Poor Man's Broadband: Peer-to-Peer Dialup Networking

Umar Saif
LUMS, Computer Science Department, Pakistan
umar@lums.edu.pk

Ahsan Latif Chudhary
LUMS, Computer Science Department, Pakistan
alatif@lums.edu.pk

Shakeel Butt
LUMS, Computer Science Department, Pakistan
shakeel@lums.edu.pk

Nabeel Farooq Butt
LUMS, Computer Science Department, Pakistan
nabeelbutt@lums.edu.pk

ABSTRACT
In this paper we present a peer-to-peer dialup architecture for accelerated "Internet access" in the developing world. Our proposed architecture provides a mechanism for multiplexing the scarce and expensive international Internet bandwidth over higher bandwidth p2p dialup connections within a developing country. Our system combines a number of architectural components, such as incentive-driven p2p data transfer, intelligent connection interleaving and content-prefetching. This paper presents a detailed design, implementation and evaluation of our dialup p2p data transfer architecture inspired by Bittorrent.

Categories and Subject Descriptors
C.2.1 [**Communication Networks**]: Network Architecture and Design; C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability

General Terms
Design, Experimentation, Performance.

Keywords
Developing World ICT, Bandwidth, Peer-to-Peer, Dialup, Bittorrent

1. INTRODUCTION
The "digital divide" between the developed and developing world is underscored by a stark discrepancy in international Internet bandwidth. Developing countries often have to pay the full cost of a link to a hub in a developed country, making the cost of broadband Internet connections prohibitively expensive for individual users. For instance, more than 40 countries have less than 10Mbps of international Internet bandwidth, whereas in Belgium, a 9Mbps ADSL high-speed Internet package is available for just USD 80 a month [1]. The cost of a 2 Mbps link in Pakistan is close to USD 4000/month, compared to a 2 Mbps ADSL link in UK for just 60 USD/month. As a result, Internet access in the developing world is primarily over dialup connections, often using pre-paid "scratch cards".

Characterized like this, the scarcity of available bandwidth in the developing world is not a "last-mile" problem. A 56Kbps modem can typically achieve an average throughput of close to 40Kbps, more than twice the bandwidth available over a typical (10-20Kbps) dialup Internet connection in Pakistan. Given the high cost of international Internet circuits, local ISP idiosyncrasies, and the economics and politics of routing on the Internet, end-users in the developing world are forced to access the Internet at a fraction of the speed possible over a dialup connection.

Unfortunately, the paucity of Internet bandwidth severely limits the utility of Internet in the developing world. In general, Internet is almost never used for accessing or transferring data larger than a few hundred kilobytes. This includes software downloads and online software updates, large email attachments and sharing and download of large files. On the other hand, such bulk data transfer comprises 70% of the overall Internet traffic [2].

To illustrate the limited role of Internet in the developing world, consider a simple scenario where two users in Pakistan wish to exchange a 3.5MB PDF file as an email attachment. On a slow dialup connection of, say 16kb/sec, it would take close to an hour to transfer the file (30 minutes each for uploading and downloading the email attachment). Accounting for possible disconnections over such long dialup connections, the effective time for file transfer may be prohibitively long for most people to even consider the Internet as a medium for exchanging such large content.

Ironically, without the Internet, if the users established a direct peer-to-peer dialup connection between the two computers, they can exchange the file in less than 18 minutes (over a v.90 peer-to-peer dialup connection at 32kb/sec) – a performance improvement of more than a factor of 3 -- at a bandwidth equivalent to a typical broadband connection in Pakistan (~35Kb/sec).

However, the contemporary view of a dialup is limited to a "last-mile" connection between the end-host and the Internet Service Provider (ISP) to access the Internet. In this paper, we propose a departure from this view of dialup to address the bandwidth limitations in the developing world. The key idea of our proposal is simple: instead of using dialup solely as a mechanism to connect to the Internet, at a fraction of bandwidth afforded by the modem, we propose to use point-to-point dialup connections, at modem-speed, to bypass the Internet when transferring large content between end-hosts. This is achieved by using a network routing layer, dubbed dialup-underlay, which monitors end-host bandwidth and interleaves the (low-bandwidth) ISP dialup connection with modem-speed peer-to-peer dialup connections to download large content on the Internet. Figure 1 sketches the proposed ISP-p2p interleaving architecture.

Perhaps radical, and closer to pre-internet technologies (such as FidoNet) than current-day broadband networking technologies, we believe that such an approach has practical appeal for the "other four billion" users of the Internet. Furthermore, while this paper is focused on using POTS to-modem dialup connections to circumvent extremely low bandwidth Internet connections in the developing world, the crux of our proposal has generic appeal in the developing world. Countries like Pakistan have seen a phenomenal growth in Telecom, resulting in cheaper and better communication infrastructure within the country -- including deployment of high-bandwidth Wireless Local Loop (WLL) and

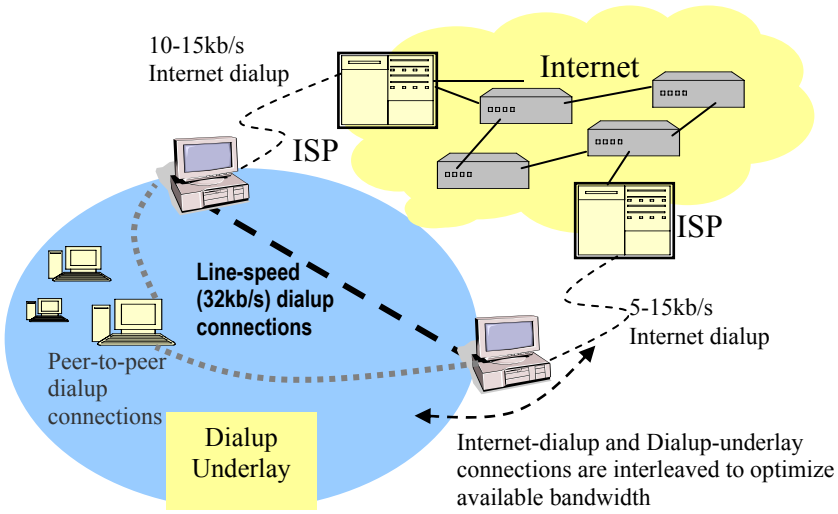

Figure 1: Our system interleaves low-bandwidth Internet connections with modem-speed p2p dialup connections

WiMAX -- while Internet connectivity and bandwidth remains scarce. It is worth reiterating that the extremely low Internet bandwidth received by end-users in countries like Pakistan is not a "last-mile problem". In our experiments, we routinely achieved a peer-to-peer dialup bandwidth of 32kb/sec (symmetric), but the bandwidth allocated by various dialup ISPs rarely exceeded 15kb/sec (typically between 8-15kb/sec). Our experiments, conducted with ISPs in the second largest metropolitan in Pakistan, repeatedly highlighted the fact that end-user bandwidth lower than 2KB/sec is typically not due to poor phone lines, rather it is because of the overloaded upstream Internet connection of the ISP.

