BitRiver: Final Report

Jonathan Stiansen, Sampoorna Biswas, Wali Usmani

April 24, 2015

Abstract

Peer-to-peer multimedia sharing has become widely popular due to its robust, fault-tolerant nature. We propose modifications to the popular peer-to-peer file sharing protocol, BitTorrent, to allow users to stream multimedia content quickly. We outline and implement several possible solutions, that replace the rarest-first policy of downloading pieces of the file, with a policy that downloads pieces in chronological order. We test our chosen solution, the BitRiver protocol, on a simulated swarm and report its results. Our findings demonstrate that our proposed modifications make it feasible to stream high quality multimedia using the BitTorrent protocol and overcome standard limitations of the client server streaming model.

1 Introduction

In recent years, media viewing services such as Netflix and YouTube has seen a huge growth in popularity. They account for the largest consumption of bandwidth on the internet, consuming over 42% during peak hours [12]. BitTorrent, a peer-to-peer file sharing service, comes in third, accounting for almost 6% of the total bandwidth in the U.S. and much more in other parts of the world, where Netflix is not available [12]. It it majorly used to download large (greater than 100 MB) multimedia files. Due to its P2P nature, the BitTorrent service does not suffer from many of the shortcomings that a client-server model would have - it is robust, fault-tolerant, and can scale very well to a large number of peers. Since there's no server, there's also no server bandwidth bottleneck. Moreover, peer-to-peer solutions neither require any special support from the network, nor any special hardware, which makes it easy for anyone to join the system. They also do not suffer from the country-specific restrictions that YouTube and Netflix have.

All of these factors contribute to BitTorrent's popularity and make it a good candidate for downloading high-quality multimedia on a large-scale.

However, BitTorrent is not designed for streaming multimedia; one typically has to wait for the whole file to be downloaded before they can play it. In this paper, we explore the BitTorrent protocol and propose various modifications that allow it to be used in media streaming. We implement our solutions, and deploy them in a simulated environment to test their effectiveness. We find that with a few basic modifications, streaming is quite feasible under normal circumstances.

1.1 Contributions

- We propose three strategies for downloading content in order
- We share an open-source BitTorrent client implementation of these strategies
- Demonstrate the efficacy of said strategies for BitTorrent streaming in a controlled environment

2 The BitTorrent Protocol

Here we describe the key aspects of BitTorrent (BT) protocol briefly; a more detailed description can be found at [4].

To use BT, users download a BitTorrent client and join one of the many *swarms* which allow many clients to upload and download a file simultaneously. Swarms have a centralized *tracker* that stores metadata about the file and helps peers connect to each other. BitTorrent recognizes two kinds of peers - *leechers* and *seeders*. Leechers do not have all pieces of the file, and participate by downloading and uploading chunks simultaneously, whereas seeders have all the pieces of the file, and participate by only uploading to other peers.

A file download on BT differs from traditional download in that, that the file is divided into pieces, or *chunks*, and peers download a file piece by piece. They are not sequentially downloaded, but instead, in a *rarest-first* or randomized fashion. This provides greater availability of the pieces of the file. The rarest-first heuristic prioritizes downloading the pieces that the least number of peers currently have, minimizing the likelihood of a download getting stranded due to seeds leaving the swarm.

BT also follows a *tit-for-tat* policy to discourage free-riders, that is, users that only download and do not contribute any pieces. Peers only upload to the k peers that provide the best download rate, *choking* the others for a preset period of time. Every couple of rounds, it *optimistically unchokes* peers from among the peer list. This allows it to search for peers that can potentially provide a better download rate, and gives newcomers a chance to join the swarm.

Finally, when the download is nearing completion, BT goes into *end-game mode*, and sends out requests to all the peers. Once all the pieces have arrived, it sends cancel requests. Although this adds a minor overhead, it speeds up the end of the download.

3 Related Work

There has been considerable work on the BitTorrent protocol. Many have studied its incentive system, and developed alternative solutions building upon the same protocol. BitTyrant [11] and BitThief [8] are two examples in this body of work, but they do not focus on streaming.

There have also been a few papers that have studied modifications for streaming using BT. A Livestreaming protocol, using BitTorrent as a foundation, was proposed in [10]. However, the solution requires a large overhaul of the protocol, to the point that it cannot interact with other BT clients. BT's rarest-first piece selection policy is ill-suited to on-demand video playback, and

research that focuses on alternative strategies can be broadly classified into probabilistic [3, 17] and window-based [2, 14, 13] approaches. Probabilistic approaches choose the next chunk to be downloaded based on some probability distribution, and window-based approaches employ a moving window scope of the next chunks to be downloaded.

