VNU Journal of Science: Comp. Science & Com. Eng., Vol. 32, No. 1 (2016) 33–41

# Reducing Startup Time in MP4 On-demand Video Streaming Services with Movie Atom Caching

Xuan Tung Hoang*, Tien Thanh Nguyen

*VNU University of Engineering and Technology, Hanoi, Vietnam*

**Abstract**

This paper points out negative effects on quality of experience of video streaming sessions caused by metadata atom in MP4 movie files. Based on experiments, it is shown that the duration for downloading metadata atom could be relatively large for high-quality full-length movie videos. This leads to noticeable and disturbing startup delay to users when watching MP4 movies online. According to our model of user behavior, such a long startup delay could result in a large number of "leaving users" who abandon their video streaming before videos start to play. In order to reduce the startup delay and the portion of users who leave video sessions early, we present a mechanism, called Movie Atom Caching, that reuses previously downloaded metadata atoms or proactively downloads and caches movie metadata atoms at video players before users actually play the video. The mechanism is implemented in our video streaming prototype system. Experiments on the system show that, in typical cases, user experience is significantly improved as startup delay is cut down.

Received 05 December 2015, revised 22 December 2015, accepted 31 December 2015

*Keywords:* Multimedia Streaming, MPEG-4, MP4, User Experience.

## 1. Introduction

MPEG-4 part 14, or MP4, is currently one of the most popular container formats for video contents because of it flexibility and extensibility in combining different timed media information into one compact file format. MP4 is currently used as the standard video format in modern smart phones, handheld computing devices, and video capture devices. Streaming MP4 videos over HTTP is also possible just by simply hosting *relocated MP4 files* in web servers [1, 2]. Such a simple streaming mechanism allows easy-to-deploy streaming services, e.g. Youtube or Facebook Video, and lead to popularity of MP4 streaming application on the current internet.

According to [3], an MP4 stream is a hierarchical structure of data units called atoms (or boxes). In general, a video can be decomposed into two parts, multimedia data and metadata. The former, real multimedia data, is contained in *mdat* atom. The latter, metadata, is stored in *moov* atom which in turn contains smaller atoms such as *trak*, *stsd*, *stss*, *stts*, *stsz* that are important for parsing and decoding data in *mdat*. As a result, playback of an MP4 stream can be started only after moov atom is successfully received. This leads to a playback startup time and the size of moov atom is crucial factor of the startup delay.

The size of moov depends on a number of parameters including frame rate, rate of I-frame, and duration of video file. Table 1 summarizes measurements on bit rate and moov atom size of sample mp4 files. The first file is the original one, and other files are processed from the first one. Particularly, the second file is a scale-down version of the original one; the third file and the forth file, respectively, are the first half and the first quarter in time duration of the second file.

* Corresponding author. Email: tunghx@vnu.edu.vn

Table 1: Size of moov atom of sample video files

| File index | Duration (hh:mm) | Frame size ($W \times H$) | bit rate (Kbps) | moov size (KB) |
|---|---|---|---|---|
| 1 | 2:10 | $1280 \times 536$ | 2935 | 4167 |
| 2 | 2:10 | $720 \times 301$ | 917 | 4027 |
| 3 | 1:05 | $720 \times 301$ | 917 | 2022 |
| 4 | 00:33 | $720 \times 301$ | 917 | 1028 |

Other parameters such as frame rate and rate of I-frames are identical for all the files. Specifically, frame rate is set at typical values of 25 frames per second and I-frame rate is 1 I-frame for every 50 frames.

Results in Table 1 reveal that while startup delay, due to downloading moov atom, could be negligible for short video clips, the startup time is noticeable in long videos and could have bad effects on user experience. For example, when each file in Table 1 is downloaded over an optimized network connection whose throughput is approximately equal stream's bit rate, the time periods for downloading corresponding moov atoms are 11 seconds, 35 seconds, 18 seconds, and 9 seconds, respectively. As presented in [4, 5], such long startup delays have negative impact on viewers convenience and is considered as poor quality of experience.

