Can Self-Organizing P2P File Distribution Provide QoS Guarantees?

Ruchir Bindal and Pei Cao
Department of Computer Science, Stanford University
{*rbindal, cao*}*@cs.stanford.edu*

ABSTRACT

In this paper, we examine the factors that contribute to the variability in download time of a self-organizing P2P file distribution application such as BitTorrent. We conducted a series of side-by-side live experiments, involving two clients running on the same machine downloading the same file at the same time. We found that the download latency varied significantly, sometimes by a factor of 2. Surprisingly, the main contributing factor isn't the network bandwidths of the set of neighbors that a client is given. Rather, it has to do with the frequency of turn-overs in "close" neighbors, i.e. those that are in a stable data-exchange relationship with the client. Analysis of the log data shows that a client obtains over 90% of the file from a small set of close neighbors, and if a close neighbor leaves the network, it takes the client a long time, *over half an hour*, to find another one. This suggests that self-organizing P2P file distributions indeed need external help in order to provide QoS guarantees, but such guarantees are achievable with proper enhancements to the P2P network.

1. INTRODUCTION

Using end-hosts (aka peers) to distribute files is fundamentally more scalable than using central servers. The supply of bandwidth grows linearly with demands. There are no central bottlenecks such as servers or links to servers. Thus, it is no surprise that many large content providers are looking into the peer-to-peer approach for content delivery [5].

However, a main challenge for peer-to-peer content distribution is the lack of quality-of-service guarantees. Since peers are not under the control of a central scheduler, it is difficult to guarantee to a peer that, upon joining the network, it will obtain the whole file within a certain time window. While this is not a problem when people do not pay for the contents, it becomes an issue when they do.

In this paper, we investigate the question: can peer-to-peer file distribution applications, such as BitTorrent, provide quality-of-service guarantees? Put it another way, what contributes to the variation in download times in P2P file distribution? BitTorrent is chosen as the application to study, not only because of its popularity, but also because of its simplicity.

To conduct the investigation, we ran a series of "side-by-side" live experiments on the Internet. Two BitTorrent clients were started on a desktop machine on the campus network. They joined the same network at the same time and attempted to download the same file. We measured how long it took for each client to complete the download. We also logged all interactions between a client and its peers, then analyzed the logs to examine what made the slower client take longer.

The findings are somewhat surprising. The upload speeds from neighbors plays a minor role in determining download latency. Instead, the client's experience is very much affected by a set of "close" neighbors. Though a client connects to thousands of neighbors during its download, it obtains the majority of the data from a very small (10s to 50s) set of neighbors (we call these the "close" neighbors). At any point during the download, the client is involved in stable data-exchange relationships with about 7 neighbors. The client constantly uploads data to these neighbors, and it constantly receives data from these neighbors. If one neighbor leaves, however, it takes the client a long time (e.g. 30 minutes) to find another one. In the majority of cases, a close neighbor leaves because it is disconnected from the network abruptly (e.g. mobile hosts), or it has finished its download. *The slower client is slow because it has more turn-overs in its close neighbors.*

The results suggest that BitTorrent, through its "tit-for-tat" scheme, essentially runs a completely distributed, randomized algorithm to establish matchings between the peers, i.e. forming a random graph where each node has a degree of 7 (each node is allowed to upload to up to 7 neighbors concurrently). Unfortunately, the algorithm appears to take a long time to converge if the matching is disturbed, e.g., when a neighbor leaves. Since peers come and go on the Internet, the results suggest that the current BitTorrent algorithm will have difficulties providing predictable download times to peers.

The analysis also suggests possible techniques to address the problem, including dedicated high-speed "seed" peers that are directed by trackers to help slow clients, and enhancements to the BitTorrent protocol to enable fast convergence time. Investigation of these techniques is part of our future work.

2. BITTORRENT MECHANISMS

BitTorrent [4] and similar applications are the most popular P2P file distribution applications on the Internet today. Various reports estimate that BitTorrent traffic accounts for a quarter of the total Internet traffic today [6]. BitTorrent aims to distribute large files to a large user population efficiently. This is achieved by making use of the **upload** bandwidth of all nodes (called peers) **downloading** the file. In the following description, the terms node and peer are used interchangeably.

To distribute a file via BitTorrent, the provider first generates a separate file containing some meta-information about the shared file (called the *torrent* file). This *torrent* file contains the address of the *tracker* (described below) and also the size of each of the small file blocks that are exchanged between the peers. The provider runs a *tracker* for this file, either on its own or by registering the *torrent* file with a public tracker. A *tracker* is a central server that keeps track of all nodes downloading this file.

The supplier then starts its BitTorrent client. The client automatically detects that it has the complete file and is thereby a *seed* node in the network. The torrent file is then published on the Internet using HTTP and interested downloaders can download it to run their BitTorrent clients with the torrent file as the input. Since the tracker's address is already embedded in the torrent file, the clients can contact the tracker using it.

