A Simulation Study of a New Green BitTorrent

Jeremy Blackburn and Ken Christensen
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida USA
jhblackb@mail.usf.edu, christen@cse.usf.edu

Abstract— **The use of P2P technologies, such as BitTorrent, to distribute legal content to consumers is actively being explored as a means of reducing both file download times and the energy consumption of data centers. This approach pushes the energy use out of the data centers and into the homes of content consumers (who are also then content distributors). The current BitTorrent protocol requires that clients must be fully powered-on to be participating members in a swarm. In this paper, we show that simple changes to the BitTorrent protocol, including long-lived knowledge of sleeping peers and a new wake-up semantic, can enable clients to sleep when not actively downloading or uploading yet still be responsive swarm members. Using ns-2 we simulate a green BitTorrent swarm. We show that significant energy savings are achievable with only a small performance penalty in increased file download time.**

Keywords- P2P; BitTorrent; green; power management

I. INTRODUCTION

Servers in data centers are used to distribute digital content to consumers. This content includes text, images, audio, and (increasingly) video files. A major operational cost of a data center is electricity. Electricity cost is the second highest expense for a data center exceeding both hardware and software costs [9]. The EPA estimates that the nation's servers and data centers consumed about 61 TWh in 2006, which is 1.5% of the total US electricity consumption at a cost of about $4.5 billion (assuming commercial electricity rates) [16]. The energy use of data centers has doubled since 2000 [16]. We note that not all data centers are distributing content – many are running database and e-commerce applications. When a particular file becomes popular, the load on the servers in a data center can become very high from the resulting "flash crowd" and file download times will increase considerably. Thus, data centers used for distributing content present two open problems to scalability – growing power consumption and an inability to effectively handle flash crowds.

P2P technologies, such a BitTorrent, have been used in recent years to distribute digital content without a centralized server. P2P has been used to distribute illegal content (such a music and video files "ripped" from copyright sources). There is now a growing interest in using P2P to distribute legal content. Rodriguez et al. [17] are investigating the future of P2P as a fully distributed architecture for cost-effective distribution of legal digital content. They see P2P as the next logical step from Content Distribution Networks (CDNs) to improving content distribution in the Internet. Already, commercial offerings such as the VUDU on-demand movie service [22] are available that allow users to access digital video files via a P2P network. Open problems in digital rights management, charging models, and excessive bandwidth use between ISPs for P2P are currently being explored and solved (for example, by Choffnes and Bustamente [3]).

We foresee a future where residences connected to a broadband link may have a file sharing device, or unit, as part of a set-top box, or on the side of the house. This unit could be owned and controlled by the operator and would allow for on-demand content access to the local customer and distribution of operator-owned content to other customers. With this possible future model, energy use for content distribution will move largely from the data center to the consumer. If we assume 100 million file sharing units in the US alone, each consuming a modest 25 W on average when powered-on, then the overall energy use would be 22 TWh/yr at a cost of $2.2 billion (assuming a residential electricity rate of $0.10 per kWh) if all of these units were fully powered-on at all times. This is a non-trivial energy use that calls for an investigation into possible methods for reduction.

P2P protocols require clients to be fully powered-on at all times that they are part of the P2P overlay network or swarm (in the case of BitTorrent). BitTorrent uses TCP connections to determine membership in a swarm. Any client dropping its TCP connection (or unable to establish a connection) is assumed to be physically disconnected and "dead". A client must be fully powered-on to maintain or establish a TCP connection. It is this premise that we challenge in our work. We explore how the BitTorrent protocol can be changed such that a client with disconnected TCP connections can remain a member of a BitTorrent swarm to make it possible for a client to sleep when not active. The contributions of this paper are:

- A new green (power managed) BitTorrent whereby clients in a swarm can drop their peer TCP connections and go to sleep, yet still be active swarm members.
- A simulation evaluation of green BitTorrent measuring the trade-off in energy savings and download time.

