Dunaytsev et al. [DKH06] argued that the PFTK model is too optimistic in its predictions and developed a revised version. To simulate TCP throughput rates we need to select an appropriate model. To find which model is most accurate we measure average TCP throughput from one Linux machine to another using NISTNet [Gro00]. NISTNet is a network emulator which routes packets with pre-configured loss rates, latencies and bandwidth limits. We measure the average throughput rate over a longer period for a loss rate of 1%. The results of these experiments are displayed in Figure 5.1. We find the original PFTK model gives the most accurate predictions overall in our experiments.

In the experiments we used the original PFTK model, with a maximum window size of 2^{16} bytes.

5.1.4 Experiment

No single BitTorrent scenario is fully appropriate to evaluate its performance, as every change to the network affects its capacity and behaviour. We try to create a typical scenario using realistic datasets and note that results may slightly vary when the network develops differently.

One constraint comes from the fact that we use a random sample of only 200 peers drawn from the iPlane dataset. The simulator requires detailed network data and obtaining this from the iPlane service is fairly expensive. Either a full graph has to be obtained on beforehand or the data must be obtained during the simulation. Both would be incredibly time- and resource-consuming for large networks. In the former case we need to do $O(N^2)$ invocations. In the latter case it will be very difficult to predict which paths will actually be used, resulting in a very large number of small requests. For that reason we do not extend our network beyond 200 nodes, but rather let the same nodes rejoin after they have left.

To have a realistic join rate we take a segment out of the Izal tracker log [IUKB+04]. For the first 1000 peers that joined during the first hours of April 3rd 2003 we join an inactive peer. To ensure the same amount of traffic is exchanged in every scenario we let the remaining peers finish their download after the last peer has joined. The peers download 256MB of data originating from a single seed, which remains available during the whole experiment and has above average upload capacity. Since our completion times will not match the tracker log we do not use the tracker departure times. Instead we let peers stay with mean departure time of 120 seconds after completing their download. The departure times are drawn from a Poisson distribution. The parameters for our experiments were chosen to keep the network size in a stable range below 200.

Our scenario goes through several phases. In the first phase our network gradually grows from a single seed. In the second phase the network size remains stable until the last peer has joined and finally all peers leave. We distinguish between these phases by looking separately at the first 200 nodes that finished.

We ran the simulation 5 times with a normal tracker, and 5 times with a latency-driven tracker.

5.1.5 Download time

The primary metric for evaluation is the download time. The potential adoption of our approach depends on whether we can provide download time improvements for a large number of peers.

Stable network

In the simulator we find that for a stable network we improve median download time by 12% and average download time by 10% for this experiment. More importantly the majority of peers experience download time improvements. We give the download time distribution in Figure 5.2.

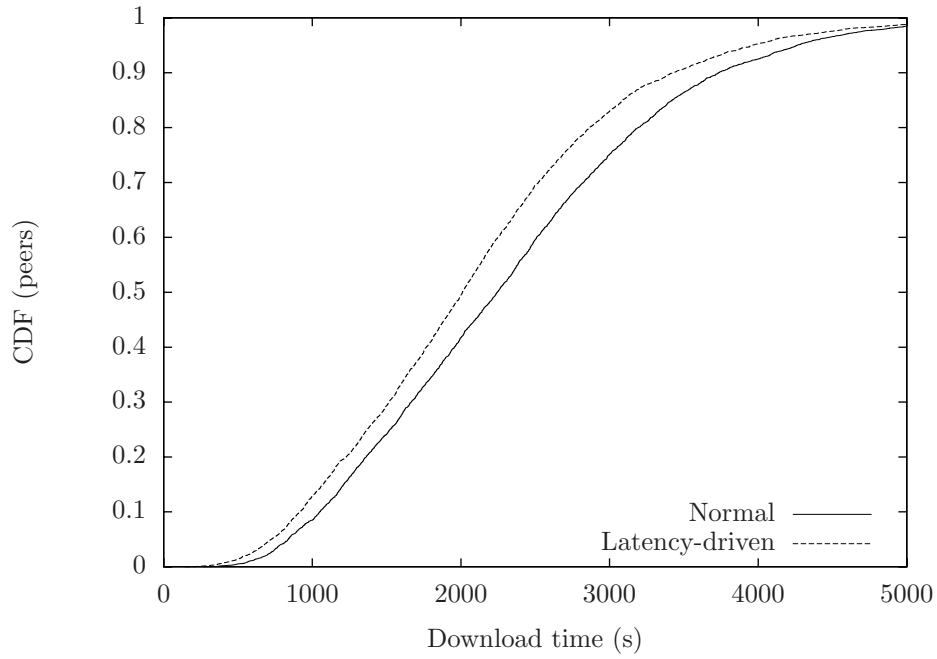

Figure 5.2: Peers finished with download time $< x$

Growing network

In the initial phase of the network the effects of latency-biased peer selection are limited. However, knowing that the network is growing and more peers will soon be available we can apply an additional optimization. If we keep the sample sizes small the peers will not immediately saturate their peer set with poorly chosen peers. We study the download times of the first 200 peers that finish in our experiments using different sample sizes. We also evaluate an adaptive approach, which has sample size $2*\sqrt{network\_size}$ (an arbitrary, slow-growing function). The results are shown in Figure 5.3.

Using the adaptive approach we improve median and average download time by 16% in the first phase. Thus latency-bias can work unexpectedly well for growing networks, even when the network is very small. We believe the reason for this is that there is a much higher amount of underutilization due to the low piece availability. This gives more benefit to latency-biased peer selection as the TCP throughput is the bottleneck more often.