2.0.0.1 Connecting to Neighbors

A peer who is interested in downloading the file first contacts the tracker hosting the file to join the BitTorrent network *of this file*. The network consists of the tracker and all nodes downloading the file. The tracker **randomly** selects C nodes (default $C = 50$) to give to p. Peer p then initiates connections with those nodes. Whenever p's neighbor count drops below a certain threshold (default is 20), p contacts the tracker again to obtain a new list of nodes.

2.0.0.2 Choking/Unchoking

The download proceeds mainly by peers exchanging blocks with their neighbors. Peers exchange bit vectors of the blocks in their possession with neighbors frequently, both at the beginning and whenever a peer obtains new content. Through the bit vector exchange, the peer p learns the up-to-date content at each neighbor. If a neighbor has blocks that p doesn't have, p sends an "interested" message to the neighbor. The neighbor, however, is not obligated to send blocks to p. Instead, when and if the neighbor sends blocks to p depends on the "choking/unchoking" algorithm in BitTorrent, also called the "tit-for-tat" mechanism.

For each peer, there may be many neighbors interested in its content. The "choking/unchoking" algorithm determines which of them it should give the contents to. All connections are "choked" by default. If a peer decides to provide contents to another peer, it "unchokes" the connection to it. A peer can upload to multiple peers at the same time; in our client, the default limit on the number of concurrent uploads is 7. Hence, a peer has 7 unchoked connections at a time.

Six of these connections are chosen using a "tit-for-tat" criteria. A node keeps track of the rates at which it downloads blocks from all neighbors. The rate is calculating by observing how much data are received from a neighbor over a time interval of about 20 seconds, and dividing the amount of data by the time interval. Thus, the rate takes into account not only the upload speed of the neighbor, but also its consistency at giving the client data. Then, among the neighbors expressing interest, six with the highest rate to this node are unchoked. In other words, a peer rewards other peers who previously gave data to it consistently at good speed. The decision to choke/unchoke is made periodically (every 10 seconds), after which some of the currently unchoked nodes may be choked and vice-versa.

For any seed (i.e. those that have the whole file), this decision is made based on the rates that the peers can receive data from it. So the peers which download quickly from the seed get a higher preference.

A pure "tit-for-tat" scheme would not allow brand new peers, i.e. peers not holding any piece, to start downloading from anybody. To solve this problem, BitTorrent uses a mechanism called *optimistic unchoke*. Every 30 seconds, a node chooses a neighbor at random to unchoke, irrespective of the download rate from that neighbor. The neighbor is the last of the 7 connections unchoked by a peer. In addition to bootstrapping new peers, this mechanism also allows a peer to find other peers that may have a better upload rate to it. The chance of a new node being selected for an optimistic unchoke is three times that of a node that already holds some blocks.

2.0.0.3 Piece Selection

Once a peer p expresses interest in a neighbor's blocks, and the neighbor unchokes the connection with p, p can request a block from the neighbor. Exactly which block is read from the neighbor is determined by the piece selection algorithm.

The default behavior in BitTorrent is that p uses a "rarest first replication" algorithm. That is, among the blocks provided by a neighbor, the block that is least replicated among all neighbors of p is chosen. Note that this is a local rarest first; a node only has visibility into the contents of its neighbors. It is not a global rarest first.

3. OVERVIEW OF EXPERIMENTS

We conducted a series of experiments with two BitTorrent clients running side by side on the same host. They were started simultaneously and they downloaded the same file. In order to prevent the two clients from connecting to each other (and thereby sharing the file among themselves), each client was explicitly programmed to avoid connecting to the host machine's IP.

For both clients, the maximum number of concurrent uploads was set to 7 and no cap was imposed on the upload bandwidth to be used. The maximum number of two-way connections was set to 50, and the maximum number of connections initiated by the peer was 35. In our experiments, it appears that external peers actually could not connect to our clients, possibly due to the firewall restrictions at Stanford. Thus, the maximum number of neighbors that the peer could talk to at the same time was 35. The environments for the two clients were kept as identical as possible. They ran on the same host; they essentially had the exact same upload and download bandwidths. They ran at the same time, so the conditions of the Internet, and of the BitTorrent network, were also the same. All parameters and policies were configured the same in both clients.

In the experiments, the original BitTorrent client [3] is changed to log all significant events. For each event, the log message includes the event ID, timestamp, the IP and port number of the neighbor that the event corresponds to, and event-specific information. At the end of each experiment, the client's log file is saved for further analysis. The events logged are:

- Tracker: Response received for a request sent to the tracker. Logged message also includes the number of neighbors received in the response.
- Connect: Connection request sent to a neighbor.
- Choke: Choke message received from a neighbor.
- Unchoke: Unchoke message received from a neighbor.
- PChoke: Choke message sent to a neighbor.
- PUnchoke: Unchoke message sent to a neighbor.
- Interested: Interested message sent to a neighbor.
- Ninterested: Not Interested message sent to a neighbor.
- Ginterested: Interested message received from a neighbor.
- Guninterested: Not Interested message received from a neighbor.
- Piece: Data received from a neighbor. Logged message also includes the amount of data received and block number in the file this data belongs to.
- Lost: Connection to a neighbor is lost.
- Close: Connection to a neighbor is closed by the client.
- Bitvector: Bitvector received indicating which file blocks are present at the neighbor.
- Have: The client now has a full file block. Logged message includes the block number which the client has finished downloading.