The remainder of this paper is organized as follows. Section II is an overview of the standard BitTorrent protocol and describes how a dropped TCP connection affects swarm membership. Section III describes our new green BitTorrent. Section IV is a simulation evaluation of green BitTorrent. Section V describes related work. Section VI is a summary and describes future work. Section VI also estimates possible large-scale energy savings from green BitTorrent.

978-1-4244-3437-4/09/$25.00 ©2009 IEEE

```
/* if peer is marked dead, free it */
if (p->state & PEER_STATE_DEAD) {
  TAILQ_REMOVE(&sc->peers, p, peer_list);
  network_peer_free(p);
  sc->num_peers--;
  continue;
}
```

Figure 1. Code from Unworkable client for removal of disconnected peer

Event 1: Detection of TCP disconnect of a peer
1. on (detection of TCP disconnect of peer p)
2. remove peer p from my peer list

Figure 2. Description of event for removal of disconnected peer

II. THE STANDARD BITTORRENT PROTOCOL

BitTorrent [5] [6] [2] is one of the most popular P2P protocols for file sharing. BitTorrent accounts for somewhere between 18% to 35% of all traffic on the Internet [14]. BitTorrent is a swarm-based P2P protocol where clients download files in pieces. A swarm is a group of clients downloading and uploading one or more files. This ability to upload pieces of a file before the file is completely downloaded is what makes BitTorrent able to respond to flash crowds more effectively than other P2P protocols that must download a file in its entirety before being able to upload it. It is thus likely that BitTorrent, or a very similar protocol, will form the basis of any future P2P infrastructure for content distribution.

BitTorrent works as follows. A client seeking to download a file obtains a *torrent file* (from a torrent server) that serves as the metadata describing the file to be downloaded. A torrent file contains the host name of a *tracker* and lists the names of the pieces (and their checksums) that comprise the file to be downloaded. A tracker is a host that contains a list of all clients in a given *torrent*, or *swarm* (this is all the clients currently downloading and uploading the file associated with the torrent file). The client obtains from the tracker the hostnames of up to 50 other clients currently in the swarm and becomes a *peer* in the swarm. These client hostnames are entered into a *peer list*. To become a peer, the client connects – using TCP – to a finite number (max_connect) of the clients named or given to it by the tracker. Connected peers in a swarm periodically exchange information on what pieces they contain. This information allows each peer to request the pieces that it is currently missing from connected peers that contain the needed pieces. Peers that contain all the pieces of the file are called *seeds*. Peers that do not yet contain all the pieces of the file are called *leeches*. Leeches both upload and download pieces. Seeds only upload. To maximize download throughput, peers employ a *choking algorithm* [4] whereby they provide upload bandwidth to a peer in proportion to the download bandwidth that they are currently receiving from the peer. Peers also periodically, or opportunistically, connect to new peers in the swarm to test if the new peer can provide a better download rate than a current peer.

Key to the operation of BitTorrent is maintaining TCP connections to currently connected peers and being available to establish connections on demand with other peers. If a local peer detects that a TCP connection to another peer is disconnected, or a connection cannot be established, the disconnected or non-responding peer is dropped from the local peer list and is considered to be "dead". Figure 1 shows a code snippet from the Unworkable BitTorrent client [19] that implements this drop from peer list of a disconnected peer. Figure 2 shows an event-based pseudocode description of this drop from peer list. It is this event that requires power management to be disabled if a BitTorrent client is to contribute file pieces for upload after it has completed its download. Even if the client is very lightly utilized (that is, it has very few upload requests and is done downloading) it still must remain fully powered-on. A sleeping peer cannot maintain a TCP connection to other peers, or respond to a TCP connection request from other peers. It is this behavior that we address in order to develop a green BitTorrent protocol.