Our proposed architecture, at an abstract level, provides a mechanism for multiplexing the scarce and expensive international Internet bandwidth over higher bandwidth peer-to-peer connections within a developing country.

The point-to-point nature of our architecture is reminiscent of routing systems from the pre-Internet days, such as FidoNet, USENET (historically dubbed poor man's ARPANET) and UUCP. However, our motivation, design goals and implementation strategy is very different from these systems. Unlike pre-Internet systems that relied solely on dialup connections for moving content between computers, our goal is to accelerate access to large content on Internet by utilizing a dialup connection at the maximum bandwidth supported by the modem. In our model, content still "resides" on the Internet, but may be downloaded using a p2p dialup connection to reduce download time.

Key to the practicality of our approach is a realization that it is not always possible or desirable to make a direct dialup connection between the "client" and "server". For instance, a Web-server may not be accessible over dialup; the server may not support a dial-in facility or may be located in a different country, requiring an expensive international phone call to establish a dialup connection. Even when the server is accessible over dialup, it would have a limited number of modems and a finite capacity for handling dial-in connections.

Our system derives its practicality from combining the p2p dialup-underlay with a data transfer architecture that enables dialup clients to cache and collaboratively share downloaded content – a peer-to-peer (p2p) file-sharing systems [3] (albeit with additional mechanism for direct p2p dialup connections). Our p2p data transfer architecture is derived from and compatible with the hugely popular Bittorrent peer-to-peer file-sharing system [3]. We chose Bittorrent over other p2p file-sharing systems like Gnutella, Kazaa, eDonkey etc. due to its download performance and robust incentive mechanism [12, 13, 14]. We call our system DitTorrent (Dialup Bittorrent).

DitTorrent is designed to address two limitations of dialup Internet access in the developing world: Extremely low Internet bandwidth (5-10kb/sec) and frequent disconnections from the Internet. DitTorrent circumvents low-bandwidth dialup Internet connections by using modem-speed p2p dialup connections, while a suspend-resume download manager mitigates disruption caused by dialup disconnections.

In addition to offering a Bittorrent-like interface for file download, DitTorrent is also used to implement a WWW download-accelerator. The download-accelerator is implemented as a browser plug-in and employs the p2p data transfer layer, when possible, to download requested content over DitTorrent (rather than using a TCP/IP connection over the Internet between the client and the Web-server). Moreover, since download via DitTorrent may involve "offline" data exchange (using p2p dialup connections), the WWW accelerator plugin uses a local pre-fetching proxy to mitigate the disruption of online access when interleaving ISP and p2p dialup connections. Finally, the suspend-resume download-manager provides a common caching and scheduling mechanism to maintain download sessions across interleaving of ISP (Internet) and peer-to-peer dialup connections. Figure 2 sketches the high-level architecture of our system.

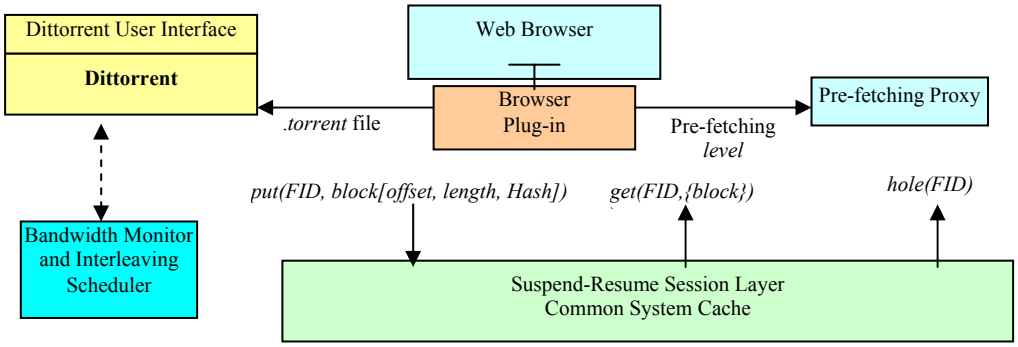

Figure 2: System Architectural Components

In this paper, we describe the design and implementation of our system and present detailed analysis of data transfer over the p2p dialup-underlay. The rest of the paper is organized as follows. Section 2 gives an overview of the system architecture. Section 3 describes the operation of our extension to Bittorrent namely DitTorrent. Section 4 outlines our implementation and section 5 presents related and future work.

2. SYSTEM OVERVIEW

Our system is based on a modular architecture, sketched in Figure 2. The DitTorrent module presents an interface similar to Bittorrent and is designed to be compatible with existing Bittorrent clients and trackers. Additionally, DitTorrent can establish direct point-to-point dialup connections to download content at modem-speed when such a connection can reduce the overall download time of a requested file. The DitTorrent client also interfaces with a Web browser plug-in to enable download of requested data over DitTorrent; depending on the size of the requested content and its availability on the p2p network, the Web browser plug-in can invoke DitTorrent instead of using the slow-speed Internet connection to download content.

Below we describe these components in detail.

2.1 Data-oriented Transfer

The internal interfacing between the Web browser and the p2p client embodies a data-oriented architecture; a user chooses a file he wishes to download, while the system automatically explores alternative mechanisms to download the file in minimum possible time [2]. To achieve this, our browser plugin recognizes .torrent files and automatically triggers the DitTorrent client when a user clicks on a file accessible on the DitTorrent network. The internal interfacing between p2p file transfer and Web download enables a model where different chunks of the same file may be downloaded by different download mechanisms, either sequentially or in parallel, to reduce the overall download time.