We also find small improvements by simply taking a smaller sample size, even when not applying any bias. This suggests that the cliques formed in a growing BitTorrent network harm the download time and BitTorrent could handle flash crowds more efficiently by searching for neighbours less aggressively.

We do not find much benefit from using a smaller or adaptive sample size in the stable phase.

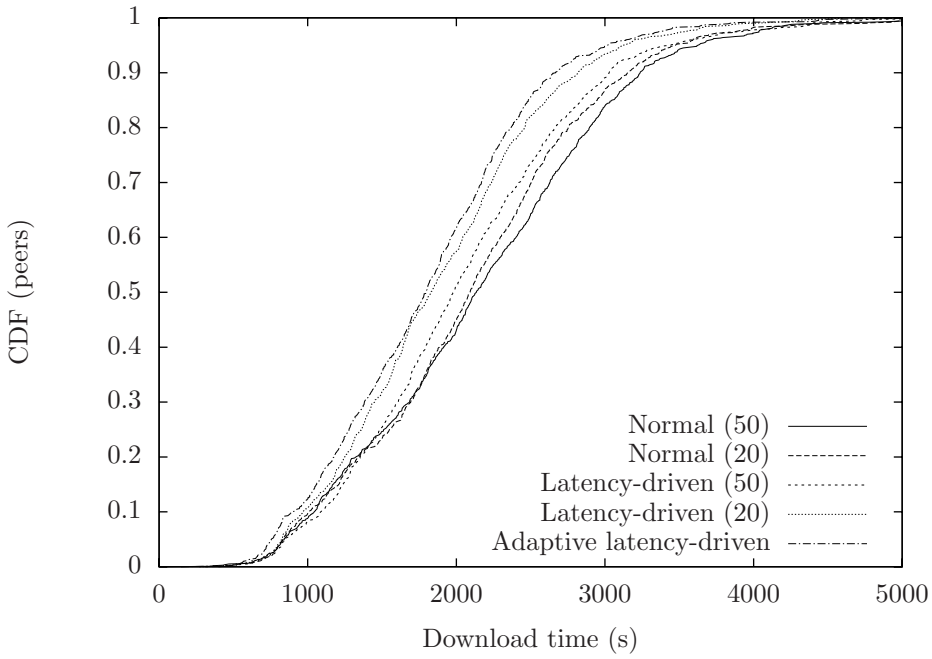

Figure 5.3: Peers finished with download time $< x$ among first 200 for different sample sizes

Seedful network

One particular element in our scenario is fairly unrealistic compared to most large BitTorrent networks. We only use a single seed and peers that have completed their download stay for a relatively short period. In many BitTorrent networks the number of seeds is much higher. In the network size sample we took from thepiratebay.org (see Chapter 3) we found that seeds in fact outnumber leechers overall.

We run our experiments again, but now starting with 100 seeds. Although this seems a peculiar start, even networks with thousands of seeds for 0-10 leechers are not a rarity. In these experiments we find median gains of up to 25%. Again we can attribute this to the low utilization of the available upload capacity which allows latency-driven BitTorrent to thrive. In such an environment BitTorrent becomes more like a Content Delivery Network and we can easily benefit from finding the closest replica (seed).

For the rest of this section we will use the stable experiment, because it most constrained and challenging to our system.

5.1.6 Utilization

For further comparison of normal and latency-driven BitTorrent the utilization of the global upload capacity is an important value. It indicates how efficiently the available capacity is being used. To compare the latency-driven and normal trackers we need to compute the utilization over the entire lifetime of the BitTorrent network.

Every time the capacity of a network changes (i.e. a peer leaves or joins) we compute

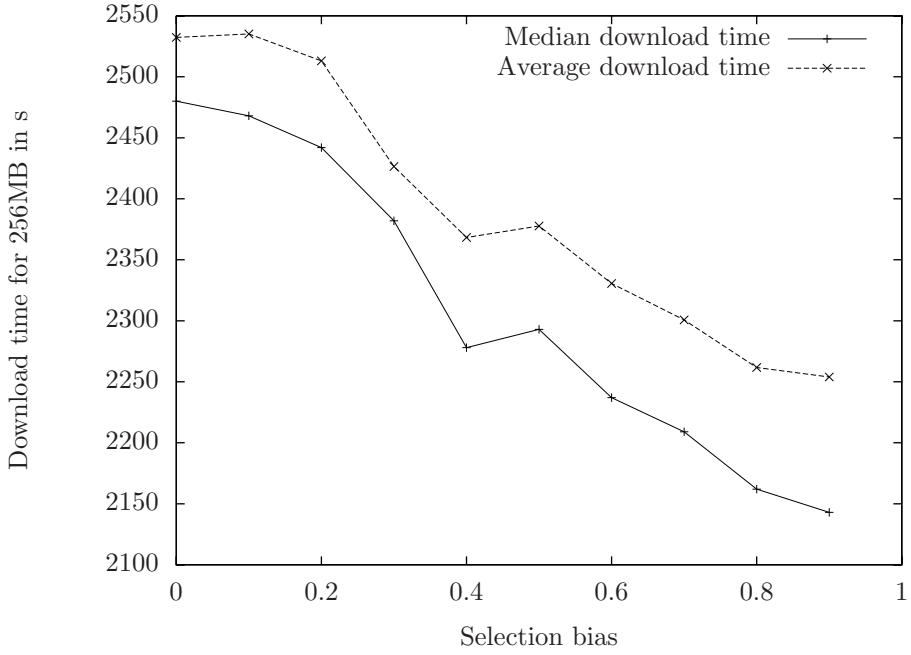

Figure 5.4: Median and average download time for bias x

the utilization since the last change. We then compute the running average to obtain the global utilization up until now. Using this method we found an average increase of 7% in the global utilization at the end of the (full) experiment. Given that most nodes are near 90% utilization at all time this can be considered as a strong increase.