Our solution proposes piece selection strategies that prioritize downloading the next-to-be-played pieces. While in [14], the authors claim that a piece selection strategy alone is not enough, and also suggest modifying the incentive system, we consider that to be out of the scope of this paper.

In [15], the authors compare and contrast three approaches from previous papers - [14, 17, 13] - to determine whether to drop data or stall the video when there is a delay. They conclude that stalling the video provides a better user experience. Although we do not implement a playback mechanism while streaming, following [15], we expect to stall the video during data delivery delay as well.

4 Proposed Modifications

We modify Taipei Torrent, an open-source BitTorrent client written in Go [9]. Taipei Torrent allows us to run the client both as a tracker, and as a peer. However, instead of the rarest-first policy, it downloads pieces in randomized fashion. We describe their piece selection strategy in Algorithm 1.

Data: peer p
$n \leftarrow$ total pieces of the file;
$start \leftarrow$ rand.Integer(n);
$piece \leftarrow$ checkRange(p, $start$, n);
if *piece == -1* **then**
  $piece \leftarrow$ checkRange($p, 0, start$);
else
  do nothing ;
end

Algorithm 1: Taipei Torrent's piece selection algorithm

CheckRange(peer, start, end) "returns the first piece in range (start, end) that is not in the torrent's pieceSet but is in the peer's pieceSet" [9]. Since the starting point to this function is randomized, the pieces are requested in a somewhat random order. We describe our modifications in Algorithm 2.

```
Data: torrent file, peer list, tracker
activePieceIndex ← next piece needed;
ready[] ← list of ready peers;
active[] ← list of peers currently downloading from;
for peer i in ready[] do
    if i has activePieceIndex then
        move i to active[];
        download piece;
        activePieceIndex++;
    else
        mark as "Not Interested";
    end
end
```

Algorithm 2: BitRiver streaming

Besides the basic modifications outlined in Algorithm 2, we made the following additional changes

- **End-game mode:** We experimented with adding end-game mode towards
  - only the end of the download
  - both the beginning and the end of the download
- **Hybrid approach**, where we download rare pieces and the closest-to-front pieces, in parallel (Fig. 1)

We were working with an existing codebase, and we were unable to implement all of our proposed modifications within the given time frame. We could not implement the hybrid approach, since that would have required a major overhaul of the existing codebase. However, we implemented the others, and we decided to focus on testing them. If a few basic modifications were effective in streaming multimedia, then it is likely that more sophisticated strategies would perform even better.

Going into end-game mode towards the end of a download is standard practice. We experimented with adding it at the start as well, in the hope that it would allow us to quickly build a viewing "buffer" for the file. Unfortunately, during preliminary testing, we found out that it resulted in our client being choked by other peers. We decided to hold out on that one for further experimentation. The protocol we selected to run further experiments on, is called BitRiver.

5 Evaluation

Our evaluation methodology measures the success in terms of being able to create and maintain a multimedia stream through the BitTorrent protocol, the performance of our modified client and the network/swarm conditions necessary to allow smooth system operation. We attempt to:

- Get a baseline download time in standard BitTorrent client and test our modified client in similar conditions to observe whether our download time is comparable.

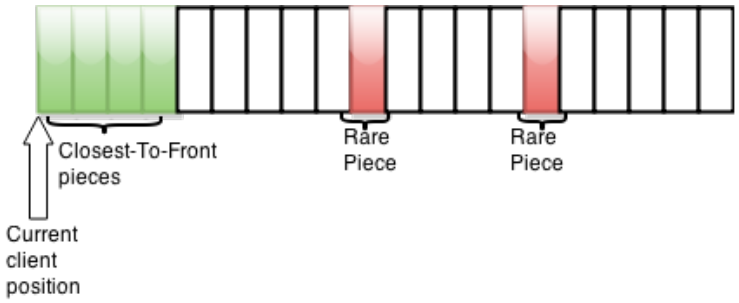

Figure 1: Hybrid Rarest-First and Closest-to-Front Approach

- To stream an audio file and if we are successful, to examine the minimum conditions required to do so in terms of torrent health (ratio of seeders to leechers) and maximum available bandwidth.

The goal for evaluation is not simply to validate BitRiver as an alternative and useful piece selection heuristic, but also to show why it would be better to use such a system over a client server model which suffers from bandwidth bottlenecks. More specifically, if the server is overloaded due to multiple clients, the server's connection to the network would prove to be a bottleneck. BT on the other hand does not rely on a single content provider. Thus even if a single seed has limited bandwidth to upload, the combined bandwidth of the other seeders means that the bottleneck is only the client's own connection to the network.