To enable multiple downloads, the common system cache provides a simple interface, permitting different download mechanisms, such as Bittorrent and Web browser, to create a file handle (encoded as a unique File ID, *FID*) and put in different blocks of a file that are pieced together to generate the complete file. The *FID* is a (MD5) hash of the file contents, permitting both unique identification of file contents for subsequent lookups and verification of the completeness and integrity of the downloaded contents. Moreover, the common system cache is implemented as a suspend-resume session layer, which enables online (HTTP and Bittorrent) and offline mechanisms (DitTorrent offline mode) to download chunks of a file over interleaving ISP and p2p connections. Different download mechanisms, such as p2p Bittorrent and client-server HTTP connections, invoke the *hole* interface of the session layer to *find* chunks of a file currently not downloaded; the hole method, when invoked with an FID, returns a pair of offsets that indicate a range of bytes in the file, or a *hole*, currently not stored in the cache. Successive invocations of the hole interface return non-overlapping file chunks not stored in the cache; the browser plugin and DitTorrent calling the hole interface are assigned non-overlapping chunks of a file to download either in parallel (online mode) or sequentially (when interleaving online and offline modes). In our current implementation, we have fixed the hole size to be the same as a Bittorrent block (128kB), simplifying the interoperation of the session layer and our p2p DitTorrent client.

2.2 Interleaving Scheduler

The switchover from the Internet connection (via ISP) to a (offline) dialup p2p is orchestrated by the interleaving-scheduler module, with the aim to reduce the overall download time of a requested file. It is worth highlighting that the bandwidth improvement resulting from a transition from a slow-speed Internet connection to a 32kb/sec p2p modem-to-modem connection is not for free. In fact, each such transition incurs an overhead of close to 30 seconds for negotiating the new modem connection. Of course, each subsequent transition between two p2p dialup connections, as well as the transition back to the Internet connection, also incurs the same overhead of modem-handshake. The interleaving-scheduler module, therefore, must take into account three factors before interleaving ISP-p2p connections: current Internet bandwidth, size of the requested file and the dialup handshake latency. Figure 3 illustrates suitable transition points with varying Internet bandwidth and the number of dialup connections necessary to download a file. For smaller files, the overhead of modem-to-modem negotiation outweighs the time-saved in downloading the file at modem speed. With a moderately low-bandwidth dialup Internet connection of 15kb/sec, and between 2-4 average modem connections (disconnection and reconnection to the Internet, as well as dialup connections between peers), interleaving of ISP-p2p typically becomes feasible for files larger than 50kB.

2.3 Pre-fetching Proxy

Another component in the architectural diagram shown in figure 2 is a pre-fetching proxy used to mitigate the disruption to normal WWW access when the interleaving-scheduler disconnects from the ISP to establish a p2p dialup connection. The pre-fetching

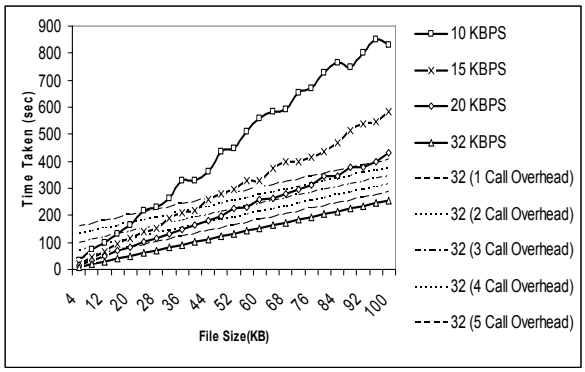

Figure 3: Feasible interleaving points between p2p and Internet

proxy is invoked by the browser plug-in when notified by the interleaving-scheduler module that a requested file may be feasibly downloaded via a p2p dialup connection. The pre-fetching proxy is invoked with a number specifying the level of web-pages to be recursively pre-fetched by the proxy starting with the egress links of the current webpage. The pre-fetching proxy notifies the browser plugin when it has downloaded the requisite webpages, triggering a switchover to the (offline) p2p dialup connection. Subsequent accesses to webpages are served by the pre-fetching proxy, while access to a webpage whose egress links are not prefetched triggers a reconnection to the Internet.

2.4 DitTorrent Peer-to-Peer Client

The centerpiece of our architecture is the DitTorrent p2p client. As mentioned earlier, our architecture derives its practicality by enabling clients (peers) to share downloaded data, minimizing the need for direct dialup connections between clients and servers. Our p2p data transfer architecture is derived from and compatible with the hugely popular Bittorrent peer-to-peer file-sharing system [3]. The key attraction of Bittorrent for us is its practical incentive-driven data-sharing model; instead of assuming a volunteer-driven model like FidoNet, where users are expected to voluntarily call one another to copy data between various nodes, our system is based on a more practical incentive-driven tit-for-tat data-sharing model of Bittorrent [3]. In our model, files are divided into smaller chunks, as in Bittorrent, that are virally replicated in the network based on opportunistic peer connections; node A may let node B download a chunk of a file as long as node B can offer another chunk in return that node A wishes to download.

However, this model also raises a number of interesting challenges. For one, download of a file in this model does not involve a single switchover from the online Internet world to a point-to-point dialup connection, but several shorter connections with different peers in the same vein as Bittorrent. Therefore, with the 30 seconds penalty of a modem-to-modem handshake, it is important to minimize the number of connections needed to download the entire contents of the file. Moreover, given the point-to-point nature of the operation over the dialup-underlay, clients must somehow discover other offline clients and establish peer-to-peer connections to exchange chunks of a file. The point-to-point nature of the dialup-underlay also introduces another interesting idiosyncrasy: if two peers are connected to each other, no other peer can connect to them. Hence, once a point-to-point connection is established, there is no mechanism for a client to opportunistically discover a better peer, till a time it hangs-up and connects to another client. Furthermore, Bittorrent's rate-based tit-for-tat data sharing model -- in which a host uploads data to a peer for only as long as it can download data from that peer at a similar data rate -- becomes superfluous in a point-to-point dialup connection. This is because upload and download rates in a standard p2p dialup connection are symmetric, e.g. 32kb/s upload and download for typical v.90 modems. Section 3 discusses the challenges and possible solutions to these challenges in detail.

3. PEER-TO-PEER DATA TRANSFER

In this section, we describe the design, implementation and evaluation of DitTorrent.

3.1 Evaluation Methodology

In order to understand the behavior of DitTorrent, especially under extreme conditions in the developing world, we use a simulation-based approach. Even though we have released our DitTorrent implementation (Client and Tracker) to a community of users in Pakistan, via sourceforge [15], we chose a simulation-based evaluation since very little data exists in terms of traces of real torrents for extremely low-bandwidth connections. Moreover, our simulator's controlled environment made it possible to evaluate aspects of our design that are otherwise difficult to infer from tracker logs or by running moderately-sized experiments.