III. A GREEN BITTORRENT

A green BitTorrent client should be able to sleep whenever it is done downloading and has no current upload requests pending from its peers independent of how many TCP connections it may have to other peers. When a peer sleeps, it must not affect its state in the peer lists of other peers. An awake peer must always have a sufficient number of other peers that are awake (and it is connected to) to download from. Thus, a peer must be able to wake-up sleeping peers in a controlled fashion. We define new peer states, timers, messages, and events to achieve these objectives.

For a green BitTorrent client we define three possible states for a peer p as seen by "me" (or "this peer"), they are:

- Unknown – a peer that has been given to this peer by the tracker, and it is unknown if the peer is sleeping or awake.
- Connected – a peer that this peer has an active TCP connection with. File pieces can be uploaded and downloaded on the connection.
- Sleeping – a peer that has disconnected its TCP connection with this peer. The TCP connection must be re-established before file pieces can be uploaded or downloaded.

In the peer list in each peer, the state of every other peer p is kept as p.unknown, p.connected, or p.sleeping. One new timer is defined, the inactivity timer. The inactivity timer times-out on peer inactivity. This timer is reset and restarted whenever a download or upload activity occurs. One new message is defined, the wake-up message. The wake-up message could be a standard Magic Packet [12] or other predefined packet type that triggers a wake-up of a file sharing unit by the NIC in the unit that receives it.

Figure 3 shows the four events for a green BitTorrent client. The first two events are existing events with new operations. The third and fourth events are entirely new. The events are:

Event 1: In this existing event rather than removing the disconnected peer from the peer list, its state in the peer list is changed from p.connected to p.sleeping.

Event 2: This event is also present in an existing BitTorrent client, but existing clients only check for new peers in the tracker (line 2). When the modified Event 1 marks a peer as Sleeping, the peer no longer is participating in uploading its pieces. As the make-up of the swarm changes over time, the pieces in a sleeping peer might be needed. Thus, in the green BitTorrent client the following new actions occur. All new peers given by the tracker are labeled as Unknown (they can be sleeping or awake) in lines 3 and 4. While the number of connected peers is less than a predetermined max_connect (line 5) the peers in the peer list are tested for possible wake-up. Peers are selected randomly from the peer list. The specific wake-up condition in line 9 is:

```
(p.state == unknown)
```

if this peer is a seed, and is

```
((p.state == unknown) OR (p.state == sleeping))
```

if this peer is a leech. A peer that is to be woken-up is sent a wake-up message and then a TCP connect is attempted. If the TCP connect attempt fails, the peer is considered to be "dead" (for example, has been physically removed from the network) and is then appropriately removed from the peer list.

Event 3: In this new event, the new inactivity timer is used to detect when a seed has not been actively uploading for a fixed period of time (the inactivity timer preset value). Expiration of the inactivity timer indicates that the pieces in a seed are no longer being downloaded, and thus the client can (and should) go to sleep. Prior to entering the sleep state, the seed sends a not interested message to every connected peer and then also chokes every connected peer. These two protocol messages determine a sleeping seed's state in the swarm as it will not be downloading from, nor responding to, any download requests for pieces from other peers. After broadcasting its intentions to connected peers, TCP connections are closed and the client goes to sleep. At this point, the client is waiting for a wake-up message to resume operation (such as an upload).

Event 4: In this new event a client wakes-up on receiving a wake-up message and a TCP connection is established by the peer that initially sent the wake-up message. When the connection is established, the client sends its file contents to the peer that woke-up the client and also runs the choking algorithm. At this point, the now awake and connected client is a fully operational peer ready to respond to upload requests.