We should note that the actual values we find for the global utilization are in the 20%-25% range. This differs significantly from the values found in a similar simulation by Bharambe et al [BHP06], which were in the 90% range. The reason for this is that we did not leave out high capacity peers from our simulation, and their upload is restricted by the maximum TCP throughput. These peers add high capacity, but relatively low throughput to the global utilization. The average (equally weighted) utilization values are similar to those found by Bharambe.

5.1.7 Bias

Latency-based selection bias has clear advantages, but applying too much of it may have negative consequences for performance. If peers form clusters, pieces may not go through the network quickly enough. To study the bias we perform our experiment with a latency-driven tracker with bias values 0.0-0.9, with 0.0 being regular BitTorrent and 0.9 meaning 90% of the samples consist of nearby peers. We plot the median and average download time over all runs in Figure 5.4. We find that although clustering may have negative effects, the effects of latency-driven BitTorrent are strong enough to counter this effect. In the experiments we use bias 0.9 unless otherwise stated.

AS number	Name
209	Qwest
701	Verizon Business
1239	Sprint
2914	NTT Communications
3356	Level 3 Communications
3549	Global Crossing
7018	AT&T

Table 5.1: Tier-1 Autonomous systems

5.1.8 Traffic cost

We cannot directly compute traffic costs without detailed cost information from an ISP, but for all ISPs the cost depends on the same metrics. We study the amount of Tier-1 traffic and the general distance distribution.

Tier-1 traffic

The cost metric that is most relevant to ISPs is the amount of Tier-1 traffic. Tier-1 is clearly defined as a network which can reach every destination on the Internet without purchasing transit or paying settlements. However, it is often not quite clear to which networks this definition can be applied as this information is rarely available. For our purpose we define the autonomous systems in table 5.1 to be Tier-1 networks.

The autonomous systems through which packets are routed can be obtained from the iPlane dataset used in the simulations. We find in our experiments that when using a regular tracker ±76% of traffic goes through a Tier-1 network for only ±67% using a latency-driven tracker. This means that if the remaining traffic is routed through settlement free peering, the involved ISPs see their cost for large BitTorrent networks reduced by 12% on average. This is a number that is likely to increase for larger networks. Since our network has 100-200 peers that were randomly chosen from a global dataset, only few peers are within the same ISP or within physically attached ISPs. With larger, or more localized networks latency-driven BitTorrent will be able to connect more ISP-internal peers.

Distance distribution

The amount of Tier-1 traffic has direct implications for transit costs, but there are also other factors to consider. For example if there is a lot of Tier-1 traffic going through relatively short paths peering becomes more interesting. Transit providers may also distinguish between various distances to determine cost. More importantly the relation between the amount of traffic and the length of the network path indicates how efficiently the network is used. If more data is sent over shorter paths the BitTorrent network consumes less capacity.

We study how much traffic is sent over paths within a specified distance. The results of the experiment are shown in figures 5.5 and 5.6. The figures show significant reductions in

