Implementation of a BitTorrent protocol client for streaming purposes

Nenad Jovanovic, Milena Milosevic, Mladen Kovacev, Krsto Lazic, Nedeljko Babic
RT-RK Institute for Computer Based Systems
Novi Sad, Serbia

***Abstract*—This paper describes one solution for using BitTorrent protocol for streaming of multimedia content and an implementation on an Android set-top box. BitTorrent has established itself as a reliable and popular data sharing protocol, resilient and suitable for large scale traffic. With few improvements, solutions like this could potentially decrease costs for content providers, most of all for those providing audio and video content to the consumers.**

***Index Terms*—BitTorrent, multimedia, streaming, content sharing.**

I. Introduction

Over the years, BitTorrent has become one of the most popular P2P (peer to peer) protocols for data sharing [1]. Even though it is often connected with internet piracy and other copyright legal issues, the protocol is still very much in use. This is mostly due to its ease of setup, ease of use, scalability, low cost, and resistance to attacks.

However, in order to utilize the advantages of the protocol for streaming purposes, the traditional algorithms used with the protocol and traditional implementation weren't suitable. BitTorrent protocol was designed with fast data multiplication in mind, as well as prevention of "free riders". When it comes to streaming, priorities are different. The most important thing is to ensure enough bandwidth for multimedia of higher quality as well as to make sure the consumer gets the multimedia content data on time for reproduction so the reproduction can go as seamless as possible. In this paper we will present a solution that differs from the usual one, but meets these requirements with greater efficiency. In order to achieve this, we traded protocol's data multiplication speed for faster data providing and providing data as fast as possible. We will also suggest some of the ways in which shortcomings, created by this tradeoff, can be compensated.

II. BitTorrent protocol

In order to fully comprehend the rest of the paper, some understanding of the protocol is needed and some of its basics will be presented in this section.

BitTorrent protocol is, as previously mentioned, a P2P protocol. The data shared by the means of this protocol can range from very small (couple of megabytes) to very large (several gigabytes). The main principle rests on dividing all of the torrent data (shared data) to equal smaller pieces (they usually range from 512 KB to 4 MB). The pieces are then internally further divided into blocks, usually the size of 16 KB. The logical torrent data structure is shown in Fig. 1. The pieces are then distributed among peers (users involved in data sharing) through block requests. The blocks are requested by each peer according to its own needs and can be requested out of order [2]. This ensures that multiple pieces, as well as multiple blocks of the same piece, can be requested simultaneously from different peers so greater transfer speeds can be achieved. It also prevents the loss of the previously downloaded data if one of the parts fails to download successfully. Due to this characteristic of the protocol, every peer is a potential source for other peers as soon as it successfully downloads at least one piece [3]. In order for this to function properly, a simple algorithm is being used to determine which piece should be requested next. This algorithm is called "rarest first" and states that the pieces that are rarest in the swarm (all peers that share the torrent data make up the torrent swarm) have highest download priority. Thanks to this rule, pieces multiply quickly and situations when some of the pieces can't be found are very rare [4]. As efficient as it is in traditional use of the protocol (simple data sharing), this rule cannot stand when it comes to using BitTorrent for streaming purposes. When streaming, the priority of fast data multiplication comes a lot after assuring that consumers can watch the content as seamless as possible. This is why this algorithm has been changed in our implementation, which will be discussed later.

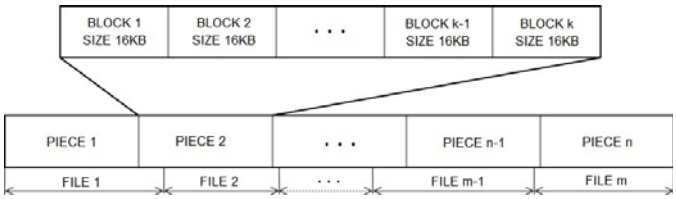

Fig. 1 Logical data structure of the torrent shared data

The second principle that BitTorrent protocol resides on are trackers. Trackers are servers that are used by peers for initial peer discovery. When entering a torrent swarm, every peer contacts the tracker in order to receive a list of peers already in the swarm [2]. After this initial contact, the tracker is no longer necessary and peers can mostly function without it until the end of the content download. This principle is responsible for protocol's resilience to attacks - if the tracker goes down, the

peers that were already in the swarm before the attack are not affected. Trackers use very little resources per torrent which makes them cheap. They can also host hundreds of torrents at a time without being overloaded. This is why content sharing using BitTorrent is very cost effective.

