Enhancing Video Streaming Quality of Service through Peer-to-Peer File-Sharing

Carlo Izumiya, Herbert Hans Leong, Nickolai Cean Cecil Uy, Gabriel Enrique Yusay, Ryan Alex Chiu, Joshua Bernard Coralde and Arlyn Verina Ong*
De La Salle University
**Corresponding Author: arlyn.ong@dlsu.edu.ph*

Abstract: Peer to peer methods for video streaming can be used because they offer scalability by sharing the task and resource requirements for distributing content among several hosts, and network fault-tolerance by avoiding single points of failure. Among P2P file-sharing protocols, one of the most widely used is BitTorrent. BitTorrent is known for its capability to reach high download speeds due to its technique of sourcing file data from multiple peers simultaneously. Despite this advantage, it does not readily lend itself to supporting video streaming due to its approach of downloading data at various random locations in the file. Missing data dispersed in a video file may lead to jitter while streaming or video playback failure while the file is not complete. To address this issue, this research modifies the behavior of the BitTorrent protocol by implementing a pseudo-sequential downloading algorithm using a sliding window approach to suit the requirement for contiguous data to achieve quality video streaming. From the series of tests conducted, the proposed piece selection algorithm proved to be a viable approach for video streaming because it achieved download speeds that were still sufficient for streaming video while reducing playback issues associated with missing video data.

Key Words: Peer-to-peer; video streaming; BitTorrent

1. INTRODUCTION

In recent years, there has been an uptake in online learning approaches as alternative means to provide continuous education without the barriers brought about by physical distance. The need for this was further highlighted by the COVID-19 pandemic which forced the suspension of face-to-face classes all over the world. One of the popular methods employed in distance learning is the use of educational video streaming to provide instructional content to learners.(Hartstell, 2001).

Video streaming allows a user to constantly receive multimedia data from a provider and video files to be played even before the entirety of the file has been downloaded. This is an alternative to traditionally downloaded files, which require the entire video data to already be acquired before the content could be watched or played. Due to this, there are a myriad of media sites from the likes of YouTube, Coursera, TED, and plenty more that use or offer video streaming as a means of presenting multimedia content. Video streaming has two approaches for downloading multimedia content: client-server-based streaming, and peer-to-peer based streaming.

Client–server-based multimedia streaming is an approach widely used on the Internet. Each client requests and obtains the multimedia file directly from a streaming server; the server is then responsible for managing and allocating resources for streaming data requests from its clients (Wu et al, 2011). With this

approach, the server must be able to manage a heavy load from multiple requests, which requires significant physical and network resources to sustain quality service and acceptable viewing experience for the audiences as the number of users increase. Furthermore, the server represents a single point of failure which may cause the unavailability of the streaming service should it encounter any issues. For educational institutions with limited resources, a robust and scalable alternative means of providing streaming video content to a large user base may potentially be a workable solution to still provide video content despite this limitation.

Peer-to-Peer or P2P based streaming is one approach that is scalable and capable of meeting the need for Video-on-Demand service (Nurminen et al, 2013). In the P2P approach, participating nodes may take the role of a server and a client simultaneously, allowing streaming clients to also provide the content they download from the server to their peers. The protocol algorithm balances the load of data and manages the movement of data from one peer to another. This method allows participating nodes to offload the originating server, and at the same time reduce the risk of a single point of failure as copies of the same content may be sourced from different nodes in the network (Schollmeier, 2001).

A widely used P2P content-sharing protocol is BitTorrent. It functions by organizing a swarm of peer nodes that each hold full or partial copies of files available for download. A client wishing to obtain a copy of a file may join the swarm as a peer and simultaneously download segments of the file from different peers and piece these together to form the complete file when done (Cohen, 2008). Currently however, BitTorrent peers download file data in a random manner. This approach does not lend itself well to video streaming which requires video data to be acquired in a sequential manner so that playback may begin immediately and proceed continuously even while the video file has not yet been downloaded in its entirety. To harness the advantages of BitTorrent as a widely available and scalable P2P protocol for video streaming, its algorithm must be modified to perform sequential data download so that users may begin watching a video without interruption even while the acquisition of its segments is still ongoing.