In Event 2 peers are woken-up. Which peers should be woken-up, those in the sleep state or those in the unknown state? Note that sleeping peers can only be seeds (since leeches are by definition actively downloading, if not also uploading, and thus are too busy to sleep). A sleeping peer may have previously given a good download rate, and it would thus make sense to wake it up. However, a new peer in the unknown state may make an even better connected peer. In our implementation sleeping peers are intentionally woken-up only by leeches. It does not make sense for a seed to wake-up another seed since uploading (and downloading) is not called for between two seeds. However, seeds that are awake may still wake-up sleeping seeds inadvertently if a sleeping peer is given to them by the tracker (and thus that seed being initially set to p.state = unknown in line 4 of Event 2).

```
Event 1: Detection of TCP disconnect of a peer
1.  on (detection of TCP disconnect of peer p)
2.      p.state = sleeping

Event 2: Time out of connection timer
1.  on (timeout of connection timer)
2.      check with tracker for new peers as needed
3.      for (all new peers in peer list)
4.          p.state = unknown
5.      while (count of connected peers < max_connect)
6.          p = randomly selected peer in my peer list
8.          if (have tested all peers) exit this loop
9.          if (wake-up condition == true)
10.             send wake-up message to peer p
11.             try to connect to peer p
12.             if (TCP connection established)
13.                 p.state = connected
14.             else
15.                 remove peer p from my peer list
16.     restart connection timer

Event 3: Timeout of inactivity timer
1.  on (timeout of inactivity timer)
2.      send not interested message to connected peers
3.      send choke message to connected peers
4.      close all of my TCP connections
5.      my.state = sleeping
6.      enter sleep state

Event 4: Detection of my wake-up triggered by peer p
1.  on (detection of my wake-up triggered by peer p)
2.      if (TCP connection is established from peer p)
3.          my.state = connected
4.          send my file contents bitfield to peer p
5.          run choking algorithm
```

Figure 3. Description of new/changed events in green BitTorrent client

A. *Backwards compatibility with existing BitTorrent protocol*

Green BitTorrent clients are backwards compatible with existing BitTorrent clients, however their participation in a swarm will degrade the performance of standard (non-green) clients unless additional measures are taken. A standard BitTorrent client will drop any peer that goes to sleep (as described in Section II and in Figures 1 and 2). A Green BitTorrent client could detect if any peers are not green and then revert to existing, standard operation. The methods to signal peer type (green or standard non-green) are beyond the scope of this paper.

IV. SIMULATION EVALUATION

We evaluated the potential energy savings of a green BitTorrent client using the ns-2 simulator [13]. We started with the BitTorrent packet-level simulation model developed by Eger et al. [7]. The model by Eger et al. includes an underlying

TCP model that is part of ns-2. The BitTorrent model is modular allowing for replacement of peer and piece selection algorithms. This model was modified to implement the green BitTorrent events described in Section III.

We view a BitTorrent system as having control and response variables to be manipulated and measured, respectively. The control variables for the system configuration were:

- Number of swarms a peer participates in
- Number of peers to maintain in a swarm
- Number of connected peers (max_connect)
- Number of seed peers at the start of a swarm
- Number of leech peers
- File size
- Upload and download bandwidth for peers
- RTT between peers

The control variables for a green BitTorrent peer were:

- Transition time to wake-up and go to sleep
- Connection timer preset value
- Inactivity timer preset value

The control variables for system workload were:

- Interarrival time distribution for peers entering a swarm
  - Distribution parameters including mean ($T_{arrival}$)

The response variables for a peer were:

- Sleep time
- Awake time
- File download time

A. Description of the experiment

We designed an experiment to measure file download, sleep, and awake times as a function of the interarrival time of peers into a swarm. Our system modeled a single swarm of 50 peers (a typical large swarm) with one additional initial seed peer (this initial seed peer was set to never sleep in any of the cases) and the rest of the entering peers initially containing no pieces. We used a max_connect value of 5 (the typical value for a BitTorrent client). We used a file size of 1 GB corresponding to a small video file. File pieces were 256 KB. We assumed an upload data rate of 2 Mb/s and download data rate of 10 Mb/s per peer corresponding to Verizon residential FIOS 2008 base rates [20]. The RTT between peers was 10 ms modeling a swarm contained within the domain of a single ISP.