In this paper, we derive a model that captures characteristics of impatience of users who watch video-on-demand service on the internet. Such a model is useful in finding out percentage of users who quits waiting for videos because of long startup delay. To alliviate bad effects of startup delay, a simple mechanism called Movie Atom Caching, or MAC, is presented. MAC simply caches moov atoms of MP4 files that it knows, or even prefetches them, for being able to start video playback without download moov atoms from servers. Such a simple mechanism is relatively simple to implement yet very efficient in reducing startup time of MP4 streaming. In our implementation, at client side, we integrate MAC mechanism into FFplay, an opensource video player in FFmpeg tool suite [6]. At server side, a daemon application runing in background will automatically split moov atom headers from the MP4 files for ease of prefetching and caching. There could be other choices for implementing MAC in real system such as developing plugins for other opensource video player like VLC [7], and integrate server side components with a web application stack (for example, LAMP [8]). Thanks to its simplicity, developing MAC in that way will be equally easy as hosting MP4 files on web servers. The mechanism, MAC, can be deployed in Linux server (CentOS 6.x) and both Linux and Windows clients. It is benchmarked against other streaming methods including progressive download [2] and HLS [9]. Our initial experimental results show that MAC can greatly improve user experience and startup much faster than other protocols.

This paper is organized as follows:

- In Section 2, we analyse effects of startup delay based on modeled user behaviors and pattern of user requests.
- Implementation of our proposal for solving startup delay problem, Movie Atom Caching, is presented in Section 3.
- Performance evaluations are presented in Section 4 to show efficiency of our MAC mechanism.
- We place our work in the context of related work in Section 5 and discuss our outstanding issues and future works together with conclusions in Section 6.

## 2. Effects of startup delays

### *2.1. System and user behavior models*

We consider a system in which there is a single server hosting a MP4 movie. Users' requests for viewing the MP4 file come to the server sequentially according to a random process. When a user request is served, it follows the state

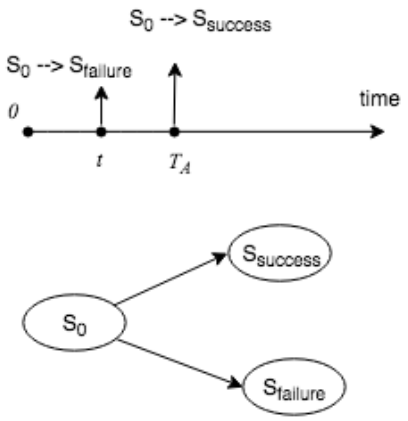


Fig. 1: User behavior state diagram.

diagram shown in Fig. 1. Particularly, after a request reaches the server, the user enters state $S_0$ at which the movie atom is fetched. From $S_0$, the user either moves to state $S_{success}$, at which he has successfully retrieved the atom and can retrieve movie frames for playing, or moves to state $S_{failure}$ if user gets impatient and quits from waiting for atom.

Illustration shown in Fig. 1 describes that behavior of users in the system model. At time 0, when a user request is processed, the server starts sending moov atom header of the MP4 file to client application. This will take a period of $T_A$ for the atom to be completely sent. If a user waits for the atom up to *atom download time, or startup delay* $T_A$, he will move from state $S_0$ to state $S_{success}$ and stays there while enjoying video playback. If a user, because of impatient, stops waiting for the atom after *waiting time* $t$, $t < T_A$, he will move from $S_0$ to $S_{failure}$. The fraction of users having such "early quiting" cases is called streaming failure rate, $F$, and it is directly influented by startup delay $T_A$. In other words, $F$ can be used to evaluate bad effects of startup delay on user experience and system performance.

Let waiting time is a random variable whose probability density function (pdf) is $p_w(t)$. $p_w(t)$ is probability that a user quits waiting for atom at time $t$ since he starts download movie atom. That probability density function can be used to model impatience of user according to the following observations:

- If $p_w(t)$ is ascending, then user is patient since when $t$ is small, the user is not as easy to quit as he is when $t$ is high.
- Similarly, descending $p_w(t)$ means user is impatient.
- The more slowly $p_w(t)$ increases, the more patient user is.
- The more rapidly $p_w(t)$ decreases, the more impatient user is.

Modeling user behavior with function $p_w(t)$ can help in quantitatively pointing out how reducing startup delay can help in decreasing percentage of users who quit waiting for videos during startup delay. In the following subsections we investigate several models of user impatience $p_w(t)$ and figure out streaming failure rate $F$ accordingly.