For our evaluation, we have implemented a discrete-event simulator of DitTorrent, which extends the Bittorrent simulator implemented by Bharambe et al. [11]. However, while the underlying framework of our simulator is derived from the simulator described in [4], our model of DitTorrent is almost diametrically different from the Bittorrent model implemented by [4]. For instance, the bittorrent simulator by Bharambe et al. is designed to simulate parallel downloads by a Bittorrent client, while DitTorrent is limited to point-to-point symmetric upload/download between only two hosts. Likewise, Bharambe's simulator [4] assumes perfect knowledge of the location of each block of a file, while a DitTorrent client's knowledge about the location of file blocks is often imperfect. Similarly, a Bittorrent client is designed to maximize download bandwidth, while a DitTorrent client attempts to minimize the time wasted in negotiating new modem-to-modem connections.

To capture the idiosyncrasies of DitTorrent's (offline) operation, we implemented the following new features in the simulator described in [4]: Point-to-point symmetric connections, call-overhead resulting from modem-to-modem negotiation, busy-tones overhead during "flash crowds", offline block-discovery and greedy peer selection with different "end-game" modes. Below we describe the motivation, implementation and evaluation of each of these features in detail.

In our experimental setup, we configured our simulator to use a swarm of 100 nodes (typical Bittorrent swarm size). Our simulations were run on a P4 machine 3.2GHz, with 1GB RAM. Our simulation environment was configured with the following parameters:

- In our simulation environment, all the participating nodes were configured with symmetric download and upload bandwidth, set at 30 Kbps.

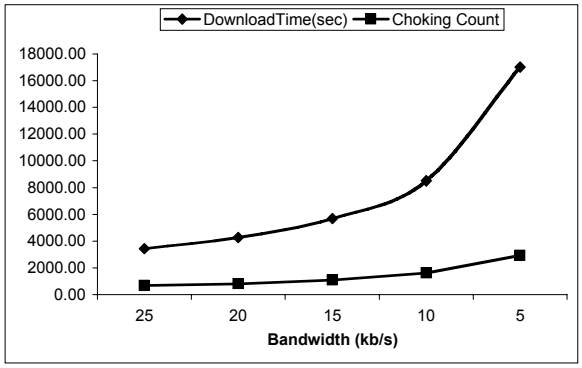

Figure 4: Bittorrent performance deteriotes sharply as the client bandwidth drops to 10kb/sec

- Peers in our simulation were bootstrapped with a single block, unless otherwise stated in the experiments described below.
- In our experiments, we measured the performance of the system by varying the size of the file to be downloaded, and where specifically mentioned, the number of initial blocks allocated to each peer.
- Given the point-to-point nature of DitTorrent, each node was configured to only connect with one other at a time.
- We set the seed leaving probability to 1 in our experiments; we do not assume that a node stays in the simulation after completing its download.
- In the simulations for *offline block discovery* and *flash-crowds* (described below), nodes were introduced in an ongoing experiment, after a random delay, to simulate late-entering nodes.

3.2 Architectural Overview

DitTorrent is designed to be backwardly compatible with Bittorrent. Compatibility with Bittorrent has obvious appeal in terms of user adoption, making DitTorrent a vehicle for using Bittorrent in the developing-world. Above all, DitTorrent derives its incentive-driven tit-for-tat data-sharing model from Bittorrent. Furthermore, high-level architectural components of DitTorrent are derived from Bittorrent; files are published by advertising a .torrent meta-file, clients make peer-to-peer connections to opportunistically download, cache and publish blocks of files, and a tracker acts as a directory service for clients to discover peers from which blocks of a file may be downloaded. The use of a .torrent file and a tracker for initial peer discovery and bootstrap provides a basis for compatibility with Bittorrent; DitTorrent tracker, as well as the DitTorrent .torrent file format, is designed to be backwardly compatible with Bittorrent. As a result, existing Bittorrent clients can interoperate with DitTorrent clients. DitTorrent clients, however, of course have the additional capability to establish point-to-point dialup connections for accelerated download of content.

Before proceeding with the description of DitTorrent, it is instructive to consider the behavior of Bittorrent over characteristically low-bandwidth connections in the developing world. Figure 4 plots the download time of a 10 MB file over Bittorrent by a client connected to the Internet on a slow dialup connection. The results shown in figure 4 were reported by the original Bittorrent simulator of Bharambe et al [4], which most notably, ignores TCP timeouts. Still, the download time of a file goes up sharply as the bandwidth is reduced from 15kb/sec to 5kb/sec. This is because the number of chokes experienced by the client (snubs by peers as they find better partners), increase as the client bandwidth nears 10kb/sec. For comparison, in the case when there are 30% low-bandwidth hosts in the mix of nodes using Bittorrent, a client with a 10kb/sec takes close to 8.5 times more than a cable node with a bandwidth of 100kb/sec for upload and 250 kb/sec for download. For these low-bandwidth nodes, point-to-point symmetric dialup connections, at 32kb/sec, can offer a substantial performance improvement. For instance, compared to a 10kb/sec Bittorrent client that downloads a 10MB file in 2 hours and 21 minutes (derived from the experiments shown in figure 4), a client using a point-to-point 32kb/sec connection can download the same file in 41.6 minutes (assuming a single point-to-point connection) -- a performance improvement of close to 70%.

DitTorrent is designed dynamically switch between its offline and online modes, depending on available bandwidth reported by the Interleaving-scheduler. Since the online mode mimics the normal Bittorrent operation, the rest of the paper describes the changes we made to Bittorrent to implement the point-to-point (offline) mode.

3.3 DitTorrent Tracker-Client Interaction

While DitTorrent's online mode mimics Bittorrent, its point-to-point mode requires special support from the tracker. Importantly, unlike a traditional Bittorrent tracker that keeps track of currently online hosts, a DitTorrent tracker must keep track of both offline and online hosts interested in downloading a file. While fundamental to the duality of operation of a DitTorrent client, this extension requires only a minor modification to existing Bittorrent trackers. Current Bittorrent trackers require a client to refresh its registration periodically by sending announce messages after interval number of seconds. A DitTorrent tracker, instead of deleting the record of a client that fails to refresh its registration after Interval seconds, simply marks the client offline and retains its entry for future lookups from DitTorrent clients. However, since dialup nodes are prone to frequent disconnections, our DitTorrent tracker marks a client offline only after the client misses successive periodic announcements.