The type of media will cause the definition of adequate to vary, as watching high definition video for one minute could be close to listening to over an hour of audio [1]. Specifically the download is a success for a given bitrate if it can supply, on average, at least as much Kbps as streaming consumes. It can be slightly less in that some data can be downloaded before the user starts watching, as long as that required offset is only a small (20% or less) portion of the total time to download the show.

6 Experimental Set-up

We run multiple experiments on the client and the environment to test their impact on content streaming.
We do three different tests for each of the four conditions. The first test includes 1 seeder, and one client, the second is two seeders, and the third is three seeders. Each seed is run on a computer in the UBC department of computer science, and connected to the same swarm, which we also run from a UBC computer. Each seeds the same 63.9 MB data file. Our control condition uses the original Taipei Torrent implementation. The control group's results for each of the three tests is recorded, that is, download and upload rates are recorded for each test. After the system is set up, we modify network configurations, link capacity, and ration of seeders to leechers in the system to better measure the minimum requirements required to achieve stable byte streaming. As part of the BitTorrent protocol the last block may or may not have all pieces downloaded. To measure we

will only count the number of pieces, not the blocks, which were downloaded in each second. We will mainly examine total download times, when using the streaming service. The efficacy of a specific model will be determined if it could finish in the required time based on a given bitrate, or if it was sufficiently close to that time.

6.1 Environment

We simulate a BitTorrent swarm on a Mininet [7] environment. This environment is fairly new, but has been evaluated in it's ability to simulate real systems[5]. They found that normal bitrate ranges, Mininet works as expected - only with very low bitrates does it not work effectively. This was acceptable for us since those bitrates were far lower than we tested in this paper.

6.2 Torrent Client

As mentioned above we use a BitTorrent client by the name of Taipei-Torrent. The client is implemented in the Go programming language, and is able to be run only through a Unix command line. It has some limitations, it is not a commercial client, meaning it is not used by 'normal' users. It is only a few years old, with little active development. To determine it's efficacy we downloaded multiple real torrents to see if it works roughly as expected, and we found that it did. Since this project is open sourced, we downloaded and modified the source code with our strategies. It's important to note that this client does not download rarest pieces first, instead it chooses it's blocks at random.

6.3 Topology

Or simulated network used a Star topology. This star topology uses a central switch, and each node in the system directly connects to that switch, and nothing else. Therefor all simulated network communication went through that switch.

6.4 Nodes

This system connected 20 nodes and one tracker to a single switch. There were four types of nodes in the system. The first was the single streaming node, using our Taipei-Torrent client implementation. One was a tracker, which kept track of nodes and churn. There was a fixed number of permanent seeds in the system, which varied based on each experiment, and the remainder were peers downloading the same file.

Since each node had only one connection, the connection to the tracker, it was that connection that was throttled to emulate different network speeds. Each experiment had a fixed number of peers seeding the file, and each experiment had different bandwidth allowed to all files. In each, the bandwidth was fixed. The first round of experiments had 0.4 Mbps, then 0.8 Mbps, and last 1.2 Mbps. These speeds are set through Mininet.

6.5 Peer Churn

Peer churn has been researched extensively [16]. We chose to model our peers in a simple way. We simply started all peers at the same time as our client. When a peer finished downloading it's downloading it's file, it immediately disconnected from it's tracker, deleted it's file and was rebooted.

6.6 Tracker

For a tracker, we use a simple BitTorrent tracker, and not a distributed hash table. The tracker we use is also part of Taipei-Torrent. In between each experiment the tracker was reset, to remove any past 'seed' or 'peer' IP addresses.

6.7 Seeding

One of our experimental variables was seeders, where those seeders were held constant for the duration of each experiment. We felt that it is a reasonable approximation of the fact that some users leave their Torrent clients on for a long period after downloading, we are simply making the assumption that the short duration which we are downloading, is not the duration that they decide to shut off their seed.

6.8 Downloaded File

Our primary goal in this experiment is to measure download rate, not to actually demonstrate streaming. To this end we held our file constant, and we simple extrapolate on the actual download speeds to determine if streaming at different qualities is possible. We used a file, which had a total size of 63.9 MegaBytes. This file, if it required a streaming rate of 400 KiloBytes per second would be a 21 minutes film. At the semi-high definition bitrate of 800 KBps it would be only 10.5 minutes. Finally, a movie of it's size in super high-def it would be only 4.7 minutes.