Table 1 shows a summary of the results. For each experiment, we estimate the size of the BitTorrent network for the file by merging the list of neighbors seen by the two clients. In all experiments the network has about 1500 to 2200 peers, which means that during the lifetime of the downloads there are thousands of other peers downloading the file as well. The effective download rate, which is simply $(filesize/download - duration)$, ranges from 10KB/s to 30KB/s. Though our host is on a university network, the majority of the peers are probably cable-modem peers, which explains the low effective throughput. The results show that, even though the two clients were configured the same and ran in the same external environments, the download time differs by up to a factor of 2. In real time this would mean a difference of 6 to 12 hours, which would not be tolerated by users who pays for the download.

Exp.	Ratio of download times	Ratio of R_1	Ratio of R_2	Ratio of R_3
1	1.82	1.10	1.03	0.97
2	1.26	1.29	1.23	1.28
3	1.09	0.70	0.71	0.66
4	1.22	1.24	1.43	1.55
5	1.58	1.54	1.87	3.62
6	1.50	1.20	1.19	1.07
7	1.27	0.91	0.81	0.60
8	1.63	1.06	1.20	1.48
9	1.03	0.88	0.79	0.70
10	1.63	2.47	2.54	3.53
11	1.21	1.26	1.26	1.12
12	1.78	1.34	1.33	1.45
13	1.28	1.15	1.37	1.17

Table 2: Ratios of download times and three estimates of unweighted average download rates. The download time ratio is calculated as *(time for slower client / time for faster client)*. The average rate ratios are *(avg. rate for faster client / avg. rate for slower client)*. R_1 is the estimate including all peers, R_2 includes only those who provide at least 256 KB of data, R_3 includes only those who provide at least 2.5 MB of data.

4. UNDERSTANDING THE CAUSE OF PERFORMANCE VARIATION

Despite the fact that the two clients run with identical configurations, there are big differences in the download times that they experienced in almost all cases (ref. Table 1). A natural question is: what made the slower peer unlucky?

4.1 Per-Slot Analysis

We first partition the download time into idle and active times, and calculate the download concurrency (i.e. how many neighbors are sending data to it at any point in time) for the client. For each TCP connection between the client and a neighbor, the download state, i.e. the state of data flowing from the neighbor to the client, alternates between waiting for data and actually receiving data. We call the state that the client is waiting for data the *idle* state, and the state that it is actually receiving data the *active* state. The download can become idle either because the client is choked by the neighbor or because it is not interested in the neighbor's data. Similarly, the download is active when the client is interested in the neighbor *and* is unchoked by it.

The log analysis tool first aggregates information for each connection. Each connection is modeled as a finite state machine. As the log lines are read, a connection's state is

Exp. #	File size	Exp. Date	Network Size	Download time (secs.)		Effective Download rate (Kilo-Bytes/sec.)	
				Client 1	Client 2	Client 1	Client 2
1	700 MB	April 11 '06	1563	23,823	43,401	30.8	16.9
2	1400.5 MB	April 12 '06	2070	51,219	64,589	28.7	22.7
3	702 MB	April 12 '06	1851	55,982	61,034	13.1	12.1
4	700 MB	April 13 '06	1246	22,688	27,594	32.4	26.6
5	702 MB	April 13 '06	1837	40,409	63,951	18.2	11.5
6	1400.5 MB	April 14 '06	1959	47,399	70,977	31.0	20.7
7	700.5 MB	April 14 '06	2112	51,304	65,128	14.3	11.3
8	700.75 MB	April 15 '06	2141	43,767	71,526	16.8	10.3
9	699.75 MB	April 29 '06	1170	40,324	41,715	18.2	17.6
10	702 MB	April 29 '06	1169	27,258	44,375	27.0	16.6
11	700.25 MB	April 29 '06	1384	33,247	40,194	22.1	18.3
12	701.5 MB	May 4 '06	1435	40,344	71,826	18.2	10.2
13	701 MB	May 4 '06	1614	52,421	67,026	14.0	11.0

Table 1: File size, Experiment date, Network size and Download time and Effective download rate in the various experiments. Network size refers to the total number of unique peers encountered by the two clients.

updated when its logged events are encountered, and the relevant information is stored in a hash table indexed by the neighbor's IP address.

The analysis tool then groups connections into "slots". Since the maximum number of connections is limited to 35, a client tries to fill all 35 slots in order to maximize its download speed. In other words, each slot is a resource and the client's goal is to maximize its utilization. The analysis tool assigns a connection to a slot on a first-come first-served basis. The tool marks a slot as active if there is a connection occupying it and the download state of the connection is active. Conversely, the tool marks a slot as idle if no connection occupies the slot (i.e. the total number of connections is less than 35), or the download state for the connection in the slot is idle. For each slot, its total idle time, total active time, and total data received are calculated.