A DitTorrent tracker distinguishes between Bittorrent and DitTorrent clients such that the latter can be given additional information for offline operation. To this end, a DitTorrent tracker accepts an additional event attribute from a DitTorrent client; a dialup client using DitTorrent registers with DitTorrent tracker using an HTTP GET request, just like Bittorrent, but the event param in the URL is set to 'dialup' to indicate that the registering client is a DitTorrent client. A client announce with event param set to dialup may include two additional parameters to the GET request URL, as described below.

The key additional parameter in a DitTorrent client registration request is a phone number to reach the client in the offline p2p mode. While in the current implementation the phone number is registered with the tracker as plain text, permitting DitTorrent clients to make direct connections, it is possible to encrypt the number for an intermediary trusted node that establishes a dialup

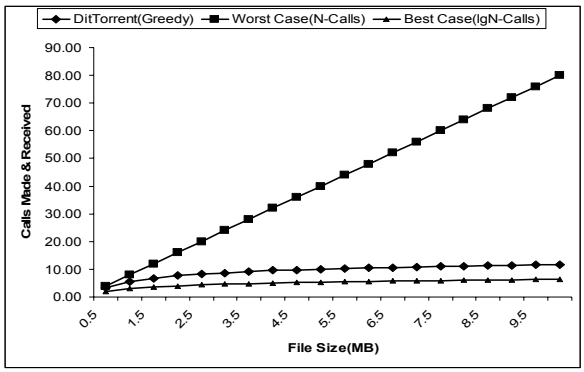

Figure 5: Comparison of the modem-to-modem negotiation overhead of the greedy peer selection with the best and worst cases

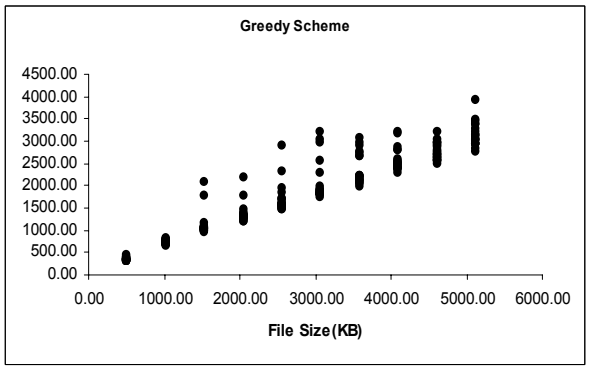

Figure 6: Variability in file download times due to the Last block problem in the simple greedy strategy

connection between DitTorrent clients. We are currently developing such rendezvous nodes, equipped with multiple modems, that accept two encrypted numbers and establish a spliced dialup connection between clients while protecting their identity (phone-numbers) from each other.

Optionally, a DitTorrent client can also include a list of time-windows that specify times of day during which dialup connections may be established with the client. A client can specify the following time-windows as part of its request to the tracker: available_time_window specifies the time interval in which the client is available for accepting phone calls, query_time_window specifies the time interval in which the client intends to make calls to download the file, and optionally, a previous_time_window in case this request updates existing registration of the client. For instance, a client installed on an office computer may advertise a time-window between 8 PM and 7 AM when phone lines for the office are generally free. Subsequent DitTorrent clients registering with the tracker are given contacts of those peers whose available_time_window overlaps with the query_time_window of the client (as well as online clients). The use of time-windows in our system is reminiscent of "zone mail hour" used by FidoNet [5] clients to specify a time-window for receiving dialup connections.

3.4 DitTorrent Peer-to-Peer Interaction

DitTorrent's offline point-to-point operation is fundamentally different from Bittorrent. Where a Bittorrent client attempts to minimize the download time of a file by opportunistically connecting with and disconnecting (choking and unchoking) from a large set of peers (peer swarm) in search for better download bandwidth, a DitTorrent client must minimize the number of peer connections when downloading a file. This is because "trying" new peers in Bittorrent is an almost zero-overhead operation (an unchoke message sent to the peer over a long-running TCP connection), while a new dialup connection in DitTorrent incurs a 30 second overhead in negotiating the dialup connection. To illustrate the point, consider the simulation results of Bittorrent shown in figure 4. The Bittorrent client shown in Figure 4, with a 10kb/sec symmetric upload/download bandwidth, makes close to 564 connections (with rate-based choking period set at 10 sec and opportunistic unchoking every 30 sec) during the download of a file of size 10 MB. If these were dialup connections, the time spent in negotiating new dialup connections would equal the actual time spent in downloading the contents of the file (an overhead of 100%).

Furthermore, given a symmetric upload/download bandwidth in a p2p dialup connection, DitTorrent's offline operation need not include Bittorrent's rate-based choking. Instead, a DitTorrent client (in offline mode) should only choke a peer when the peer can no longer upload newer blocks needed by the client; peer connections last for as long as peers can reciprocate each other with non-overlapping blocks of file, while there is no need for opportunistic unchokes. This approach of peer-choking, similar to pairwise block-level tit-for-tat (BLTFT) proposed in [4] (as opposed to rate-based tit-for-tat implemented by Bittorrent), is a natural fit in a point-to-point dialup setup and is implemented by DitTorrent.

However, while disabling opportunistic unchokes and using Block-level TFT avoids unnecessary connections in DitTorrent's point-to-point mode, the overhead of negotiating dialup connections must be carefully managed, especially for smaller files. This is because dialup connections, worth 30 seconds each, incur an overhead equivalent to exchanging a whole block (128kB) on Bittorrent at 32kb/sec.

3.4.1 Peer-to-Peer Dial-up overhead

In order to understand the overhead of modem-to-modem negotiation in DitTorrent, consider the theoretical best and worst case for downloading a file of size N blocks, with a client starting with a single block of the file.