Parameters specific to our green BitTorrent were set as follows. The wake-up and go to sleep transition times were 300 ms each. This is, we believe, a reasonable time for an operating system to save its state and for a processor to recover this state and resume execution. The Linux-based OLPC machine requires only tens of milliseconds to wake-up [21]. The connection timer was set to 300 s (5 min). This is the default connection timer value used in BitTorrent client implementations for checking with the tracker for new peers. The inactivity timer was set to 15 s. This value was selected as reasonable to prevent oscillation between awake and sleep.

Peers arrived into a swarm as a Poisson process (that is, interarrival times are exponentially distributed). This models independent human behavior very well. The mean interarrival times were varied from very short to model a flash crowd to very long to model peers arriving into a swarm where all other peers had already completed their downloads (that is, all "old" peers were seeds). The mean interarrival times used were, $T_{arrival}$ = 0, 0.5, 1, 2, 4, 8, 16, and 32 minutes. The time to download a 1 GB file at 10 Mb/s is about 14.3 min, thus the 16 min and 32 min mean interarrival times will have most, if not all, peers arriving into a swarm where all other peers have already completed their download.

Two systems (or cases) were modeled – a control network consisting of standard BitTorrent where no peers could sleep and a green BitTorrent network. For each interarrival time we ran 30 replications, each replication with a different initial random number seed. The average of the replications was computed and plotted.

B. Results from the experiment

The results from the experiment are plotted in six graphs, three for the case of standard BitTorrent (Figures 4, 5, and 6) and three for green BitTorrent (Figures 7, 8, and 9). Figures 4 and 7 show mean download times for the 1st, 25th, and 50th client (peer) to enter the swarm as a function of peer interarrival time. Figures 5 and 8 show awake and sleep time for all peers for a mean peer interarrival time of 16 min. Figures 6 and 9 show awake and sleep times for the 25th client as a function of interarrival time. The gray areas of Figures 8 and 9 show sleep times for green BitTorrent clients. From Figures 4 and 7 it can be seen that:

- The mean download time for client #1 is high and fairly constant independent of the client interarrival time and case.
- The mean download time for clients #25 and #50 decreases as the interarrival time increases for the standard BitTorrent and green BitTorrent.
- Green BitTorrent has larger file download times than standard BitTorrent. Table I shows the percentage increase in download time for green BitTorrent compared to standard BitTorrent.

From Figures 5 and 8 it can be seen that:

- There is no sleep time for standard BitTorrent, but there is considerable sleep time for green BitTorrent. For standard BitTorrent the sum of the awake times for the entire swarm is 324.6 hours, for green BitTorrent it is 72.1 hours (253.8 hours is now spent in sleep). This represents an energy savings of 77.8% measured over all peers.

From Figures 6 and 9 it can be seen that:

- There is no sleep time for standard BitTorrent, but there is considerable sleep time for green BitTorrent. For standard BitTorrent the sum of the awake times for client #25 for all interarrival times is 29.6 hours, for green BitTorrent it is 10.2 hours (19.9 hours is now spent in sleep). This represents an energy savings of 65.5% measured over all interarrival times.

C. Discussion of the experiment results

Two key questions emerge from the observed results, they are 1) why does client #1 always have a large download time that is roughly the same for green and standard BitTorrent, and 2) what explains the slightly larger download time for green BitTorrent as compared to standard BitTorrent?