Let T be the total download time of the client, and W be the average idle time of a slot. Since the default number of concurrent uploads in BitTorrent is 7, the average concurrent downloads that the client receives is also about 7 (the presence of seeds, which only upload, may increase this number). Let C be the average download concurrency that the client experiences. Since the total number of slots is 35, $W = T * (35 - C)/35$. Thus, C can be calculated using W.

Let S be the total file size. Let R_i be the speed at which neighbor i sends data to the client, and D_i be the amount of data that neighbor i sends to the client. Then the harmonic mean of the neighbor's speeds is $S/\sum_i D_i/R_i$, which measures the effective rate experienced by the client during the active times of the slots. Since D_i/R_i is essentially the total active time of the connection with the neighbor, the harmonic mean can also be calculated as S divided by the total active time over all slots.

Table 3 shows the average idle time, harmonic mean of download rates, and average download concurrency in each experiment. The results show that the slower peer is not slow because it has a lower download concurrency; in fact, in a few cases its download concurrency is higher. In most experiments, the download concurrency is quite similar for the two clients. Hence, the difference in performance is caused by the effective download speed that the client obtains from its neighbors (i.e. the weighted harmonic mean of the download speeds).

4.2 Impact of Neighbors' Upload Speeds

Naturally, our next question is: was the slower client unlucky because the tracker gave it a slower set of neighbors? To answer this question, we estimate the "upload speeds" of the neighbors. It is difficult to measure the throughput of the TCP connection with a neighbor from the application. However, we can estimate the network bandwidth by observing the amount of data arrived during an active interval. The estimate does not necessarily reflect actual network speed, but is useful for comparisons of the neighbors' speeds.

Specifically, for each connection, the analysis tool calculates its total active time (i.e. the duration when the download state is active) T_i, and the total amount of data that arrived from this connection D_i. The "upload speed" for the neighbor is set to be D_i/T_i. Three estimates are used: R_1 is unweighted average of all neighbors that ever give any data to the peer, R_2 is unweighted average of neighbors providing more than 256KB of data, and R_3 is that of those providing more than 2.5MB of data. Three estimates are used because the method of calculating the upload speed is inaccurate, and each estimate essentially represents a different way to sample the upload speeds. The relative ratio of the download times and of the rates of the two clients in our experiments are shown in Table 2

The results show that the "upload speed" is only a minor factor in determining the download time. Indeed, in 3 out of the 13 experiments, download is faster for the peer that has a slower set of neighbors. The correlation coefficient of the download time ratio and the average "upload speed" ratio is only 0.57 for R_1, 0.43 for R_2, and 0.47 for R_3. Thus, while "upload speed" does play a role in determining download time, its effect is moderate. Other factors are at play here.

4.3 Effect of "Close" Neighbors

In each experiment, the client connects to a lot of peers. However, not all peers upload data to it. In fact, the majority of the data is given by a handful of peers. Table 4 shows the number of unique peers that a client connects to dur-

Exp.	Client	Total download time (secs.) (T)	Avg. idle time (secs.) (W)	Avg. download rate (Bytes/sec.) (R)	Avg. Concurrency
1	1	23823.00	19623.56	3521.18	8.81
	2	43401.00	36091.60	2041.75	8.42
2	1	51219.00	42012.06	3207.98	8.99
	2	64589.00	51861.36	2324.59	9.85
3	1	55982.00	45913.18	1489.09	8.99
	2	61034.00	51906.68	1632.28	7.48
4	1	22688.00	19077.66	4106.17	7.96
	2	27594.00	23651.58	3741.93	7.14
5	1	40409.00	35190.96	2865.46	6.46
	2	63951.00	53207.12	1408.89	8.40
6	1	47399.00	37694.90	3035.93	10.24
	2	70977.00	56108.20	1989.55	10.47
7	1	51304.00	40044.62	1342.52	10.97
	2	65128.00	54450.10	1400.65	8.20
8	1	43767.00	34170.76	1557.26	10.96
	2	71526.00	56537.36	1012.79	10.48
9	1	40324.00	31852.40	1765.58	10.50
	2	41715.00	33277.98	1783.22	10.11
10	1	27258.00	22994.96	3472.64	7.82
	2	44375.00	35299.76	1660.82	10.23
11	1	33247.00	27451.22	5086.57	8.72
	2	40194.00	32952.00	4089.30	9.01
12	1	40344.00	33110.24	2058.47	8.97
	2	71826.00	58120.46	1100.94	9.54
13	1	52421.00	45063.66	2027.80	7.02
	2	67026.00	55409.32	1294.53	8.67

Table 3: Average idle time, download rate and concurrency in each experiment.