The theoretical best case, aimed at minimizing the number of calls required to download a file, may be understood as peer selection policy that exactly matches peer needs; a client calls only that peer which has exactly the same number of *complementary* blocks. In this case, with BLTFT, the first dialup connection of a new client with a single initial block will result in the exchange of 1 block, making it 2 blocks at the client. The next call will result in 2 blocks exchanged, making it 4 blocks at the client. Likewise, the next call will result in 8 blocks at the client, then 16 blocks and so on. Therefore, for a file of size N blocks, it will take at least $\log_2 N$ calls to download all blocks of the file. This theoretical best case assumes a perfect match of peer needs on every call made by the client -- any mismatch would result in additional future calls for either the caller or the callee. However, while simple to implement, this scheme, which we dub *perfect-*

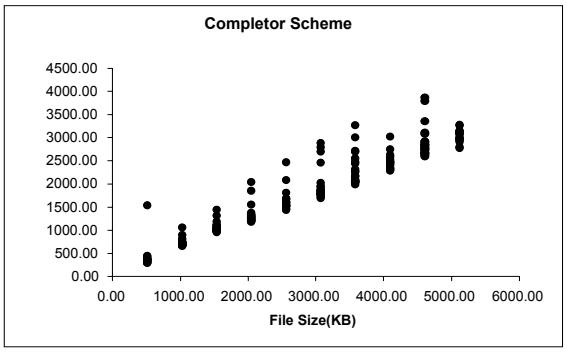

Figure 7: Variability in the file download times due to the last block problem in the greedy-completor strategy

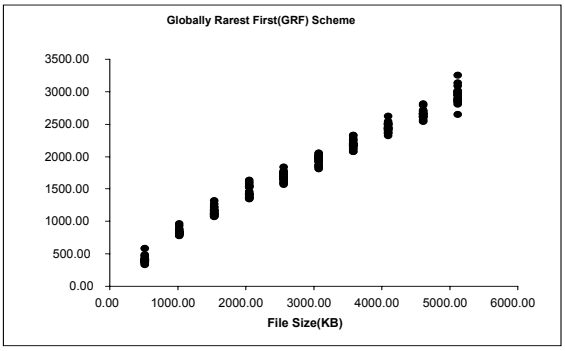

Figure 8: Variability in file download times due to the last block problem with Global-Rarest-First peer prioritization

match, is prone to starvation of nodes in practice. The reason for this is an implicit assumption in this scheme about a delicate balance in the number of peers vying to download the file and the total number of file blocks. For one, the total number of blocks in a file must be in powers of 2, which cannot be assumed in a p2p system that serves arbitrarily sized files. These blocks, moreover, must be equally divided among peers at bootstrap, implicitly requiring the total number of file blocks to be an even multiple of the total number of peers. Finally, this best-case scenario is only possible if pairs of nodes proceed in lockstep, a condition that is difficult to orchestrate in practice since different nodes may join the p2p network at different times.

The theoretical worst case may arise in a scenario in which a client must make N calls to download a file of N blocks, incurring a modem-to-modem negotiation overhead for each block of the file. Keeping with Bittorrent's bootstrap model of granting a single block to a caller even when it has no new blocks to reciprocate, this scenario is possible in practice when a solitary new node joins the p2p network in which existing nodes have close to 100% of the file. In this case, since the new node never acquires unique blocks with respect to existing nodes, each call only results in a single block given to the callee in the bootstrap mode of Bittorrent.

At an abstract level, the impracticality of the best case, perfect-match, approach stems from its rigidity; a peer calls another peer only when there is an exact match of not only the blocks stored at the peers but their mutual needs. This inflexibility can be addressed by using an equally simple *greedy strategy*. In the greedy strategy, a client simply calls that peer which can give it the maximum number of blocks. A greedy client grabs the maximum it can at any point in time, regardless of whether it is an exact or a suboptimal match on either side of perfect-match. Figure 5 compares the performance of this simple greedy-strategy with the best (Log_2N calls) and worst case (N Calls). The clients in the simulation are bootstrapped with an initial set of 1-5 random blocks for swarm of size 100. However, while a simple greedy strategy performs adequately well on average, we quickly realized the impact of the "last-block-problem" in p2p systems [4]. Figure 6 plots the download times from 20 different simulations as we varied the file size. The variability in the measured times for a given file size reflects the time for which different nodes in the swarm may be "stuck" trying to download the last few blocks of a file. Bittorrent employs two techniques to help nodes that are near completion to finish quickly: End-game mode, which enables a client close to finishing to quickly search for the last few missing blocks, and Local Rarest First (LRF) which helps balance the rarity of different blocks by requiring clients to download the rarest block first from a connected peer.

In Ditttorent, we experimented with analogues of both of these schemes. In the first implementation, we mimicked the effect of the end-game mode by modifying the greedy policy, dubbed *greedy-completor*. With this modification, a greedy client breaks ties by favoring those peers that will finish the file download at the end of the connection. Intuitively, this scheme is aimed at enhancing the chances of relative newcomers to expedite the completion of peers closer to finishing. Conversely, peers relatively early in the race get blocks from those near completion, causing rare blocks (typically stored at seeds or near-seeds) to be transferred from the old to the new. Figure 7, like figure 6, plots download times across 20 simulations, illustrating the reduction in the variability of measured times for *greedy-completor*.

Our second strategy to combat the last-block-problem is inspired by Bittorrent's LRF strategy. However, LRF's rationale is that a peer should grab the rarest block first from a peer in case it is choked prematurely. This has little impact in DitTorrent since a p2p connection in DitTorrent lasts as long as peers have something to exchange; there is no danger of unanticipated choking due to a dip in the client's upload bandwidth. Instead, we implement Global Rarest First (GRF) strategy in DitTorrent. Unlike LRF, in which a client grabs the rarest blocks stored at a peer, a client using GRF *chooses* a peer that has the rarest blocks stored at it. Viewed differently, LRF is a block prioritization strategy, while GRF is a peer selection policy; a client using GRF prioritizes its connections with peers according to the rarity of blocks stored at them. For instance, in our greedy strategy, a client that could exchange three blocks each with two of its peers, will call that peer first which can offer comparatively rarer blocks.

Figure 8 shows that our greedy peer selection policy with GRF performs better than *greedy-completor* in terms of the variability in download times across 20 simulations.