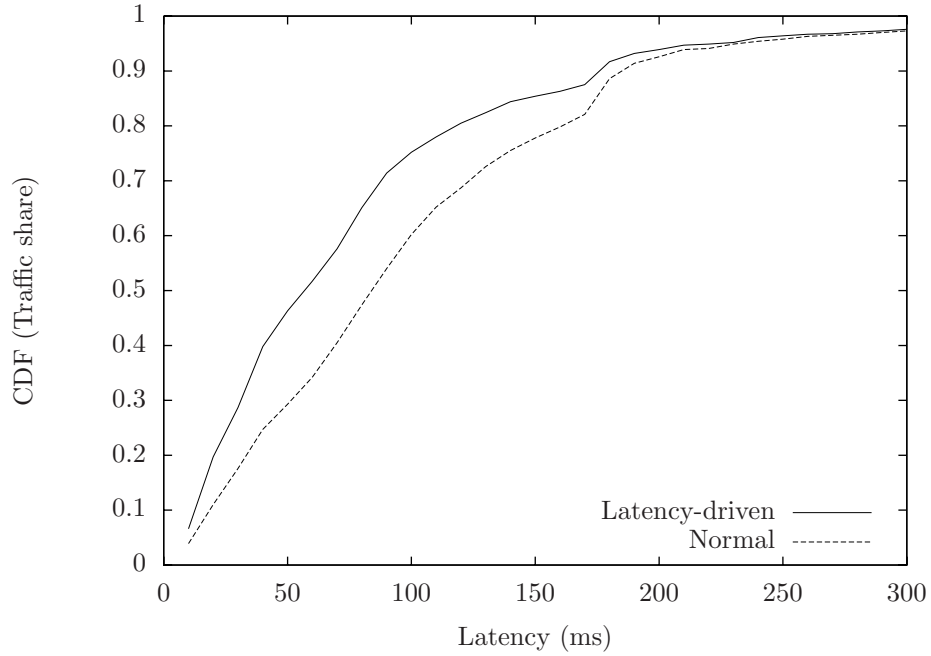

Figure 5.5: Amount of traffic exchanged over links with latency $< x$

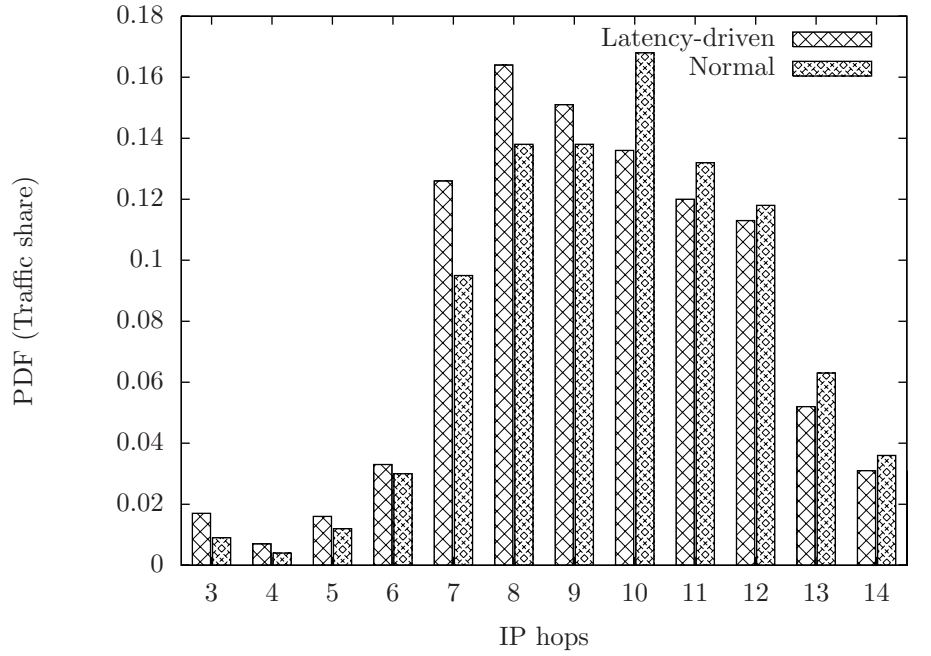

Figure 5.6: Amount of traffic exchanged over links with x IP hops

traffic cost. The number of IP hops is reduced by 2 on average. When using latency-driven BitTorrent, 75% of traffic goes over $< 100ms$ links against 60% for normal BitTorrent.

5.2 PlanetLab Evaluations

To evaluate our implementation in a real network and compare it to the simulator results we also deployed our system on PlanetLab and performed several experiments.

The implementation we used for the evaluation differs from the implementation described in Chapter 4 in one way. At the time of the experiment it was not possible to monitor packets sent during the TCP handshake on PlanetLab nodes. We therefore replaced SYNACK/ACK measurements with Double segment/ACK measurements.

5.2.1 Download time

We deployed the mainline BitTorrent client (5.2.0 [Bita]) on 150 PlanetLab nodes and let them download random data from a single, high-capacity seed. We ran each experiment 5 times with the original BNBT tracker and 5 times with the latency-driven BNBT tracker on the same set of nodes. We obtained the download times by processing the tracker log produced during the experiment.

Stable network

For the evaluation we are primarily interested in a stable scenario in which the tracker already knows all peers. However, on PlanetLab our experiments should not run for extended periods. The amount traffic generated would be inappropriate use of the facilities. Instead, we take a more practical approach by letting all peers join the network before returning any peer samples. This allows us to compare a fully random graph against a fully latency-biased graph.

The scenario proceeds in two rounds. In the first round all peers register to the tracker and receive the addresses of the landmarks in their peer sample. In the second round all peers have been positioned and the peers start obtaining biased peer samples. Since peers join in sequence and may be contacted before obtaining a peer sample the delay of the first round is non-constant among peers. However, the delay is stable across runs, making it suitable for comparison. They delay is part of the download time in both experiments.

Figure 5.7 displays the download time distribution for a 100MB file, including the first round. These results match those found in the simulator for a bias of 0.5, but turn out much lower for bias 0.9. The reason for this is not quite clear, but we suspect the BitTorrent client will itself apply some bias against peers with poor connectivity. When all active connections are latency-biased the flow of pieces into the network is harmed and all download times are reduced.

Growing network

In the simulator we were able to show significant download time improvements in a growing network, despite not being able to know whether better peers will be available in the future.

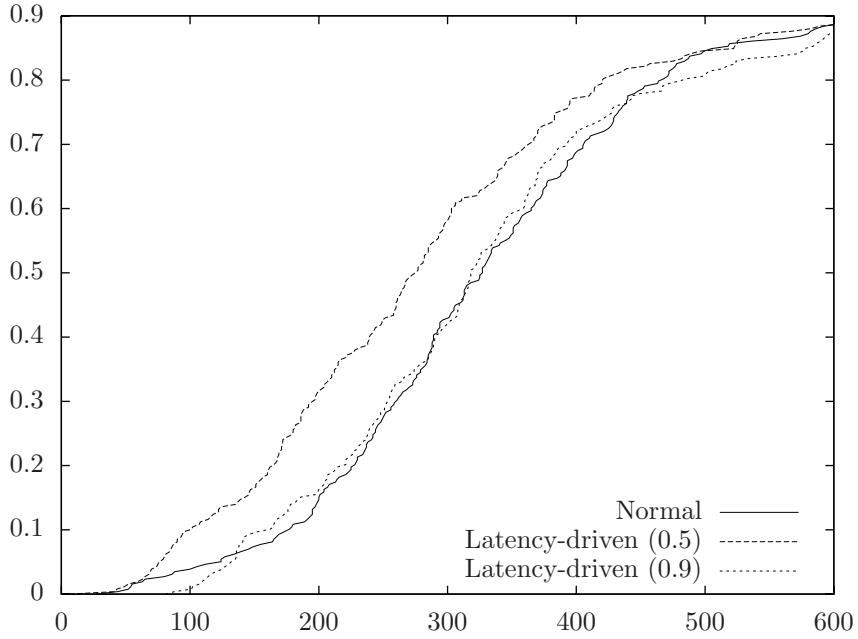

Figure 5.7: Download time for PlanetLab nodes in a stable network

However, the simulator did not factor in the measurement delay. When a client joins the network its position is unknown, at this point the tracker gives it the set of landmarks and a number of random peers. It takes until the client reinvokes the tracker before it can receive a biased peer set. Although this delay is small and does not prevent clients from downloading, it causes many registered peers to have unknown positions.