### 2.2. Simple model of user behavior

As a simple model that captures user behaviors, ones can let the waiting time $t$ an exponential random variable with mean $T_0$ seconds. That is

$$p_w(t; T_0) = \frac{1}{T_0} e^{-\frac{t}{T_0}}, t \geq 0 \quad (1)$$

Here $T_0$ is a parameter that captures user behavior and $T_A$ represents for system characteristics that is independent of users behavior. The streaming failure rate due to startup delays, $F_{simple}$ is:

$$\begin{aligned} F_{simple} &= Pr\{t < T_A\} & (2)\\ &= \int_0^{T_A} p_w(t; T_0) dt & (3)\\ &= 1 - e^{-\frac{T_A}{T_0}} & (4) \end{aligned}$$

User impatience model described as an exponential distribution is simple but may not be sufficiently good for modeling user behavior in reality because of the following reasons:

- It only captures the case in which users are relatively impatient since $p_w(t;T_0) = \frac{1}{T_0}e^{-\frac{t}{T_0}}$ is a quickly descending function. Such a function is more suitable for modeling users who just click on movies that they encounter and do not intentionally watch movies that they like.
- It lacks of capability in parameterizing user impatience.

### 2.3. Parameterizing user impatience

A better model in capturing how impatient a user is can be developed from the following assumptions. Firstly, a user waits for video to start up to a threshold $T_0$. If the waiting time reaches $T_0$, the user does not wait any longer and he aborts his session. Secondly, probability distribution function $p_w(t)$ of waiting time $t$ increases as $t$ increases. An typical behavior of such characteristics is described as follows. At the beginning, a user tries to wait for video to start up to the threshold $T_0$. Probability that he aborts his session at the early stage is rather small. However, the longer he waits, the less patient he becomes. And when his waiting time is up to $T_0$, he definitely aborts his video streaming session.

We define probability distribution function of waiting time $t$ for the above user behavior as belows:

$$p_w(t;k;T_0) = \begin{cases} \frac{(k+1)t^k}{T_0^{k+1}}, & t \le T_0 \\ 0, & t > T_0 \end{cases} \quad (5)$$

Here, $k$ and $T_0$ are two parameters such that, $k \in \mathbb{Z}, k \ge 0$, and $T_0 > 0$. Parameter $k$ can be used to characterize *level of patience* of users or *patience factor*. The higher value of $k$ is set, the more patient users we have. When $k = 0$, $p_q(t) = \frac{1}{T_0}$. It means waiting time is uniformly distributed in $[0, T_0)$, and user's patience is neutral.

Let us consider relationships between waiting threshold $T_0$ and startup delay $T_A$ in this model. If $T_A > T_0$, we can say that startup delay $T_A$ is too long and no users are sufficiently patient to wait for video playback to start. If $T_0 \ge T_A$, only a fraction of users whose waiting time periods are less than $T_A$ is moved to state $S_{failure}$. Thus the streaming failure rate for this model, $F_{patient}$, can be calculated as:

$$\begin{aligned} F_{patient} &= Pr\{t < T_A\} & (6)\\ &= \int_0^{T_A} p_w(t;k,T_0)dt & (7)\\ &= \frac{T_A^{k+1}}{T_0^{k+1}} & (8) \end{aligned}$$

Let $\alpha = \frac{T_A}{T_0}$, equations 4 and 8 become:

$$\begin{aligned} F_{simple} &= 1 - e^{-\frac{T_A}{T_0}} = 1 - e^{-\alpha} & (9)\\ F_{patient} &= \left(\frac{T_A}{T_0}\right)^{k+1} = \alpha^{k+1} & (10) \end{aligned}$$

Figure 2 and figure 3 show streaming failure rate $F$ as function of $\alpha$ in simple user behavior model and in model that captures user patience. Those plots can be used to visualize dependency of streaming failure rate to startup delay for a specific waiting time threshold $T_0$.