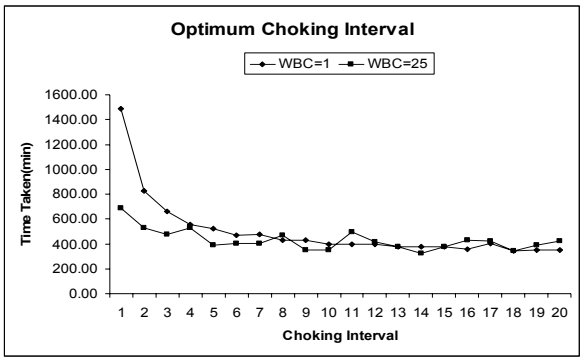

Figure 9: Analysis of overhead due to "busy-tones" in flash crowds

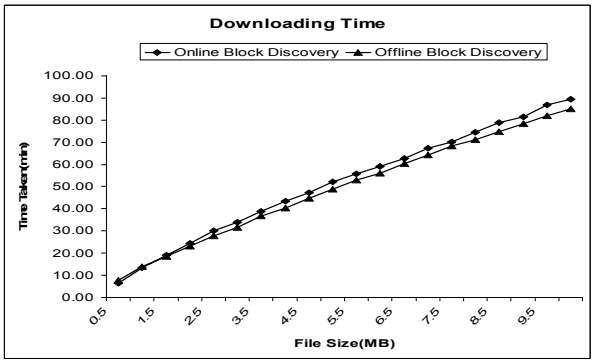

Figure 10: Comparison of DitTorrent's offline block-discovery with Bittorrent "perfect knowledge" about block locations

3.4.2 Flash Crowds in DitTorrent

As mentioned earlier, the point-to-point nature of DitTorrent's offline operation precludes multiple peer connections. In fact, a client calling a peer that is already connected to another peer would get a "busy-tone". Moreover, unfortunately, this operation of checking the availability of a peer incurs an extra overhead of close to 10 seconds (time to call the number and receive a busy tone). This overhead becomes particularly significant during the so-called flash crowds [4], in which many clients want to quickly download a newly accessible file. In a point-to-point setup, flash crowds result in a large fraction of the call attempts failing with busy-tones.

Intuitively, this may be addressed by introducing a wait-time between calls during times of congestion. Our implementation combats flash crowds by using two simple heuristics: 1) Each client must wait for n seconds before trying again if it finds all of its potential peers (peers with non-overlapping chunks) busy; 2) A client must wait for n seconds between successive calls. The former introduces a backoff period in the times of congestion, while the latter is aimed at giving clients a chance to receive calls in between making calls.

In our simulation experiments, we found that the backoff time was more useful than the introduction of wait-time between calls. For instance, figure 9 shows the file download time with respect to the backoff time and a fixed time to wait-between-calls (WBC). The performance of the system (file download time), improves significantly by introducing a backoff time, but quickly tapers off at around 8 seconds for a setup in which the average call time is 3 minutes. It is worth highlighting though that while we achieved best performance with WBC set to a nominal 1 second, a WBC of zero (no wait between calls) makes the performance exponentially worse. A WBC of, say 25 seconds, however, makes the performance more variable, without resulting in any real performance advantage. We are currently exploring an adaptive policy for WBC such that the value of WBC is dynamically adjusted according to the congestion in the network. For instance, the value of WBC may be increased multiplicatively by a constant factor upon a failed call, and reduced correspondingly if a call succeeds.

3.4.3 Offline Block Discovery

In Bittorrent, a client starts the download of a file by acquiring from the tracker a list of peers that have chunks of the file. The client then connects with the set of 80-100 peers returned by the tracker (its peer swarm) and acquires from each a list of blocks currently stored at the peer (called peer handshake in Bittorrent). Subsequently, peers in a swarm keep each other informed about newly acquired blocks by sending "have" messages. All this mechanism of discovering file blocks rests on a Bittorrent client's ability to talk to multiple peers in parallel. Hence, this cannot be directly mapped to our p2p dialup architecture.

We implement block discovery in DitTorrent's offline mode by using a scheme inspired by distributed gossip protocols [6]. In this scheme, when a client calls a peer, it not only exchanges complementary file blocks, the peers also exchange lists of blocks discovered at hosts that previously connected to the clients (including their own). The aim is to virally spread the knowledge about the blocks stored at various hosts, while minimizing the number of connections required spreading the information. For instance, consider a DitTorrent client A that has previously connected with (either as a caller or callee) nodes B and C, each with the following sets of blocks: B{0,5,3}, C{1,2,9}. With our gossip-based approach, when a newly arrived client D (with zero current blocks) calls A, it is not only given a single block of file (c.f. DitTorrent bootstrap mode), it is also given lists of blocks stored at B and C (including the blocks exchanged in their connection with A). Once understood like this, it quickly becomes apparent that a client in this scheme will greatly benefit by initially calling nodes that have been around for a long time i.e. nearing completion.

Given this background, block discovery in DitTorrent's offline mode works as follows. Newly arrived clients "scrape" the tracker to find out the percentage of file downloaded by each client in its swarm (as reported by their last update message to the tracker). Armed with this information, the client goes into offline mode. In the offline mode, it first calls the host that has downloaded the maximum number blocks of the file. Of course, this could be a seed node, but a DitTorrent client only calls non-source seeds since they have acquired a diverse knowledge while working their way up from a single block to the completion of the file.

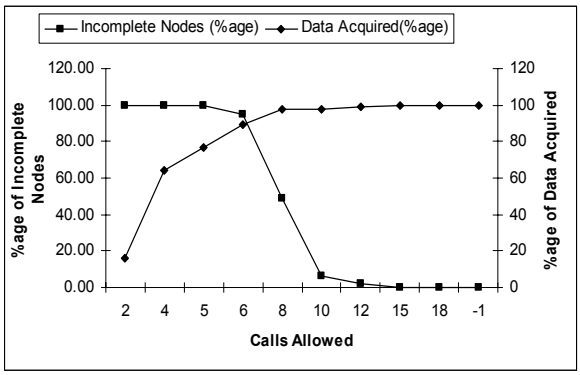

Figure 11: Analysis of budget-based download

Diversity of knowledge is important since the new client also has to work its way up to completion, while abiding by the block-wise tit-for-tat principle. In our implementation, a DitTorrent client initially calls a fixed number of peers, three in our current setup, that have downloaded the maximum fraction of the file. After this initial bootstrap, a client using offline block discovery proceeds in this mode till a time it has acquired information about all N blocks of the file. If a non-seed node is available, this information may be obtained in a single call. Subsequently, the client simply follows the greedy scheme when choosing peers to exchange blocks. Figure 10 compares the performance of DitTorrent's gossip-style offline block discovery with a setup in which a client has perfect knowledge of its peers as in Bittorrent. The performance of DitTorrent closely mirrors Bittorrent with its gossip-style offline block discovery, with the overhead of block discovery becoming more visible as we increase the number of file blocks (that must be discovered).