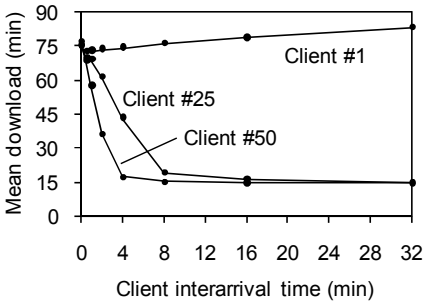

Figure 4. Download time for standard BitTorrent

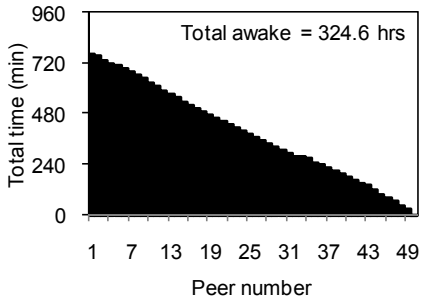

Figure 5. Sleep for standard BitTorrent

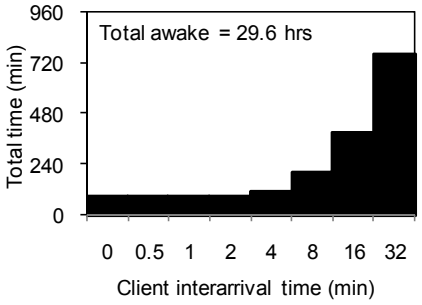

Figure 6. Sleep for peer #25 standard BitTorrent

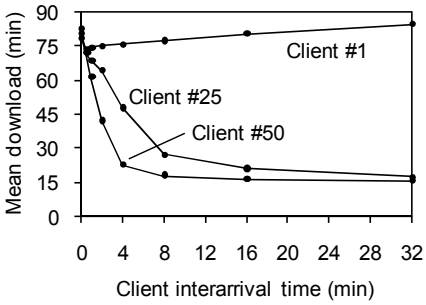

Figure 7. Download time for green BitTorrent

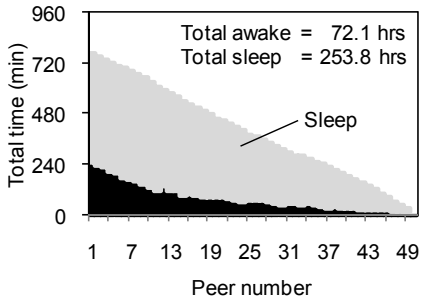

Figure 8. Sleep for green BitTorrent

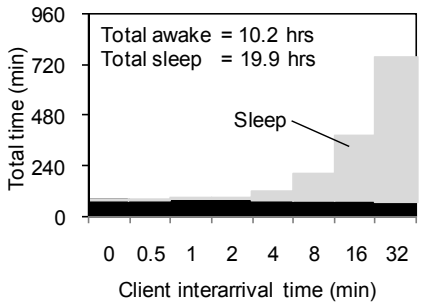

Figure 9. Sleep for peer #25 green BitTorrent

TABLE I. INCREASE IN DOWNLOAD TIME FOR GREEN BITTORRENT COMPARED TO STANDARD BITTORRENT

	Mean client interarrvial time							
Client number	**0 *min***	**0.5 *min***	**1 *min***	**2 *min***	**4 *min***	**8 *min***	**16 *min***	**32 *min***
Client #1	4.8 %	1.5 %	1.8 %	1.9 %	1.9 %	1.9 %	2.1 %	1.6 %
Client #25	6.7	3.7	–0.7	5.2	11.0	44.8	30.9	18.1
Client #50	7.0	4.6	6.0	15.5	32.8	18.7	9.4	8.2
Average for all	4.9	3.4	3.5	5.2	13.7	23.0	22.0	18.3

For the first question, client #1 always arrives into a swarm with only one initial seed present. Later clients arrive into a swarm with many peers present (of which many are also seeds by virtue of already having completed their download). When the client interarrival time is small, other peers will arrive and be present during the download period for client #1. Thus, client #1 can download from multiple peers in the case of small interarrival times, but not so when interarrival times are large. A slight reduction in download time can be seen for the small interarrival times (but, at these times there will also be competition from other peers preventing client #1 from getting a full download rate from the other peers).