6.9 Torrent File

For a torrent to be downloaded, and originally uploaded, a Torrent file must be created which includes a IP address of the tracker (s) which are currently managing the file. Those trackers can then give a downloader a reasonably current list of possible uploaders of the file. We first downloaded a torrent file from eztv.it, which was for the above file. We used torrent-editor.com to modify the tracker information to match our own tracker.

6.10 Streaming vs Random : Speed Test

To ensure fair and realistic conditions to test how fast the steaming heuristic is compared to Taipei Torrent's Random piece selection heuristic, we referred to [6]. The paper performed rigorous analysis of above 300000 torrents on the popular torrent site, PirateBay to discover that movies have an average Seed to Leach Ratio of 1.4. This translates to 11 seeds out of 19 peers in a 20 node swarm. Therefore, in the Speed Tests on various peer bandwidths, we fixed the number of seeds to 11 and leaches to 8. See figure 2.

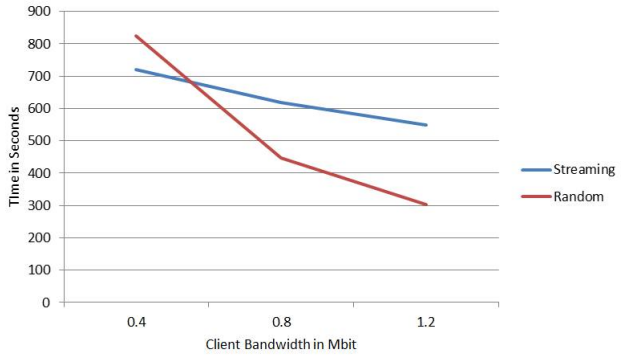

Figure 2: Speed of Download Methods: Streaming vs Random

7 Results and Discussion

The below table shows our mean finishing time for each set of trials done (MT) along with the variance of those times (V). Both columns are in seconds. For these, our number of trials were 9 - with the exception of 17, and 7 where for some reason our torrent files were choked each time they were run. The different different sets of columns, separated with a single line, represent different megabits per second (Mbps) trials. Each row is the number of seeds (out of 20 peers).

Trials	0.4 Mbps		0.8 Mbps		1.2 Mbps	
	Mean	Variance	Mean	Variance	Mean	Variance
5 Seeds	1711	1363032	814	2992	713	8948
7 Seeds	573	90	683	675	-	-
9 Seeds	855	2999	688	62493	603	3098
11 Seeds	709	647	563	335	575	4295
13 Seeds	601	51	546	190	498	872
15 Seeds	572	89	518	11	506	33
17 Seeds	552	3	499	7	-	-
19 Seeds	539	0	497	4	489	1

Figure 3: Mean Download Times At Different Bandwidth Restrictions

As mentioned, the efficacy of a solution is determined based on its download time relative to a given bitrate. The above times are successful if they are downloaded around, or under certain times - see Figure 4.

Quality	Bitrate	File Completion Time (s)
Low Def	400	1,260
Standard High Def	800	630
1080p High Def	1800	280

Figure 4: Completion Time Required for Immediate Streaming

The results in Figure 3 paint a generally clear picture, this solution usually works. Though we see in some cases, especially where few seeds are available,

there seems to be lots of contention for seeder bandwidth. When only 9 seeds or less are available we occasionally see extremely high variances. Though we argue that these variances are a function of our tests, and not the reality of our solution.

In a real scenario, someone would be monitoring the download waiting to start watching. If this sort of 'choking' or starvation behavior happened, a live user could simply tell the client to look for different peers. Further, in real life many trackers exist often with thousands of peer connections, which could allow for minimal congestion or 'starvation behavior'.

While it seems unlikely that 1080p streaming is an option with so few seeders, this may not emulate a popular show. In some of our testing we found that some recent and popular torrents had upwards of hundreds, and sometimes thousands of peers. Without further investigation we are unable to comment on it's possibility, but it seems likely that 1080p streaming would work. It should also be noted, that our results indicate that even when a client is part of a network with generally low bandwidths, a client with enough bandwidth can stream even standard high-def video.

Given our scenario we consider our implementation a success, in that almost all trials would have downloaded a standard high def video in the required amount of time. If we could use a client which uses rarest first, there would be further room to experiment and likely improve our final minutes of downloading.