This paper presents a P2P system which uses a modified BitTorrent protocol to enable sequential data download to support MPEG video streaming. Part 2 provides an overview of the BitTorrent protocol and its design elements that limit its viability for video streaming. Part 3 discusses the design of the modified protocol and part 4 discusses results of testing it on a video streaming application prototype. Finally, part 5 identifies the findings and conclusions drawn from this study.

2. THE BITTORRENT PROTOCOL

The BitTorrent protocol functions using an architecture that centers around a tracker server. It keeps track of peers called seeders that have the data of a file available for download. When a file is to be made available for download through BitTorrent, it is initially seeded by a primary server which segments the file into fixed-size chunks called pieces that peers may download. A torrent file is created for it containing its metadata such as tracker and other file information (Cohen, 2008).

A client that wishes to download the file must have a copy of the corresponding torrent to be able to locate the tracker which will then provide the list of peers that are seeding the file. With this list, the client can join the P2P swarm and connect to seeder peers to begin downloading pieces of the file until all pieces are acquired. Pieces of the same file may come from different seeders in the swarm and are not necessarily collected in the same order as their sequence within the file. As the client accumulates pieces of the file, it may also begin seeding the file itself to further add to the sources of the file data within the P2P swarm. As more peers obtain copies of file pieces and seed these to the swarm, the potential data sources increase, thereby also increasing the speed at which the file may be downloaded by a requesting peer.

Using this protocol design, BitTorrent is not yet well-suited for video streaming. Video data is organized such that it also generally follows the same sequence at which it will be played back. For video to be streamed, data needs to be acquired in sequence so that a video may be continuously played back even before its download is completed. Using the distributed manner by which BitTorrent acquires file pieces, a downloading peer may receive data at latter parts of the video file not yet needed for current viewing. With this, it is possible that playback cannot begin even if some data is already available. Likewise, user viewing may be interrupted or severely degraded if playback reaches a point in the file where data is still missing. At the same time, downloading pieces in a strictly sequential manner defeats the speed advantage that BitTorrent provides by allowing a client to download pieces from multiple sources simultaneously.

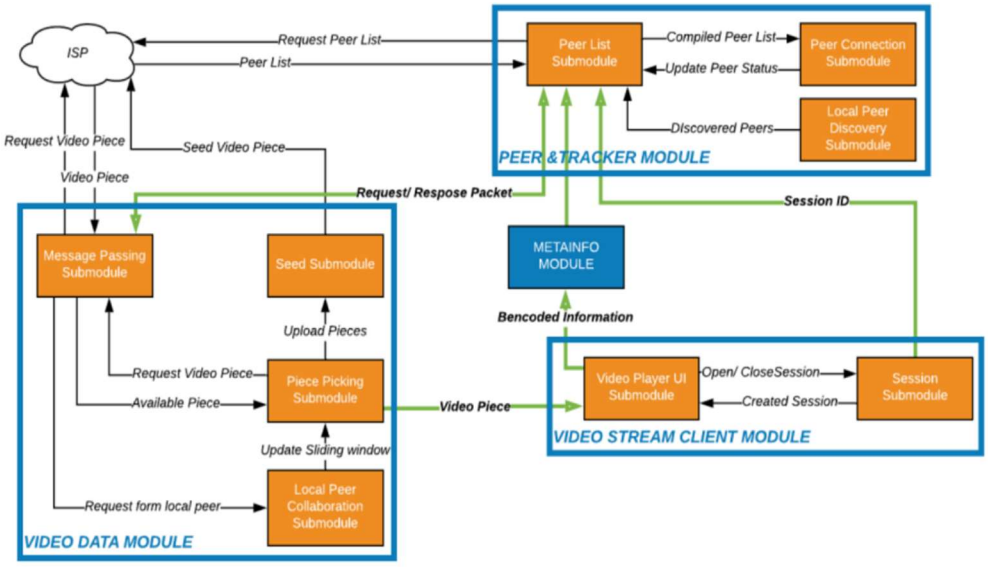

Fig. 1. Architectural diagram of P2P Video Streaming Client Software

3. P2P VIDEO STREAMING SYSTEM DESIGN

To balance the need for contiguous data download while efficiently using piece availability within the BitTorrent swarm, the proposed approach employs a pseudo-sequential piece selection algorithm that prioritizes the acquisition of video file pieces needed for immediate playback while still allowing a client to download pieces for future playback when the opportunity allows. The researchers implemented the algorithm as a prototype peer-to-peer (P2P) file downloading application with basic video playback functionality. The Libtorrent application programming interface (API) (Norberg, 2016) is used to provide the standard BitTorrent functions for the program, with the modifications focusing on the addition of local network peer sourcing and video file piece-picking procedure. Fig. 1 illustrates the architectural design of the prototype application.