To find the extent to which can benefit growing networks in practice, we perform a second experiment on PlanetLab, in which we incrementally add peers and have the tracker operate normally. We use the adaptive approach from the simulator experiment to select the sample size for the Latency-driven tracker. The measurement delay is set at 30 seconds and consecutive invocations at 5 minutes. The join rate was again taken from the Izal dataset. We emulate the first part of the upcoming flash crowd in the dataset, with the constraint that we are only able to add the first 150 peers.

Despite our earlier result we return to a bias of 0.9 for this experiment. The reason for this is that peers now also receive random peers in their initial peer sample. By applying higher bias to the next sample the total bias is still approximately 0.5.

We obtain download times for a 256MB file from the tracker and find the distribution displayed in Figure 5.8, with a reduction of median download time of 14%. We improve performance for most peers, except those with the very high or low capacity. The high-capacity peers manage to finish most of their download before the measurement delay passes and thus only benefit from the larger peer sample they receive from a regular tracker. The low-capacity peers are constrained by their download limit and thus they can not experience any performance improvements.

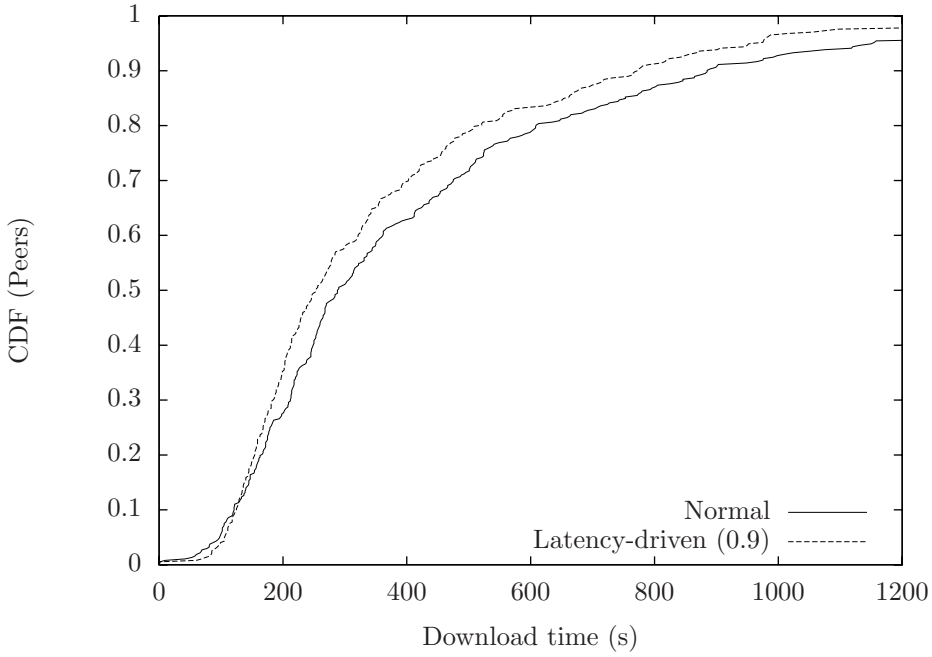

Figure 5.8: Download time for PlanetLab nodes in a growing network

5.2.2 Traffic cost

To determine traffic cost we need to know how much data every peer sends to every other peer. Unfortunately the standard BitTorrent client does not offer a convenient way to determine these values. We therefore repeat the (stable) experiments with the cTorrent [Hol] client, which has a very detailed debugging output. Using the raw output of the cTorrent client we were also able to obtain the exact amounts of data transferred between every pair of clients. With these numbers we can compute the traffic cost induced by our experiment in the same way as we did before.

In the PlanetLab experiment we do not significantly reduce Tier-1 traffic, but the results also indicate a possible reason. With either tracker the Tier-1 traffic is 32% of the total traffic for and thus PlanetLab networks use much less transit than regular networks. This is most likely due to the fact that much of the traffic between universities is routed over research networks such as GÉANT 2 and Internet 2. The notions of peering and transit we described cannot be easily applied to PlanetLab.

The second reason for finding lower cost reduction is that our improvements are much less effective when using cTorrent. The IP hop distribution in Figure 5.9 shows some improvement, but at the same time a reduction in traffic over very short paths.

cTorrent includes a number of bandwidth- and latency-based optimizations that turn out to be very effective. In general we find the median download time in the cTorrent experiment is up to 20% lower than that of the mainline BitTorrent client using a regular tracker. Latency-biased peer selection can only further improve download time by 5% for a network of cTorrent clients. Clearly widespread adoption of the cTorrent client could potentially be more beneficial than our tracker optimization, but as we pointed out before