Exp.	Client	Total peers	Close peers	Data to close peers
1	1	618	16	86.05
	2	1065	32	85.47
2	1	1044	24	90.12
	2	1331	40	87.23
3	1	1138	27	80.47
	2	1250	31	90.17
4	1	612	14	77.62
	2	731	20	88.75
5	1	954	11	52.28
	2	1344	47	90.26
6	1	946	24	93.25
	2	1237	40	90.00
7	1	1142	32	81.01
	2	1360	43	88.84
8	1	1022	26	90.86
	2	1531	58	86.03
9	1	733	33	81.05
	2	790	28	77.51
10	1	578	14	78.18
	2	830	33	84.96
11	1	772	12	68.47
	2	892	22	89.40
12	1	777	27	83.54
	2	1140	48	73.66
13	1	999	28	74.89
	2	1094	34	87.14

Table 4: Total number of peers versus the number of close peers for each experiment. "Data to close peers" is the total amount of data uploaded to close peers as a percentage of the total data uploaded.

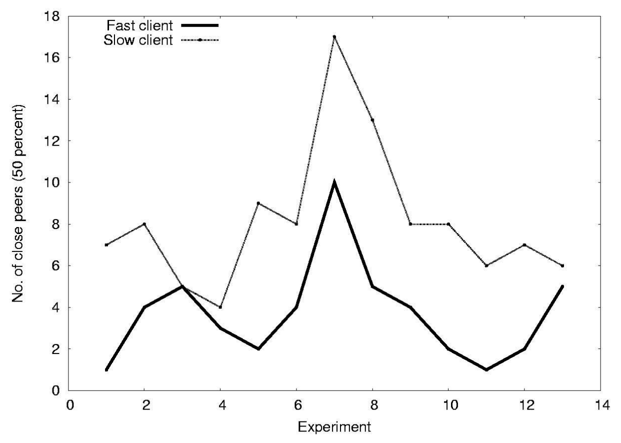
ing the course of its download, and the minimum number of peers that give the client 90% of the file. In other words, if we sort all peers by the amount of data that they give to the client, and find the top N peers that together give 90% of the file, the result N is a very small number. We call the top N peers "close" peers of the client.

As one can observe from the Table 4, the # of close neighbors *consistently* predicts performance. The smaller the number is, the faster the download proceeds. A client that gets its data from fewer peers always performs better. The only exception to this rule is experiment 9, but in this case the difference in download times between the two clients is very small. In fact, if one compares the minimum number of peers that provide 50% and 75% of the data to the client in each experiment, these numbers *consistently* predict performance as well. Figure 1 illustrates the results. In both graphs, the line representing the faster client is consistently lower than the line representing the slower client. This result leads us to examine the relationship between the client and its "close" peer more carefully.

4.3.1 *Matching Between the Peers*

What makes "close" peers close to the client? We initially guessed the following: that their upload speeds to the client are the highest among the peers, that the client connects to these neighbors first, that the client has been unchoked by these neighbors first, etc. However, examination of the logs proves these guesses to be wrong.

Instead, the "closeness" seems to reflect a stable data-exchange relationship between the client and the peers. Table 4 shows the total data uploaded to these close peers as a percentage of the total data that the client uploads to other peers. The percentage is very close to 90%.



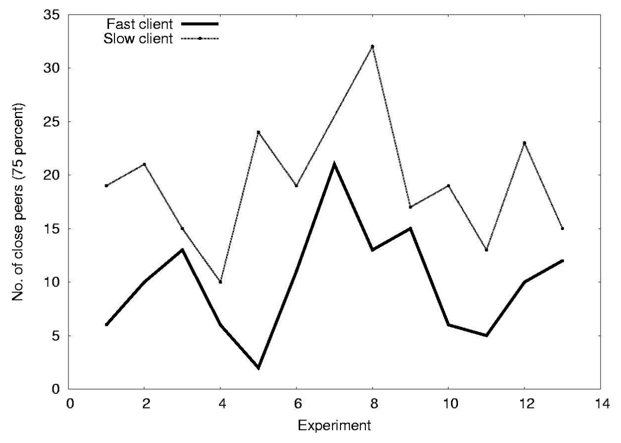


Figure 1: Minimum number of peers that give 50 % (left) and 75 % (right) of data, both for faster and slower clients.

Figure 2 shows the relationship graphically. For each connection between the client and a close peer, the figure shows three intervals: the duration of the connection, the duration of the client downloading data from the peer, and the duration of the client uploading data to the peer. The figure clearly shows a high overlap of the three intervals. Due to space constraints, we present the graph only for client 1 of experiment 5, but it is similar for all other experiments and clients. The graphs for all other experiments can be found at [2].

We believe that the "tit-for-tat" mechanism in BitTorrent is responsible for establishing a close relationship between close peers and the client. This relationship can be initiated both ways; either the remote peer starts uploading to the client and then the client reciprocates by unchoking and uploading to it, or vice-versa. When selecting a peer to unchoke, the BitTorrent client keeps the effective rate of download, i.e. amount of data received from a peer divided by the observation interval, of all peers, and selects the one with the highest effective download rate. Since the observation interval is usually more than 20 seconds, the effective download rate favors peers which have been giving data consistently, even if the download was at a low rate. So, once a peer becomes close, it stays close for a long time.

Since each peer can upload to 7 neighbors concurrently, BitTorrent effectively runs a randomized, completely distributed algorithm to obtain a perfect matching between peers, i.e. an undirected random graph where each node has degree of 7. In other words, each peer can have 7 close neighbors at a time, and peer A only becomes close to peer B if both have openings for a close neighbor.