III. Concept and implementation

The main idea behind using BitTorrent for streaming purposes is adequately substituting "rarest first" algorithm with an algorithm appropriate for this purpose. Concept used in our solution is following. The pieces can be requested only from a sliding flexible window. A window is composed of maximum of N missing (not yet downloaded) pieces. The maximum number of missing pieces in the window is flexible. It can be increased in order to increase the download speed if the available bandwidth allows it, or it can be decreased to conserve memory if the available bandwidth has decreased during the download. We've also introduced piece priority based on a different criterion. The newly introduced priority level of the piece indicates from how many extra peers the blocks from the piece can be simultaneously requested. The reason for this is to acquire the prioritized data as soon as possible. Every piece is initially assigned priority level zero and only the piece that needs to be reproduced next is allowed to have priority higher than zero. This limitation minimizes the unnecessary extra traffic, but still allows effective data prioritization.

Thanks to these concepts, pieces next in line for reproduction will be downloaded as soon as possible while the transfer speed is still held as high as possible. One downside of the concepts are the extra traffic caused by simultaneous multiple requests of the same piece block. This problem is minimized with the before mentioned limitation as we will shoe in the testing results. Another downside is the fact that parts are not multiplied as efficiently as when using "rarest first" algorithm and swarms consisting only of our client application would possibly die out more quickly.

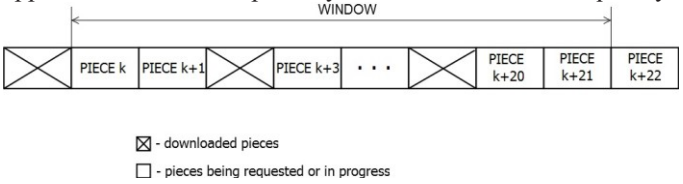

Fig. 2 Example of the moving window

To compensate piece multiplication inefficiency, content providers relying on our or similar solutions can combine BitTorrent content providing with traditional methods. A standard HTTP or FTP server with lower capacity can be established in order to help speed up initial forming of the swarm (so the first users wouldn't suffer for the sake of the rest) as well as when some of the pieces are extremely rare or missing. Due to some of the users interrupting streaming before the end, the rarest pieces would probably be the ones last in line for reproduction. The helping server doesn't need to take much load, because as the swarm gets bigger, peers will become sources themselves and help new peers. There are already two protocol extensions that enable realization of helper servers and are implemented in many of the online available clients.

A very simple playback control algorithm has been implemented. The downloaded data is buffered until the amount of successfully downloaded bytes surpasses both 30 MBs and 3% of the total file size. These conditions are necessary in order to compensate the limitations of certain platforms and multimedia players that need a certain amount of data before the reproduction can actually start. After these conditions are met, the playback is started. If the difference between the total amount of downloaded bytes and the total number of bytes used for reproduction drops below 1% of the total file size, the playback is paused so the player doesn't get into a situation where no data is available. The playback is resumed when the difference rises above 3% of the total file size. The algorithm is a simple one and even though it's improvement could greatly improve the overall user experience, it was enough to be able to test the application in the real world environment.

We have so far only managed to implement a client application with minimal functionality. It supports only the basic BitTorrent protocol with few necessary protocol extensions (mostly extensions for wider tracker support). Even with these minimal features, the realized client application has, nevertheless, handled itself well in swarms of peers with traditional clients. With some initial reproduction delay, the realized client application can provide seamless experience in "friendly" environments - swarms that are not in initial state and have enough bandwidth potential that surpasses the bitrate of the streamed content [5]. This means that our client application can be used for streaming content currently shared by traditional client application, even without the helper servers, although the usage is somewhat limited.

IV. Testing and results

In order to determine the quality of user experience (compared to the experience given by the standard streaming), three different parameters were monitored during the application testing. The parameters monitored are:

- the amount of time needed for the initial buffering before the playback can be started
- the total amount of time during which the playback was in paused state caused by the lack of downloaded data
- the total number of playback pauses (number of playback interruptions) caused by the lack of downloaded data

In this paper, the quality of user experience when using our application is considered the highest when the initial buffering time is below 180 seconds and when there were no playback interruptions.

The application was tested in swarms with different characteristics as well as with torrents with content of various length and bitrate in order to measure three different aspects of application's performance.

A. Comparing the minimum bandwidth needed by the application and the content bitrate

The first aspect measured was the difference between the minimum bandwidth needed by the application in order to achieve the highest quality and the content bitrate. This test consisted of multiple downloads of the same torrent but with different maximum download bandwidths. The video content of the torrent used in the test was 340 MBs in size, with length of 2597 seconds (43 minutes and 17 seconds) and had an average bitrate of 1072 kbps. The torrent swarm had a population of 1246 seeders and 25 leechers. The results of the tests are presented in table 1. All results presented are the average of three different test runs.