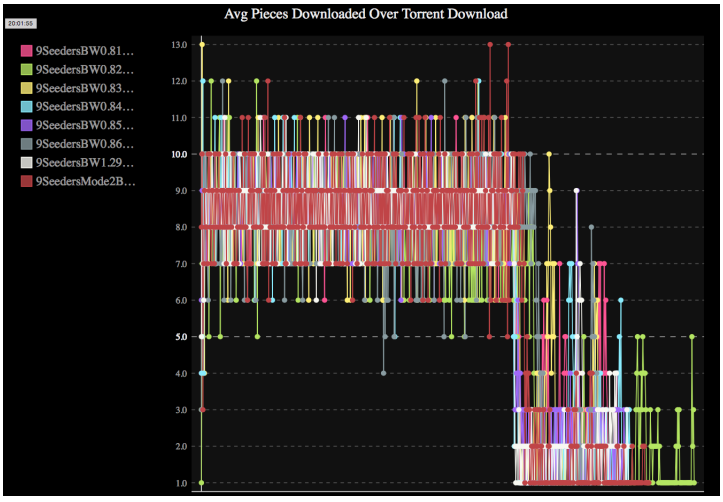

Figure 5: Trials with 9 Seeders at 0.8 Mbps - Each color is another trial, 9 in total were run

8 Threats and Challenges

Exploring hundreds of lines of code of Taipei Torrent was a challenge, since the task of locating and modifying the download scheme was not straightforward.

During our experiments, we noticed that the client was choked towards the end of the download which was quite problematic. One of the limitations of our

experiments was that we used a single tracker to locate peers while disabling the DHT and peer lookups. There was significant choke towards the end of the download, which can be seen in Figure 5. It shows the number of blocks downloaded per second, per trial over time. We chose this graph because it was representative of many of the rest of them, and as figure 3 shows that 9 seeders was right in the middle of efficacy.

We believe this significant drop in downloading could have been avoided by forcing an announce for new peers to continue download at an optimal rate. Furthermore, there was no force announce methods or new peers to add to the list which significantly slowed the speed at which we were downloading.

Due to the limitation of time, we repeated each run of the experiments between 10 and 15 times. For more accurate results we would have wanted to run this many more times. BitTorrent is a notoriously difficult protocol to experiment with since it has many variables that one must keep constant, however simulating a real swarm, churn and tracker behavior would be very difficult to create and evaluate.

9 Conclusion

Our experiments show that streaming is a viable option, especially when individual seed bandwidth is low. This is an effective method, given that it can be used in conjunction with normal torrent files. Further optimization on our strategy, and further exploration of other strategies is the next step, including experimentation with various size of torrent file. Creating a better automated framework to run these tests would be crucial in further research, and will enable researchers to get better averages and stronger results.

References

[1] Adobe.com. Table 1. recommended bit rates for live streaming, 2015.

[2] Y. Borghol, S. Ardon, N. Carlsson, and A. Mahanti. Toward efficient on-demand streaming with bittorrent. In *NETWORKING 2010*, pages 53–66. Springer, 2010.

[3] N. Carlsson and D. L. Eager. Peer-assisted on-demand streaming of stored media using bittorrent-like protocols. In *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pages 570–581. Springer, 2007.

[4] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[5] M. Großmann and S. J. Schuberth. Auto-mininet: Assessing the internet topology zoo in a software-defined network emulator.

[6] R. D. Jie Cheng. The pirate bay torrent analysis and visualization. In *IJCSET*, pages 38–42. IJCSET, 2013.

[7] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[8] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in bittorrent is cheap. In *Proc. Workshop on Hot Topics in Networks (HotNets)*, pages 85–90. Citeseer, 2006.

[9] J. Palevich. Taipei torrent, 2015.

[10] M. Piatek, C. Dixon, A. Krishnamurthy, and T. Anderson. Liveswarms: Adapting bittorrent for end host multicast. *University of Washington, Technical TR*, pages 11–01, 2006.

[11] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent. In *In NSDI'07*, 2007.

[12] G. I. P. Report. Sandvine internet usage report, 2015.

[13] P. Savolainen, N. Raatikainen, and S. Tarkoma. Windowing bittorrent for video-on-demand: Not all is lost with tit-for-tat. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–6. IEEE, 2008.

[14] P. Shah and J.-F. Paris. Peer-to-peer multimedia streaming using bittorrent. In *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*, pages 340–347. IEEE, 2007.

[15] C. Stais and G. Xylomenos. Realistic media streaming over bittorrent. In *Future Network & Mobile Summit (FutureNetw), 2012*, pages 1–8. IEEE, 2012.

[16] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[17] A. Vlavianos, M. Iliofotou, and M. Faloutsos. Bitos: Enhancing bittorrent for supporting streaming applications. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–6. IEEE, 2006.