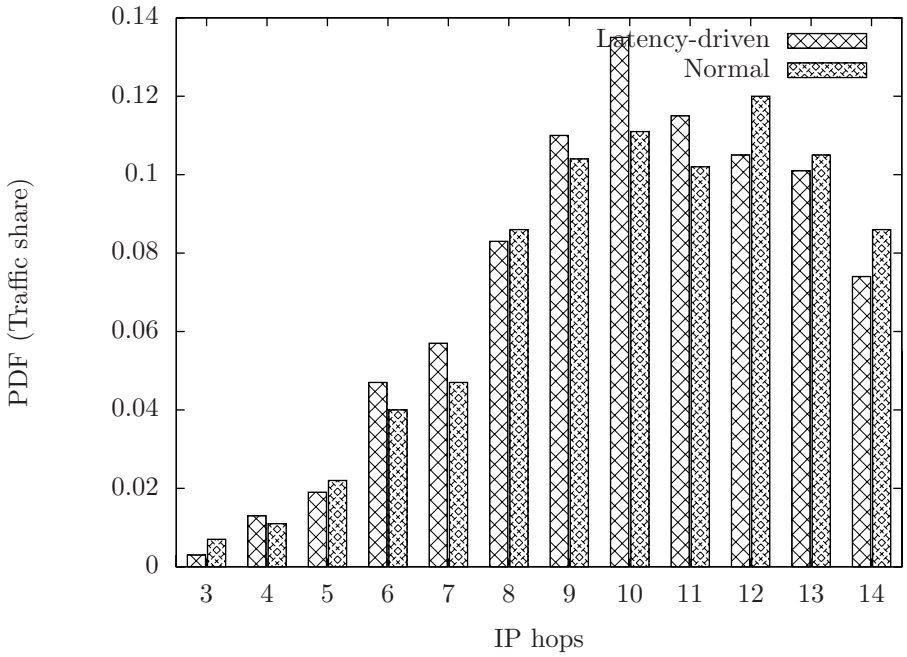

Figure 5.9: Amount of traffic exchanged over links with x IP hops

such adoption is difficult to achieve. We expect most implementations to be similar to the original BitTorrent implementation.

5.3 Discussion

We took a fairly pessimistic approach in our evaluation, using small networks without surplus capacity. Nonetheless we were able to achieve significant gains in both simulations and PlanetLab experiments using our relatively simple architecture. We find our gains are much higher in network with a large number of seeds, with is very common for popular BitTorrent networks. We have also shown for the first time that rapidly growing BitTorrent networks benefit from smaller sample sizes, and that when this is applied, latency-bias will thrive in such a network. In the simulations we were able to reduce traffic cost in several ways. Although we could not reproduce the traffic cost reduction in the PlanetLab experiment, the similarity in download time leads us to believe our simulations are fairly accurate. Even so, we did find different behaviour from different BitTorrent implementations and we found that values such as the optimal selection bias may depend on this. We intend to evaluate our system in real BitTorrent networks in the future.

Chapter 6

Related work

Several previous approaches to bias BitTorrent traffic towards nearby peers in order to improve download time and network cost exist. The main difference with our work is that they rely on external infrastructure or client-side modifications, while our approach only requires tracker-side modifications. In this chapter we identify the specific differences with each approach.

6.1 ISP bias

In the initial work done by Bindal et al. [BCC+06] peer selection was biased to nodes in the same ISP. Classifying hosts by ISP is a difficult problem for which the authors present a number of possible solutions. A tracker can use an Internet topology map, but this is typically hard to obtain and process. An alternative is to have ISPs publish their IP ranges or to append a tag to HTTP requests to the tracker. ISPs could also choose to perform peer selection themselves by modifying tracker responses or refusing outside connections. The last approach is in fact used in practice [Prea], but is generally not appreciated by users.

Using their approach the authors were able to show very strong reduction of the number of times the same blocks of data passed the ISP gateway. The simulations assumed access link limits to be the only bandwidth bottleneck and thus did not study the effects of the wide-area network. The results did show a reduced variation in download time.

A problem with the approach is that it assumes that a considerable number of peers are within the same ISP. Normally this would only be the case in very large networks.

6.2 iPlane

Madhyastha et al. evaluated latency-biased peer selection as part of the iPlane project [MIP+06]. iPlane is an oracle for network conditions. It is designed to actively monitor network routes from hundreds of dispersed hosts in order to build a single, coherent view of the Internet. iPlane can be queried by any Internet application through a Web Service.

The authors present different use cases to evaluate iPlane, among which a biased

BitTorrent tracker. Rather than biasing peer selection just by network distance, the iPlane tracker also uses loss rates to compute the expected TCP throughput between its clients. Every peer sample consists partly of random peers and partly of peers with the highest TCP throughput. In an experiment on PlanetLab the biased tracker improved download time for roughly 50% of peers in a BitTorrent swarm by biasing half of the peer sample.

The fine-grained network measurements and models of iPlane can be expected to produce more accurate results than GNP, but they require a vast infrastructure. This infrastructure is deployed on PlanetLab for research purposes, but deploying a commercial or public alternative would be far more expensive than a cooperative GNP system. More importantly, for a large tracker iPlane has considerable overhead. In a 10,000 peer BitTorrent network a tracker would have to perform on the order of 100,000,000 pairwise lookups to know which peers are closest to each other. In this case invoking the web service is not an option. The alternative is to download a multi-GB Internet map on a daily basis, but even then the computation is considerably more expensive than computing GNP distances, which can also be strongly optimized using spatial data structures.

6.3 P4P

P4P [XKLS07] is an architecture in which ISPs publish their network information and preferences for peer-to-peer networks to use. This solution is similar to iPlane in that the ISPs provide network oracles, but the ISPs can provide much more fine-grained and knowledgeable network information. The disadvantage is that only information from participating ISPs and client implementations is available. This may lead to poorly informed decisions for the user, such as selecting known low-capacity peer over unknown high-capacity peers. It is still unclear how P4P will perform on an Internet scale with only part of the networks and/or part of the peers participating.

It also remains to be seen to what extent peer-to-peer networks will be willing to adopt the approach. Participation in peer-to-peer network is often a privacy sensitive issue and the P4P architecture requires ISPs to have knowledge of all peers in a swarm.