For the second question, the larger download times for green BitTorrent can be explained as seeds going to sleep putting green BitTorrent at a disadvantage with regards to download time (compared to standard BitTorrent). As there are fewer awake peers in the green BitTorrent swarm, there are also then fewer peers that might initiate a new inbound connection. While the number of outbound connections is limited, inbound connections (and thus the total number of connections) are effectively not limited. This reduces the overall possible bandwidth available to a given peer, however it is extremely dependent on the make-up of the swarm at any given time. The time to wake-up sleeping seeds also factors into the increased download time for green BitTorrent.

V. RELATED WORK

To the best of our knowledge, this is the first work to specifically look at, and offer a specific proposal for improving, the energy efficiency of swarmed P2P by focusing on the P2P protocol itself. We see this as a first step towards "green applications". Previous work has studied how proxying could reduce energy use of Gnutella-like P2P protocols [10]. Significant work has been done in studying how to improve the energy efficiency of edge devices in the Internet, including both servers and clients.

Virtualization is a key method used for improving energy efficiency of servers in data centers. Virtualization offers a means of reducing (or consolidating) the number of server machines or blades and increasing the utilization of remaining machines. This work does not apply to our domain of P2P nodes where it is envisioned that homes will have independent P2P units to both serve content to the home entertainment system and to other P2P units in other homes as part of a P2P swarm. Consolidation of P2P units between homes does not make sense in this future scenario.

Network connectivity proxying (NCP) has been proposed and studied as a means of enabling edge devices, such as desktop PCs, to sleep and maintain network connectivity (for example, [1], [8], [11], and [18]). An NCP, or proxy, is an

entity that maintains full network presence for a sleeping network host. In [1] it is described how a proxy – possibly implemented within a NIC – could complete a BitTorrent download for a sleeping PC, but the means to also upload (share content) from a proxy is not feasible. A recent work by one of the authors has looked at preserving TCP connections for sleeping end devices with a SOCKS-like service in a switch [11]. Preserving TCP connections is only part of the problem with BitTorrent; responding to BitTorrent protocol messages in a timely fashion is also critical. Proxying of TCP connections offers another direction to be explored as future work to achieve more energy efficient BitTorrent operation.

VI. Summary and Future Work

We have shown how relatively simple changes to BitTorrent can achieve a more energy efficient operation – a green BitTorrent. These changes allow peers to sleep without being dropped from peer lists. Effectively, we have decoupled TCP connection state from peer state. We implemented our new green BitTorrent in the ns-2 simulation framework and showed that green BitTorrent could consume less than 25% of the energy as standard BitTorrent (where all clients are fully powered-on 24/7) with only modest penalty in increased download time. For small interarrival times, file download time is increased by less than 10% (as seen in Table I). For medium and large interarrival times, download time is increased by up to about 45%, but typically by much less (as seen in Table I). A 45% increased download time would be the difference between 18.8 min and 27.3 min to download a file.

The overall energy savings achievable with green BitTorrent if clients sleep 75% of the time could be *over $1.6 billion per year* in the US alone if we assume 100 million file sharing units, each consuming a 25 W on average (and assuming a residential electricity rate of $0.10 per kWh). This level of savings appears to be feasible based on the methods developed and evaluated in this paper.

For future work we are currently building a prototype green BitTorrent client and will use it to measure download time and energy consumption trade-offs in an experimental swarm in PlanetLab [15]. We plan to extend our simulation evaluation to study the effects of key control variable including membership in multiple swarms, heterogeneous link rates, and varying RTTs to complement our work with a real client in PlanetLab. We would also like to compare green BitTorrent content distribution to a centralized server-based distribution for both performance and energy consumption. Finally, we would like to gain a better understanding of how proxying of TCP connections could be useful for allowing BitTorrent peers to sleep and not lose peer state.

Acknowledgment

The development of this material is based upon work supported by the National Science Foundation under grant 0754537 (Blackburn) and CNS-0520081 (Christensen). The authors would like to acknowledge the use of high performance computational services provided by Research Computing, University of South Florida. The authors thank the anonymous reviewers for their helpful comments.