3.4.4 Budget-based Download

Often in our work, we were asked about the economic feasibility of our approach. This concern was raised because Internet is typically a flat-rate service while phone calls are charged by the minute. First, it is worth noting that POTS is increasingly becoming like Internet in terms of service charges; it is typical for telephone service providers to offer flat-rate regional or national plans. In case of such a flat-rate subscription, our approach offers "near-broadband" speed for no additional cost as long as calls are localized in the flat-rate region. DitTorrent clients can be configured to only call those peers that are within their calling "region" by, for instance, matching the ISP calling code with available peers. Importantly, given the p2p nature of our data transfer scheme, the burden of making a phone call is shared between peers; a peer downloads data both as a caller and a callee.

Initially, we considered providing an interface to the users to limit the number of calls to be made by the client when downloading the file. The intuition was that after the client has made the specified number of calls, it goes into a passive mode in which it simply waits to be called to acquire more file blocks. If a client fails to download the file in a specified time, it prompts the user to increase the call budget. In some sense, this scheme exposes a tradeoff of calling-cost vs. timely-download to the user. It is worth highlighting that with the current download speeds in the developing world, such an architecture is still very practical; more often than not, a user may be happy with a one-time extra cost to quickly download an important file, otherwise not possible with an extremely slow and intermittent Internet connection. However, we found that limiting the number of calls this way not only leads to starvation of peers, it results in an increase in the average number of calls each client has to make to download a file. Figure 11 shows the percentile of nodes that complete a file download as we increase the call budget. With a call-budget restriction, a 100% completion rate is achieved for a call-budget that is slightly worse than the average number of calls made in a setup without a call-budget (average number of calls without a call-budget in this simulation is 9).

4. IMPLEMENTATION

DitTorrent's implementation comprises of the following three components: DitTorrent Daemon, Browser plugin and the DitTorrent Tracker.

1. The DitTorrent Daemon is written in python 2.4 and is an extension of Bram Cohen's Bittorrent version 4.4.0. The Daemon includes the additional capability to make point-to-point telephone calls using PPP connections for data transfer.
2. The Browser plugin is a plug-in for firefox (xpi file) which detects and passes links .torrent files to the DitTorrent Daemon. The Daemon component also provides the URLs of the web pages for prefetching to the pre-fetching proxy. The prefetching proxy in our implementation extends the open source wwwoffle [7] proxy, specifically designed as a dialup offline proxy server.
3. The DitTorrent Tracker is written in C++ as an extension to the open source BNBT tracker [8]. Our extension to the BNBT tracker include time-window based lookups using efficient interval-trees, as well as compliance with additional parameters introduced to enable point-to-point dialup connections.

The source code of our implementation is available at [15]. The distribution at sourceforge also includes the DitTorrent simulator described in the paper.

5. SUMMARY AND FUTURE WORK

In the current age of broadband networks, dialup networking is mostly a forgotten technology. Perhaps the most relevant technologies for our system date back to the pre-Internet days, including FidoNET [5], UUCP [9] and USENET [10]. However, our use of an incentive-driven p2p data exchange mechanism makes our architecture fundamentally different from such systems. Importantly, unlike pre-Internet systems that relied solely on dialup connections for moving content between computers, our goal is to accelerate access to large content on Internet by utilizing a dialup connection at the maximum bandwidth supported by the modem. In our model, content still "resides" on the Internet, but may be downloaded using a p2p dialup connection to reduce download time. Our system combines a number of architectural components, such as p2p data transfer, intelligent connection interleaving and content-prefetching to make a practical system.

Our p2p data transfer layer, DitTorrent, derives all of its major design considerations from Bittorrent, and implements analogues of concepts such as Tit-for-Tat download, end-game mode, block-discovery, and bootstrap for our point-to-point dialup implementation. The Block-level Tit-for-Tat (BLTFT) policy in our system is derived from Bharambe et al [4]. Additionally, we revisit the policy of optimistic unchokes to prevent unnecessary connections in the point-to-point mode.

Currently, our system lacks comprehensive security architecture. We are currently working on rendezvous servers much in the same vein as the recently introduced Google click-to-call service which hides the identities of the peers connected on a dialup.

While this paper is focused on using POTS modem-to-modem dialup connections to circumvent extremely low bandwidth Internet connections, we intend to explore high-bandwidth intra-country deployments of Wireless Local Loop (WLL) and WiMAX as well. Our proposed architecture, at an abstract level, provides a mechanism for multiplexing the scarce and expensive international Internet bandwidth over higher bandwidth peer-to-peer connections within a developing country.

6. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their comments that helped improve this paper. The work presented in this paper was in part funded by the Microsoft Research (MSR) Digital Inclusion Grant, 2006. We would also like to thank Ghulam Murtaza for his help in the final structure and layout of the paper.

7. REFERENCES

[1] "The Digital Divide at a Glance", World Summit on the Information Society, Tunis 2005.

[2] Niraj Tolia, Michael Kaminsky, David G. Andersen, and Swapnil Patil, "An Architecture for Internet Data Transfer", *Proc. 3rd Symposium on Networked Systems Design and Implementation (NSDI)* San Jose, CA, May 2006

[3] Bram Cohen, "Incentives Build Robustness in BitTorrent", Workshop on Economics of Peer-to-Peer Systems, 2003

[4] AR Bharambe, C Herley, VN Padmanabhan. "Analyzing and Improving a BitTorrent Network's Performance Mechanisms", IEEE Conference on Computer Communications (INFOCOM), 2006.

[5] FidoNet, http://en.wikipedia.org/wiki/FidoNet/

[6] AJ Ganesh, AM Kermarrec, L Massoulie. "Peer-to-peer membership management for gossip-based protocols", IEEE Transactions on Computers, 2003.

[7] WWWOFFLE, http://www.gedanken.demon.co.uk/wwwoffle/

[8] BNBT, http://bnbt.depthstrike.com/

[9] UUCP, http://en.wikipedia.org/wiki/UUCP/

[10] Usenet, http://en.wikipedia.org/wiki/Usenet/

[11] Bharambe, A. and Herley, C. and Padmanabhan, V. N., "Microsoft Research Simulator for the BitTorrent Protocol," http://www.research.microsoft.com/projects/btsim.

[12] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks," SIGCOMM, Sep. 2004.

[13] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips, "A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System," Technical Report PDS-2004-003, Delft University of Technology, The Netherlands, April 2004.

[14] S. Saroiu, P. K. Gummadi, S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", Multimedia Computing and Networking 2002 (MMCN '02).

[15] DitTorrent, http://dittorrent.sourceforge.net/