6.4 Ono

Ono [CB08] is an approach to bias peer-to-peer traffic towards nearby neighbours using only existing infrastructure. Ono peers derive proximity using the redirection mechanisms of Content Delivery Networks (CDN). A CDN is a web hosting provider which replicates documents on a large number of servers world-wide. When requesting a document clients are typically redirected to the closest replica through DNS.

Ono peers use the DNS names they are redirected to when querying for the names of popular websites at large CDNs as a set of coordinates. The vector of DNS names can be exchanged and compared for similarity. In all likelihood two peers with similar vectors are relatively close. The Ono Azureus plug-in prioritizes traffic among Ono peers by letting them keep each other in the set of candidates for optimistic unchoke. This way Ono peers

will unchoke each other more often than others and typically exchange more traffic.

The Ono plug-in has been adopted by over 120,000 Azureus users and this has allowed the authors to measure Ono's effectiveness in actual torrent networks. The Ono approach of finding nearby peers is effective, but unfortunately it does not lead to median performance gains for BitTorrent in regular networks. Ono does perform very well in networks with higher internal than external bandwidth.

6.5 Discussion

The existing approaches to bias BitTorrent traffic each have their advantages, but they do not bring quick relieve to an urgent problem. The systems each have some form of deployment, but it may take years to establish the cooperations and software deployment required for them to significantly reduce long-distance BitTorrent traffic. Our approach can be deployed immediately by trackers by finding a few other trackers to cooperate with. The most popular trackers may even be able to deploy it without any help, provided their servers are geographically dispersed.

Chapter 7

Conclusion

This thesis presents the design and implementation of Latency-driven BitTorrent, an effective approach to improve client performance and reduce ISP traffic cost in BitTorrent. The tracker uses coordinate-based latency prediction to bias peer samples towards peers near the requester. The originality of our system is that it requires no patches to client software or network oracles. Instead, the latency-driven tracker only needs access to a small number of geographically dispersed landmarks. Trackers can provide each other with landmarks in a cooperative fashion.

We evaluated the approach using simulation and Planet-Lab experiments. The simulator was carefully designed to model relevant network conditions using a realistic dataset. In both environments we could show a improvement of the median download time of 10-15% for networks of 100-200 nodes with a single seed and up to 25% when using a large number of seeds. We find that we can also achieve high gains in a growing network by reducing the sample size. We reduce expensive Tier-1 traffic by 12% and in general data is sent over much shorter distances.

We have shown that trackers can significantly reduce the amount of long-distance traffic generated by BitTorrent without having to wait for advanced measurement infrastructure or large-scale software deployment to become available. Using latency-driven BitTorrent trackers can directly benefit their users, while reducing pressure on Internet infrastructure and operational cost of ISPs. Based on our results we expect many BitTorrent trackers to adopt our approach or implement similar optimization policies in the future.

A few areas remain to be explored. We have yet to study our approach in real BitTorrent networks. The simulator and Planet-Lab each have their limitations with regard to the evaluation of our approach. For example, real BitTorrent clients may be in multiple networks and have limited access link capacity available, which may reduce the gains we make in download speed. On the other hand, we also have not studied the potentially higher gains in very large networks yet.

Regarding the approach itself, an open issue is performance. Although computing GNP distances is computationally cheap, ordering them is at least an $O(N)$ operation for network size N. However, finding nearby peers could be made much more efficient using spatial data structures such as quad trees [FB74] or more elaborate ranking schemes [HS95].

Finally our finding of reducing in the sample size to increase performance in growing network needs to be explored further. We have not yet studied or developed mechanisms to detect the state of the network and select an appropriate sample size. These mechanisms may have a lot of potential to handle flash crowds in BitTorrent more efficiently.

Acknowledgments

We would like to thank all people who have made this research possible. In particular we would like to thank Harsha Madhyastha from the University of Washington for kindly providing access to the iPlane measurement and prediction data [MIP+06]. We would also like to thank Michal Szymaniak, currently at Google, for providing his SYNACK/ACK and GNP implementations used in his earlier work [SPPvS08].

Bibliography

[ASW+02] Matthew Andrews, F. Bruce Shepherd, Peter Winkler, Aravind Srinivasan, and Francis Zane, *Clustering and Server Selection using Passive Monitoring*, INFOCOM, 2002.

[Aue04] Karl Auerbach, *Limitations of icmp echo for network measurement.*

[Axe84] Robert M. Axelrod, *The evolution of cooperation*, Basic Books, New York, 1984.

[Azu] Azureus, *Distributed hash table*, `http://www.azureuswiki.com/index.php/DistributedTrackerAndDatabase`.

[BCC+06] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, and Amy Zhang, *Improving Traffic Locality in BitTorrent via Biased Neighbor Selection*, 26th IEEE International Conference on Distributed Computing Systems (26th ICDCS'2006) (Lisboa, Portugal), IEEE Computer Society, July 2006, p. 66.

[BHP06] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan, *Analyzing and Improving a BitTorrent Networks Performance Mechanisms*, INFOCOM, IEEE, 2006.

[Bita] Inc BitTorrent, *Bittorrent*, `http://www.bittorrent.com/download`.

[bitb] bittorrent.org, *Bittorrent specification*, `http://www.bittorrent.org/beps/bep_0003.html`.

[CB08] David R. Choffnes and Fabin E. Bustamante, *Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems*, Proceedings of ACM SIGCOMM 2008, August 2008 (English).

[CBAT] Zesheng Chen, Tian Bu, M. Ammar, and D. Towsley, *Comments on Modeling TCP reno performance: a simple model and its empirical Validation*, .

[CNE] CNET, *AT&T: Internet to hit full capacity by 2010*, `http://news.cnet.com/ATT-Internet-to-hit-full-capacity-by-2010/2100-1034_3-6237715.html`.