References

[1] Y. Agarwal, S. Hodges, J. Scott, R. Chandra, P. Bahl, and R. Gupta, "Augmenting Network Interfaces to Reduce PC Energy Usage," to be submitted, 2008. URL: http://research.microsoft.com/users/ranveer/docs/somniloquy.pdf.

[2] BitTorrent, Inc. 2008. URL: http://www.bittorrent.com/.

[3] D. Choffnes and F. Bustamente, "Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems," *Proceedings of ACM SIGCOMM*, 2008.

[4] B. Cohen, "The BitTorrent Protocol Specification," Standard, January 10, 2008. URL: http://bittorrent.org/beps/bep_0003.html.

[5] B. Cohen, "Incentives Build Robustness in BitTorrent," *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[6] B. Cohen, "The BitTorrent Protocol Specification," January 10, 2008. URL: http://www.bittorrent.org/beps/bep_0003.html.

[7] K. Eger, T. Hoßfeld, A. Binzenhöfer, and G. Kunzmann, "Efficient Simulation of Large-Scale P2P Networks: Packet-level vs. Flow-level Simulations," *Proceedings of the Second Workshop on Use of P2P, GRID and Agents for the Development of Content Networks*, pp. 9-16, 2007.

[8] C. Gunaratne, K. Christensen, and B. Nordman, "Managing Energy Consumption Costs in Desktop PCs and LAN Switches with Proxying, Split TCP Connections, and Scaling of Link Speed," *International Journal of Network Management*, Vol. 15, No. 5, pp. 297-310, September/October 2005.

[9] "Increasing Data Center Density While Driving Down Power and Cooling Costs," Intel White Paper, 2006. URL: ftp://download.intel.com/design/servers/technologies/thermal.pdf.

[10] M. Jimeno and K. Christensen, "A Prototype Power Management Proxy for Gnutella Peer-to-Peer File Sharing," *Proceedings of the IEEE Conference on Local Computer Networks*, pp. 210-212, October 2007.

[11] M. Jimeno, K. Christensen, and B. Nordman, "A Network Connection Proxy to Enable Hosts to Sleep and Save Energy," *Proceedings of the IEEE International Performance Computing and Communications Conference*, pp. 101-110, December 2008

[12] Magic Packet Technology, AMD, 1998. URL: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/20213.pdf.

[13] The Network Simulator – ns-2, 2008. URL: http://www.isi.edu/nsnam/ns/.

[14] P2P Research Institute, 2008. URL: http://p2presearch.com/.

[15] PlantLab, 2007. URL: http://www.planet-lab.org/.

[16] Report to Congress on Server and Data Center Energy Efficiency, Public Law 109-431, U.S. Environmental Protection Agency ENERGY STAR Program, August 2, 2007.

[17] P. Rodriguez, S-M. Tan, and C. Gkantsidis, "On the Feasibility of Commercial, Legal P2P Content Distribution," *ACM SIGCOMM Computer Communication Review*, Vol. 36, No. 1, pp. 75-78, January 2006.

[18] "TC32-TG21 – Proxying Support for Sleep Modes, Ecma International," 2008. URL: http://www.ecma-international.org/memento/TC32-TG21.htm.

[19] Unworkable: A Free BitTorrent Implementation, P2P Research Institute, 2008. URL: http://p2presearch.com/unworkable/.

[20] Verizon FIOS Internet Packages and Prices, 2008. URL: http://www22.verizon.com/content/consumerfios/packages+and+prices/packages+and+prices.htm.

[21] W. Vota, "Speedy OLPC Suspend/Resume," One Laptop Per Child News, October 5, 2006. URL: http://www.olpcnews.com/software/operating_system/speedy_olpc_suspend.html.

[22] VUDU, 2008. URL: http://www.vudu.com/.