4.3.2 *Cost of Peer Leaving*

It should not be a surprise that once an established matching is disturbed, it takes a long time for the algorithm to re-establish the matching. For a client, the departure of a close peer incurs a significant penalty in terms of increased download time. The client establishes a close relationship with a peer by keeping it unchoked and uploading data to it.

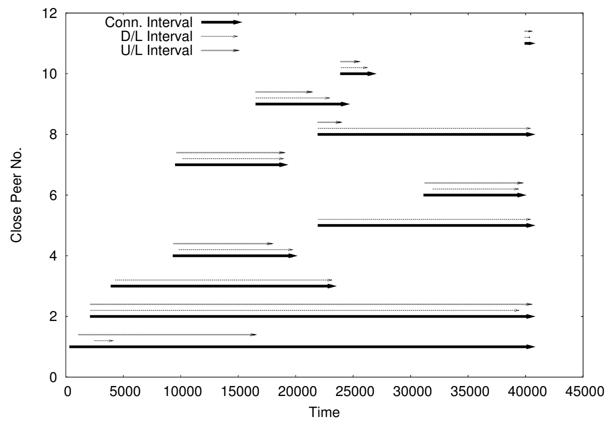


Figure 2: Time intervals in which the client is connected to, downloading from and uploading to close peers for experiment 5, client 1.

So if a close peer leaves the network, the client has to look for another peer to fill the void created. Since the number of concurrent uploads is limited and not every peer would reciprocate by giving data back, finding another close peer takes up a lot of time.

Table 5 shows, for each experiment, the average time it takes the client to find another close peer after a close peer leaves. To calculate these times, the analysis tool views each close peer as occupying one of 7 or 8 concurrent upload slots. When one close peer leaves and a new one occupies the same slot, the difference in time between the two events is the time wasted for making this *switch*. Averaging the wasted time over all such switches gives the average wasted time for the client.

The results explain why a higher number of close peers indicates a slower peer. A higher number of close peers means that the client has to pay the switching overhead more often, thus increasing the download time.

Exp.	Client	Avg. Time wasted (secs.)
1	1	2008.20
	2	2624.33
2	1	2120.73
	2	2828.04
3	1	3002.38
	2	5362.95
4	1	2981.33
	2	1509.00
5	1	2611.00
	2	3223.81
6	1	1921.73
	2	1781.20
7	1	1617.35
	2	2479.07
8	1	1004.14
	2	3307.49
9	1	2831.19
	2	3449.86
10	1	1916.60
	2	2805.72
11	1	1621.67
	2	2497.58
12	1	2535.85
	2	2866.87
13	1	5464.06
	2	5487.29

Table 5: Average time wasted between losing a close peer and finding a new one.

4.4 Reasons of Close Neighbors Leaving

Why do close peers leave a client? We observe that once a peer becomes close, it continues uploading data to the client until one of the following situations occur:

1. the peer disconnects from the network;
2. the peer decides to choke our client;
3. the client becomes uninterested in the peer;

Peers that do not encounter any of the above situations keep uploading till the client finish downloading the entire file.

Table 6 shows the number of peers in each of the above categories for each experiment. It shows that a slower client has more peers falling in categories 1 and 2 than the faster client. The number of peers in category 3 is very small across all experiments.

What cause peers in category 1) and 2) to stop sending data to the client? We further categorized these peers' actions as follows:

1. Case A: the peer disconnects from the network while it is still receiving data from the client. The fact that it is downloading from the client when it gets disconnected strongly suggests that the disconnect has happened randomly, e.g. a mobile host.
2. Case B: the peer chokes the client, but the client does not choke it back and keeps sending data to the peer. This represents a potentially buggy behavior in the standard BitTorrent client.

Exp.	Client	Case 1	Case 2	Case 3
1	1	6	5	3
	2	14	11	0
2	1	8	9	2
	2	18	14	2
3	1	10	11	0
	2	10	15	1
4	1	3	5	2
	2	7	9	1
5	1	3	4	1
	2	16	25	1
6	1	8	11	1
	2	16	19	1
7	1	17	9	1
	2	16	23	3
8	1	14	6	1
	2	28	20	4
9	1	8	16	2
	2	7	10	3
10	1	1	6	1
	2	8	14	1
11	1	5	2	2
	2	4	14	0
12	1	10	10	2
	2	15	27	0
13	1	12	10	0
	2	22	9	0
Case 1 : # of peers that disconnected				
Case 2 : # of peers who choked				
Case 3 : # of peers who got uninteresting				

Table 6: Distribution showing why download from a peer stops

3. Case C: the peer chokes the client, followed by either the client choking it in return or the connection being severed.
4. Case D: the peer is a seed and therefore the client never uploads to it. After uploading data to the client for some time, either the peer chokes the client or just disconnects.
5. Case E: the client chokes this peer first, and either this peer chokes the client back or the connection is severed. This usually happens when our client decides that it has found a better peer.
6. Case F: the peer became uninterested in our client. Most likely, this is because it has finished its download.