[Coh03] B. Cohen, *Incentives Build Robustness in BitTorrent*, Proceedings of the Workshop on Economics of Peer-to-Peer Systems (Berkeley, CA, USA), 2003 (English).

[DCKM04] Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Morris, *Vivaldi: a decentralized network coordinate system*, SIGCOMM (Raj Yavatkar, Ellen W. Zegura, and Jennifer Rexford, eds.), ACM, 2004, pp. 15–26.

[DKH06] Roman Dunaytsev, Yevgeni Koucheryavy, and Jarmo Harju, *The PFTK-model revised*, Computer Communications **29** (2006), no. 13-14, 2671–2679.

[FB74] R. A. Finkel and J. L. Bentley, *Quad trees: A data structure for retrieval on composite keys*, Acta Informatica **4** (1974), no. 1, 1.

[Gro00] NIST Internetworking Technology Group, *NISTNet network emulation package*, http://www.antd.nist.gov/itg/nistnet/ (2000).

[HLB07] Anwar Al Hamra, Arnaud Legout, and Chadi Barakat, *Understanding the properties of the bittorrent overlay*, CoRR **abs/0707.1820** (2007), informal publication.

[Hol] Dennis Holmes, *Enhanced ctorrent*, `http://www.rahul.net/dholmes/ctorrent/`.

[HS95] G. R. Hjaltason and H. Samet, *Ranking in spatial databases*, Lecture Notes in Computer Science **951** (1995), 83–95.

[IUKB+04] Mikel Izal, Guillaume Urvoy-Keller, Ernst W. Biersack, Pascal Felber, Anwar Al Hamra, and Luis Garcés-Erice, *Dissecting BitTorrent: Five Months in a Torrent's Lifetime*, PAM (Chadi Barakat and Ian Pratt, eds.), Lecture Notes in Computer Science, vol. 3015, Springer, 2004, pp. 1–11.

[JBB92] V. Jacobson, R. Braden, and D. Borman, *TCP extensions for high performance*, RFC 1323, Internet Engineering Task Force, May 1992.

[JD02] Hao Jiang and Constantinos Dovrolis, *Passive estimation of TCP round-trip times*, Computer Communication Review **32** (2002), no. 3, 75–88.

[Loe] Andrew Loewenstern, *DHT protocol*, `http://www.bittorrent.org/beps/bep_0005.html`.

[LSB+05] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca, *Measuring Bandwidth Between PlanetLab Nodes*, PAM (Constantinos Dovrolis, ed.), Lecture Notes in Computer Science, vol. 3431, Springer, 2005, pp. 292–305.

[LWM] LWM.net, *Tcp window scaling and broken routers*, `http://lwn.net/Articles/92727/`.

[MIP+06] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas E. Anderson, Arvind Krishnamurthy, and Arun Venkataramani, *iPlane: An Information Plane for Distributed Services*, OSDI, USENIX Association, 2006, pp. 367–380.

[NM65] J. A. Nelder and R. Mead, *A simplex method for function minimization*, Computer Journal **7** (1965), 308–313.

[Nor00] William B. Norton, *Internet service providers and peering*, Proceedings of NANOG 19 (Albuquerque, New Mexico), June 2000.

[NZ02] T. S. Eugene Ng and Hui Zhang, *Predicting Internet Network Distance with Coordinates-Based Approaches*, INFOCOM, 2002.

[OCP+06] David L. Oppenheimer, Brent N. Chun, David A. Patterson, Alex C. Snoeren, and Amin Vahdat, *Service Placement in a Shared Wide-Area Platform*, USENIX Annual Technical Conference, General Track, USENIX, 2006, pp. 273–288.

[PFTK98] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose, *Modeling TCP throughput: A simple model and its empirical validation*, SIGCOMM, 1998, pp. 303–314.

[Prea] Associated Press, *Comcast blocks some Internet traffic*, `http://www.msnbc.msn.com/id/21376597/`.

[Preb] ———, *Time Warner Cable tries metering Internet use*, `http://www.msnbc.msn.com/id/24936796/`.

[rfc89] *Requirements for Internet Hosts - Communication Layers*, RFC 1122, Internet Engineering Task Force, October 1989.

[SM07] Hendrik Schulze and Klaus Mochalski, *Internet Study 2007: The Impact of P2P File Sharing, Voice over IP, Skype, Joost, Instant Messaging, One-Click Hosting and Media Streaming such as YouTube on the Internet*, Tech. report, ipoque, 2007.

[SPPvS08] Michał Szymaniak, David Presotto, Guillaume Pierre, and Maarten van Steen, *Practical large-scale latency estimation*, Elsevier Computer Networks **52** (2008), no. 7, 1343–1364, `http://www.globule.org/publi/PLSLE_draft2006.html`.

[the] theory.org, *Unofficial BitTorrent specification*, `http://wiki.theory.org/BitTorrentSpecification`.

[Tor] TorrentFreak, *utorrent gains popularity, azureus loses ground*, `http://torrentfreak.com/utorrent-gains-popularity-azureus-loses-ground-071216/`.

[W3C] W3Counter, *Global web stats*, `http://www.w3counter.com/globalstats.php`.

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes, *The network weather service: a distributed resource performance forecasting service for metacomputing*, Future Generation Computer Systems **15** (1999), no. 5–6, 757–768.

[XKLS07] Haiyong Xie, Arvind Krishnamurthy, Yanbin Liu, and Avi Silberschatz, *P4P: Explicit Communications for Cooperative Control Between P2P and Network Providers*, P4PWG Whitepaper (2007).

[ZDN] ZDNet, *CacheLogic says 35% of all Internet traffic is now BitTorrent*, .