3.1 Metainfo Module

The Metainfo Module performs the decoding of torrent metadata in order to determine the torrent tracker, file length, piece information and piece hash value needed for data verification when pieces are downloaded. The metadata extracted from the torrent will then be passed on to other modules to perform the actual connection to peers and download of file pieces. This module is invoked and will serve as the starting point for the operation of the application once the user selects a video torrent file for streaming.

3.2 Peer and Tracker Module

The Peer and Tracker Module forms the peer swarm from both Internet and local network hosts, initiates the connection to peers to begin file download, and monitors peer status while data transfer in ongoing. It begins by using its Peer List submodule to acquire the list of Internet peers that are seeding the file from the tracker.

To maximize the possible sources of the file data, a Local Peer Discovery submodule actively discovers clients within the local network that may also be streaming the same file. This submodule is necessary given the limitation of the BitTorrent protocol in locating and differentiating potential peers within the same network if they are using a private IP address behind a gateway that is using port address translation. To do so, the submodule employs the Local Service Discovery Protocol extension of BitTorrent (BitTorrent.org, 2015). Any local peers discovered are added to the peer list.

From the combined list of Internet and local peers, the Peer Connection Submodule then initiates the connections to torrent peers and regularly uses keepalives to monitor peer status. This monitoring information is used to keep the peer list updated in terms of determining viable peers to source file pieces.

3.3 Video Data Module

The Video Data Module manages the selection and download of file pieces. Its Message Passing submodule uses standard BitTorrent protocol control messages through the Libtorrent API in communicating with torrent peers to transmit and receive messages that communicate availability of file pieces, signal interest in a piece currently available on a peer, request for a specific piece, transfer actual piece content and cancel a piece request.

The Piece Selection Submodule determines which pieces are available and which pieces to request from torrent peers. To support video streaming, it implements a sliding window algorithm that designates a block of contiguous critical pieces that must be prioritized for download to sustain continuous video playback. This window is then automatically advanced through the sequence of file pieces as critical pieces are downloaded.

The size of window is set to the equivalent number of pieces needed to play 20 seconds of video. This is calculated based on Equation 1 using 256kb as the torrent piece-size.

$$W = (B / 256\ kb) * 20\ secs \quad \text{(Eq. 1)}$$

where:

W = Window size in number of file pieces

B = Video bitrate in kilobits per second

The window size is also the basis for grouping video pieces into blocks. When streaming begins, the window starts from the first block of the video file. Pieces of the block within the window are given high priority and a download deadline so that these are prioritized for request by the LibTorrent API using the default rarest first scheme from seeders. Fig. 2 illustrates this concept where the download window is currently positioned on the first block (B_0) and pieces within it ($P_{0.0} – P_{0.n}$) will be downloaded first.

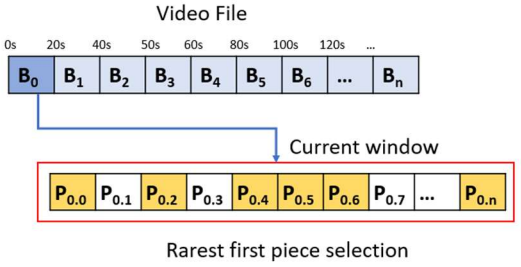

Fig. 2. Sliding Window for Piece Prioritization

A timer is also set for movement of the window to the succeeding download block so that downloading of succeeding video file pieces may still proceed even if a critical piece is difficult to source within the swarm. The download window slides to the next video block when either of the following conditions is met: (1) all pieces belonging to the block have been downloaded or (2) the block timer has expired. In the latter condition, the unfinished pieces in the block remain in high priority status even when the window moves to the next video block. These are coordinated with the Message Passing Submodule so that the appropriate LibTorrent API calls can be made to control BitTorrent protocol operations.

The Peer Collaboration submodule assists in downloading and sharing video file pieces within the local network if local peers streaming the same file are available. If a peer determines that it has streamed pieces ahead of those currently needed by another local peer, it directly shares these pieces to the local peer so that the pieces need not be requested from peers on the Internet.