Table 7 shows the results of classifying the peers in categories 1 and 2 of table 6 using the above method. Case A and Case F are situations not affected by any BitTorrent mechanism, and not under the influence of our BitTorrent client. Table 7 shows that across all experiments, the slower client has more case A and case F peers than the faster client.

Case B is rare among the experiments, indicating that the BitTorrent client tends to implement tit-for-tat correctly. Case D has a higher count in the slower client than in the faster client. Intuitively, seeds should have helped our client. However, further examinations of the log find that these seeds are responsible for a small fraction of the data. In other words, they rank quite low in the list of close peers. Thus, it is possible that the slower client finds more of them because its download takes longer.

Case C and E are results of the choking/unchoking interactions between the client and its peers. Case E is initiated by the client: it believes that it has found something better. We examine the connection interval graphs (e.g. Figure 2) of all experiments, and find that in many cases, the time when the download from one peer ends is quite close to the time when the connection from another close peer begins, indicating that indeed, the client has switched to a faster peer. However, there are also cases where the new “close” peer only sticks around for a very short time, indicating that perhaps it has not been a good idea for the client to switch to the new peer. Case C is initiated by a close peer, indicating that the peer has found a better client. Case C is not under the direct control of the client, and its impact on the clients appear to be minor.

Summarizing the results, we conclude that the reasons for higher turn-over in close peers for the slower client are mainly events outside its control, including peers that are mobile hosts and disconnect randomly, peers that finish its download and exit, or peers that simply do not stay in the P2P network for long. In the next section, techniques that could potentially help the unlucky client are explored.

5. TECHNIQUES TO IMPROVE QOS

Our investigation naturally leads to the following ideas to improve the predictability of download times of P2P file distribution. Some of the ideas are more suited for content providers that can spend some resources to facilitate the download process, and use client software that is not easy to modify. Others are more suited for pure peer-to-peer systems.

- Dedicated “helper” seeds that aid peers during their transition from one close peer to another. Assuming that the client software is trustworthy, the client can notify the tracker that it has just lost a close peer and is in the process of trying to find another one. The tracker can then assign one of the helper seeds to send data to the client, so that its download is not slowed down. The help ends when the client finds another close peer.

- Algorithm techniques to speed up the convergence of the tit-for-tat matching algorithm. For example, the BitTorrent protocol can be changed so that peers notify each other of their respective close neighbors. The intention is that when a peer A leaves the network, those who are close to A can try A’s neighbors first in order to find new close peers, since A’s neighbors are most likely to have an upload slot open.

- Refinements of choking/unchoking algorithm to avoid unnecessary turn-overs in close peers. This would reduce mistakes in Case E as discussed in Section 4.4.

Unfortunately, detailed explorations of these techniques are out of the scope of this paper and remain our future work.

Exp.	Client	A	B	C	D	E	F
1	1	1	0	0	1	3	6
	2	3	1	2	4	4	11
2	1	4	0	4	0	4	5
	2	6	0	3	3	12	8
3	1	5	0	1	2	4	9
	2	2	0	5	3	6	9
4	1	0	0	2	1	1	4
	2	2	0	2	3	1	8
5	1	1	1	1	1	0	3
	2	6	0	8	6	6	15
6	1	3	0	5	0	6	5
	2	6	1	6	8	4	10
7	1	3	0	4	3	10	6
	2	9	0	5	4	9	12
8	1	5	0	2	2	6	5
	2	8	0	5	5	18	12
9	1	5	0	8	1	4	6
	2	0	1	4	0	5	7
10	1	0	1	0	3	0	3
	2	4	1	1	4	3	9
11	1	0	0	1	1	2	3
	2	1	1	0	4	3	9
12	1	3	0	2	2	5	8
	2	7	0	4	13	4	14
13	1	7	1	1	2	5	6
	2	8	0	0	2	10	11

A : # that disconnected randomly
B : # that choked first, were never choked back
C : # that choked first, then got choked or disconnected
D : # of seeds
E : # that were choked first, then choked or disconnected
F : # that became uninterested

Table 7: Details of clients’ uploads behavior

6. RELATED WORK

Because of its popularity, in recent years there have been many studies on BitTorrent. At its heart, BitTorrent attempts to solve the “broadcasting problem”, i.e. disseminating M messages in a population of N nodes in the shortest time. In the setting of the Internet where nodes can communicate in both directions simultaneously and have the same bandwidth, the lower bound on download time is $M + log_2(N)$ units, where a unit is the time it takes for two nodes to exchange a message. Assuming that all nodes are completely connected, an optimal algorithm that relies on a centralized scheduler is described in [12]. The algorithm goes in rounds, and establishes matching pairs among nodes in a deterministic fashion. Similar algorithms are also discussed in [13, 8]. We note that the matchings used in these optimal algorithms are quite similar to the matching performed by BitTorrent.

Simulation studies have found that in typical settings of cable modem and DSL nodes the links are almost fully utilized all the time [1, 8]. The results indicate that the random algorithms used by BitTorrent lead to nearly optimal performance. However, these studies do not look into the impact of peers leaving, and do not investigate the variations in download performance.