TABLE I. AVERAGE TEST RESULTS WHILE MEASURING THE DIFFERENCE BETWEEN THE MINIMUM BANDWIDTH AND THE CONTENT BITRATE

MAXIMUM BANDWIDTH	INITIAL BUFFERING TIME	TOTAL TIME IN PAUSED STATE	TOTAL NUMBER OF INTERRUPTIONS
1300 kbit/s	290,33 s	297,33 s	8
1500 kbit/s	222,66 s	147,33 s	2,66
1800 kbit/s	154,33 s	21,33 s	0.66
2000 kbit/s	120 s	0 s	0

The results show that the minimum bandwidth needed for the highest quality of user experience is significantly higher than the bitrate of the torrent content. This is most probably due to the stochastic behavior of the protocol itself. The protocol doesn't guarantee the delivery of data so the speed can vary greatly during the download. Every drop in download speed has to be compensated later with speed significantly greater than the content bitrate so the total average speed can be high enough for the seamless experience.

B. Behavior of application in different swarms

The next performance aspect measured was the behavior of application in swarms with different characteristics. Swarm characteristics considered relevant here are the swarm population (total number of peers, number of seeders and the number of leechers) as well as the seeders/leechers ratio. For this test four different torrents were downloaded with no bandwidth limitations with maximum bandwidth of 10 Mbps – the maximum allowed by our link. The characteristics of the torrents' content are described in the following table:

TABLE II. CHARACTERISTICS OF THE TORRENTS' CONTENT WHILE TESTING APPLICATION WITH DIFFERENT SWARMS

TOTAL CONTENT SIZE	CONTENT PLAYBACK LENGTH	AVERAGE BITRATE OF THE CONTENT
529 MB	1389 s (23 min i 9 s)	3120 kbit/s
354 MB	2554 s (42 min i 34 s)	1135 kbit/s
421 MB	2613 s (43 min i 33 s)	1320 kbit/s
148 MB	1431 s (23 min i 51 s)	847 kbit/s

The results and swarm characteristics of the torrents used in test are displayed in table 3. All results presented are the average of three different test runs.

TABLE III. AVERAGE TEST RESULTS WHILE OBSERVING APPLICATION PERFORMANCE IN SWARMS WITH DIFFERENT CHARACTERISTICS

SEEDERS / LEECHERS	INITIAL BUFFERING TIME	TOTAL TIME IN PAUSED STATE	TOTAL NUMBER OF INTERRUPTIONS
730/10	72,66 s	0	0
320/407	67,33 s	0	0
55/114	160	0	0
12/1	231,33 s	>1431	38,33

The results show that the application performed well with highest quality of user experience in most of the test runs. The only torrent, the content of which the application was unable to deliver on time, had a swarm with very low population, presumably with insufficient number of good sources to download from. It should be noted that content of the first torrent in the test runs was a video with full HD resolution with a bitrate of over 3 Mbps, meaning that the application can successfully handle even high quality content

C. Amount of overhead created by the application

The last aspect of performance observed was the amount of extra traffic created by the application in order to acquire the data on time for reproduction. During all of the tests, the amount of extra traffic created has never exceeded 2.5% of the total size of the file being downloaded, which can be considered acceptable for most use cases.

We should also note that, in high quality swarms, the application easily reached the maximum download speed allowed by the bandwidth.

The application was originally developed for and tested on PC Linux and later ported to Android and tested on an Android based set-top box.

V. CONCLUSION

Our solution represents a proof of concept and a base for further development. Our final goal is to develop and test a full provider - consumer environment and test the potential of the concept presented in this paper. The concept is intended for use in both environments with traditional clients as well as environments with provider support.

ACKNOWLEDGMENT

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia under Grant III-44009.

REFERENCES

[1] Hendrik Schulze, Klaus Mochalski, "Internet study 2008/2009", ipoque, Leipzig, Germany, 2009.

[2] Official specification of BitTorrent protocol version 1.0 and BitTorrent protocol extensions, http://www.bittorrent.org/, 2013.

[3] Bram Cohen, "Incentives build robustness in BitTorrent", Berkeley, USA, June 1963.

[4] Arnaud Legout, G. UrvoyKeller, P. Michiardi, "Rarest first and choke algorithms are enough", INRIA, Sophia Antipolis, September 2006.

[5] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garc´es-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime" PAM 2004, Antibes Juan-les-Pins, France, April 2004.