To allow the application to contribute to the swarm, the pieces that it has downloaded are made available to other peers using the Seed Submodule.

3.4 Video Stream Client Module

The Video Stream Client Module serves as the interface to the user for basic application control and video playback.

The Session Submodule begins the streaming session by handling the peer-to-peer protocols initiating from the streaming client where the user requests a media file to be played; while the Video Player serves as the graphical user interface which allows the user to select a video file to stream and watch the decoded video data. The video stream is fed to an installed VLC media player which performs the actual decoding and playback of video data. In the event that the video playback reaches a missing piece in the file, the VLC client is capable of continuing the playback albeit with some video degradation so long as the missing data is within tolerable limits.

4. RESULTS AND DISCUSSION

To evaluate the performance of the proposed approach, the streaming application was tested on two local network hosts with Internet access acting as peers and compared against a P2P client using standard BitTorrent protocol.

The video file used for the sequential mapping and download speed tests had a file size of 128.8 Mb with a length of 16 minutes and 21 seconds. At the time of testing, the file had approximately 139 peers available on the Internet. The purpose of the tests performed was to determine the viability of the proposed approach for streaming and to identify potential performance tradeoffs.

4.1 Sequential Mapping Test

The sequential mapping test aims to observe the behavior of the protocol in downloading file pieces using the sliding window piece picking algorithm.

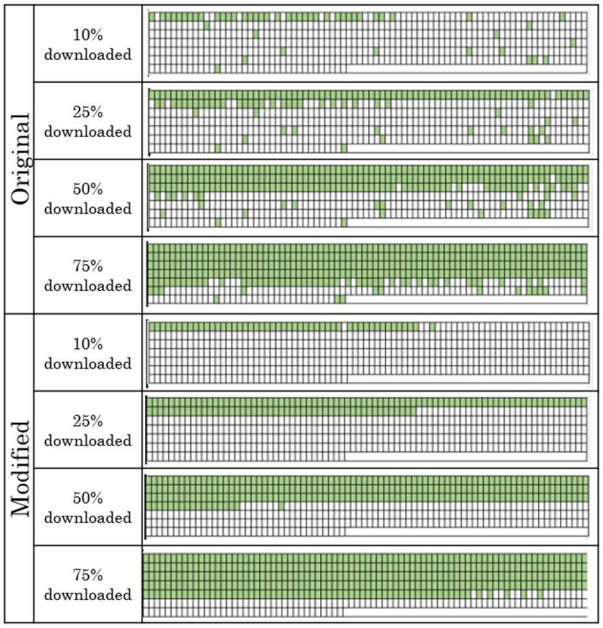

Fig. 3. Dispersion of Downloaded Pieces

Fig. 3 shows a map of file pieces acquired by a client using the original non-sequential BitTorrent algorithm compared against the sliding window approach as file download progresses. Green boxes represent downloaded pieces, while white ones represent pieces that are still pending download.

It was observed that there is a noticeable difference in the progression of downloaded pieces. The original BitTorrent piece selection algorithm follows a near-sequential downloading scheme of pieces; however, it still has the tendency to download pieces at more dispersed locations in the file that are not needed until a later time in the video playback. As the file is played back, there will be a higher likelihood of encountering a missing piece which can degrade the video or halt its playback entirely. Conversely, the sliding window algorithm tends to gather pieces that are contiguous and in-sequence. This will result to a better chance of uninterrupted video playback even in the early stages of file download.

4.2 Download Speed Test

A simple comparison of performance can be measured by measuring the speed at which file download can be completed by both approaches. In this test, three scenarios are compared: original BitTorrent (non-sequential), Sliding Window approach (sequential) without local peers, and Sliding Window approach with 1 local network peer already having 50% of the file available. Each scenario is tested 5 times. Averaged results are shown in Fig. 4.