There are numerous measurement studies of BitTorrent traffic on the Internet [7, 11, 10]. In [7], detailed traces gathered over a period of 8 months (Jun ’03 to Mar ’04) are analyzed to study the availability and performance of BitTorrent networks. Their findings of average download speed of 30KB/s are quite close to the results in this paper. In [11], the BitTorrent network for the Linux RedHat 9 distribution was studied via both a 5-month tracker log and detailed logs of a single peer's download. The study found that the average download rate was around 500kb/s, possibly due to the presence of many dedicated high-speed seeds for the file. The study also noted that there was a high variation in the download throughput experienced by the peers. However, the study did not look into the causes of the variation. In [10], interactions between multiple BitTorrent networks were studied, and the authors proposed that BitTorrent networks cooperate to improve performance. The study noted that the download performance fluctuated widely with the peer population, but did not compare performance of peers downloading files during the exact same time periods. These studies do not focus on investigating the detailed causes of variations in download times.

Finally, studies have investigated using network coding to improve BitTorrent [9]. The use of network coding solves the "last missing block" problem and significantly improves content availability in the network. The use of network coding is complementary to the issues we investigate here.

7. CONCLUSIONS AND FUTURE WORK

This paper tries to answer the following question: if two BitTorrent clients are downloading the same file at the same time and their download times differ significantly, what is causing one client to be slower than the other? Answers to this question not only suggest ways to improve the performance of BitTorrent, but also highlight factors that determine the download performance and point out ways to offer QoS guarantees on the download time.

Analysis of 13 side-by-side experiments demonstrates that a dominant factor in a client's performance is the behavior of its close neighbors, including how often they leave the network and how often they choke the client. This factor is more important than the network bandwidth of the set of peers that the client is given, a result that is somewhat surprising. Part of the reason lies in the fact that the current tit-for-tat mechanism appears to converge slowly; it takes a long time for a client to find a new close peer if an old one leaves.

There are a number of limitations to our study. Our measurement of a peer's network bandwidth is not precise. More comprehensive studies could utilize the various information related to an IP address, such as the AS and geo-location, to obtain better estimates. We have also not studied QoS issues related to the use of TCP protocol in BitTorrent, particular the fairness among parallel connections and its impact on choking/unchoking. Finally, we have not examined in detail the impact of the size of the P2P network on download performance. Investigation of these issues remains our future work.

Our immediate next step is to explore, via simulation and implementation, techniques that reduce the variance in BitTorrent peers' performance, including ideas described in Section 5. We plan to validate the techniques with real-life experiments such as hosting trackers for legitimate popular content, and would welcome any sources of such content for this study.

8. REFERENCES

[1] A. Bharambe, C. Herley, and V. N. Padmanabhan. Understanding and deconstructing bittorrent performance. In *Proceedings of SIGMETRICS*, 2005. *http://research.microsoft.com/p̃admanab/papers/msr-tr-2005-03.pdf.*

[2] R. Bindal and P. Cao. Bittorrent performance study: Connection interval graphs, 2006. *http://crypto.stanford.edu/c̃ao/bt-study/conn_intervals/.*

[3] B. Cohen. *http://download.bittorrent.com/.*

[4] B. Cohen. Incentives build robustness in bittorrent, 2003. *http://citeseer.nj.nec.com/cohen03incentives.html.*

[5] B. B. Corporation. Warner bros to sell movies on net, 2006. *http://news.bbc.co.uk/1/hi/entertainment/4665438.stm.*

[6] EContentMag.com. Chasing the user: The revenue streams of 2006, 2006. *http://www.econtentmag.com/Articles/ArticleReader.aspx ?ArticleID=14532&ContextSubtypeID=8.*

[7] P. G. Epema. The bittorrent p2p file-sharing system: Measurements and analysis. *http://citeseer.ist.psu.edu/725723.html.*

[8] P. Ganesan and M. Seshadri. On cooperative content distribution and the price of barter. In *Proceedings of ICDCS*, 2005. *http://dbpubs.stanford.edu:8090/pub/2005-4.*

[9] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proceedings of IEEE INFOCOM*, 2005. *http://research.microsoft.com/p̃ablo/papers/-nc_contentdist.pdf.*

[10] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measuremsnts, analysis and modeling of bittorrent-like systems. In *Proceedings of the Internet Measurement Conference*, 2005. *http://www.imconf.net/imc-2005/papers/imc05efiles/guo/guo.pdf.*

[11] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *Proceedings of the 5th Passive and Active Measurement Workshop*, 2004. *http://citeseer.ist.psu.edu/izal04dissecting.html.*

[12] J. Mundinger and R. Weber. Efficient file dissemination using peer-to-peer technology. *http://www.statslab.cam.ac.uk/Reports/2004/2004-01.pdf.*

[13] X. Yang and G. de Veciana. Service capacity of peer to peer networks. In *Proceedings of IEEE INFOCOM*, 2004. *http://citeseer.ist.psu.edu/yang04service.html.*