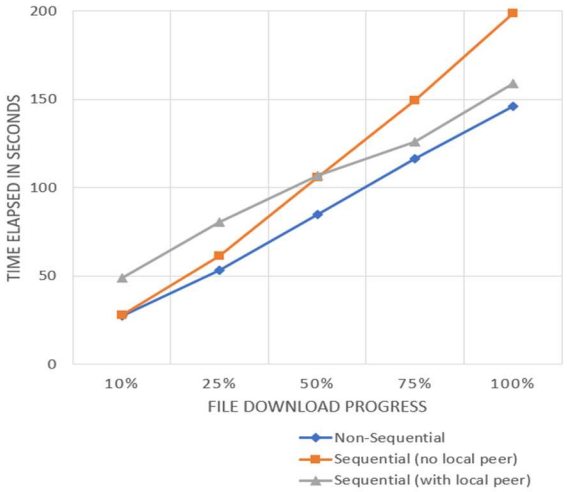

Fig. 4. Comparison of File Download Time

Results indicate that the download duration for the same amount of data is longer when the sliding window algorithm is used even if a local peer is available when compared against the original BitTorrent protocol. This is expected as the modified algorithm forces the client application to download pieces within the current window position even if other pieces are readily available. This can increase the download time when delays are encountered in acquiring pieces currently being requested from peers.

By comparison, the original BitTorrent piece picking algorithm uses available bandwidth more efficiently by allowing the client to quickly download readily available pieces at various locations in the file. It can be noted, however, that if local peers are present

and have available pieces for sharing by the time a client begins streaming, the total elapsed time to download a video file becomes more comparable to the original BitTorrent implementation due to significantly faster local network speeds compared to Internet connections.

4.3 Video Streaming Test

To determine the effect of the modified algorithm on the actual video streaming experience, trials were conducted to measure video playback statistics. These include the number of video frames successfully decoded and displayed, number of lost data blocks, the total time elapsed from when the user initiated the streaming to the time that the video began actual playback, and the cumulative amount of time that playback was temporarily paused to recover from missing data within the first 10 minutes of playback. MKV and MPEG-4 files were selected for this test, being common formats for videos. Table 1 shows these results.

Table 1. Video playback statistics

	Trial 1: MKV		**Trial 2: MPEG-4**	
Seeders	2177		72	
File size	215 MB		951 MB	
Length	58 mins.		2 hrs 3 mins.	
Ave. Video Bitrate	1002 kbps		929 kbs	
	Original	**Modified**	**Original**	**Modified**
Decoded	1035	14370	0	14203
Displayed	1050	14197	0	14160
Lost Blocks	13	195	n/a	166
Start Time	120 secs	43 secs	n/a	96 secs
Total Pause	3600 secs	0 secs	n/a	0 secs

Based on results, the trials using the modified algorithm was able to successfully decode and display more frames that their unmodified counterparts. In Trial 1 using the original algorithm, the user experienced a longer waiting time before the piece containing the starting data of the video was downloaded and playback could begin, after which the playback could no longer continue after the first 1050 frames until the time that the entire video finished downloading. Results were even less favorable in Trial 2 as the video could begin playing only after the entire video was downloaded.

In contrast, in tests where the modified algorithm was used, playback began shortly after the download was started and continued without pausing until the end of the test. It should be noted however, that missing data was still encountered during playback using the revised algorithm, hence some video distortion was experienced although these were tolerable enough for the video player to continue the playback.

5. CONCLUSION

Based on the results conducted, it can be concluded that using a pseudo sequential piece picking algorithm allows BitTorrent P2P file sharing a viable alternative to a client-server based video streaming service. The modification achieves a near sequential download of video file pieces, allowing playback to begin even while download in ongoing. It results in a tradeoff in download speed compared to the original BitTorrent algorithm but is still sufficient to perform continuous playback albeit with video degradation in instances when few seeder peers are available.

6. REFERENCES

Hartsell, T., Yuen, S. & Yuen, Y. (2006). Video streaming in online learning. AACE Journal. 14. 31-43.

Wu, D., et al. (2001). Streaming video over the Internet: approaches and directions [Electronic version]. IEEE Transactions on Circuits and Systems for Video Technology, 11, 282-300.

Nurminen, J., et al (2013). P2P media streaming with HTML5 and WebRTC [Electronic version] 2013 IEEE Conference on Computer Communications Workshops, 2013, 63-64

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications [Electronic version]. Proceedings First International Conference on Peer-to-Peer Computing

Cohen, B. (2008). The BitTorrent protocol specification. Retrieved March 13, 2021 from https://www.bittorrent.org/beps/bep_0003.html

BitTorrent.org. (2015). Local Service Discovery. Retrieved Oct 26, 2018 from http://bittorrent.org/beps/bep_0014.html

Norberg, A. (2016) LibTorrent. Retrieved March 14, 2021 from http://www.libtorrent.org/