## 3. Movie Atom Caching

Startup delay can be easily reduced by a simple caching/prefetching mechanism, called Movie Atom Caching (MAC), as described by flowchart in fig. 4. A client application, MAC client, first checks whether it has moov atom of a requested MP4 movie locally. If yes, the client immediately downloads multimedia data from mdat atom of the requested file. Thus, the video playback can be started almost immediately and startup delay will be greatly reduced. If no, the MAC client should behave similarly to a normal player by getting moov atom from server. However, it will completely fetch the moov atom even when the user impatiently aborts his video streaming session. The moov atom after being successfully downloaded will be stored locally at MAC client for the successive video requests. All MAC protocols' operations work on top of the typical HTTP protocol.

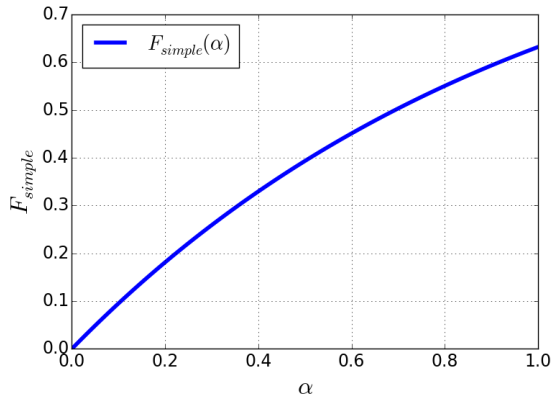


Fig. 2: Streaming failure rates in simple user behavior model.

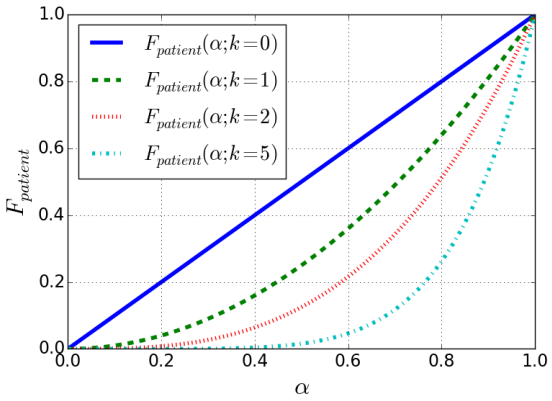


Fig. 3: Streaming failure rates in patient users model.

Although the algorithm shown in figure 4 only mentions an on-demand caching algorithm, it can be easily extended into a proactive prefetching mechanism in which an in-background application at client side can be deployed to download moov atoms of files that are likely to play by the user. Such files can be obtained by using a recommendation system that belong to content management system of a video site, for example.

Although MAC algorithm is very simple, implementing it into good working software applications needs following requirements:

- First of all, MAC should be extensions or add-on components of existing solutions for streaming MP4 files. Development of a new streaming server software is considered a bad design choice because of high implementation costs.
- The streaming service for MAC protocol should be compatible with non-MAC clients. Again, developing a completely new streaming server software is not a good idea because of potential incompatibility with existing clients.
- Finally, with similar reasons as above, ones should not develop a new video player for MAC. Insteads, plugins should be developed to integrate with existing video players.

Our implementation for MAC protocol is shown in figure 5. At the beginning, MP4 files are uploaded to ”publishing area” on server. For uploading MP4 files, any method that can serve this purpose, e.g. FTP, scp, or rsync, can be used. A daemon application, MAC daemon, will monitor the publishing area for new files and automatically split each new file into two files containing moov atom and mdat atom respectively. As examplified in figure 5, v1.atom and v1.mdat are the two files splitted from the original file v1.mp4. Those files will be moved to ”public area” and become ready for streaming.

When a MAC client is commanded to view a MP4 video, it will check for an apropriate atom file in its local video cache. If such a file is found, it will be used as the atom header for parsing multimedia data retrieved from server. In case MAC client has to download atom file from server, the successfully downloaded atom file will also stored into the local video cache.

At server side, existing HTTP-based solutions for streaming atom and mdat files of MP4 movies are used. Specifically, files in public area and publishing area are stream by typical webserver like nginx [10]. This can be done simply by configuring those locations as accessible resources to the webserver and the web application hosted by the webserver. Additionaly, a simple web application developed in PHP is deployed on the webserver for handling both MAC non-MAC (normal HTTP) MP4 video

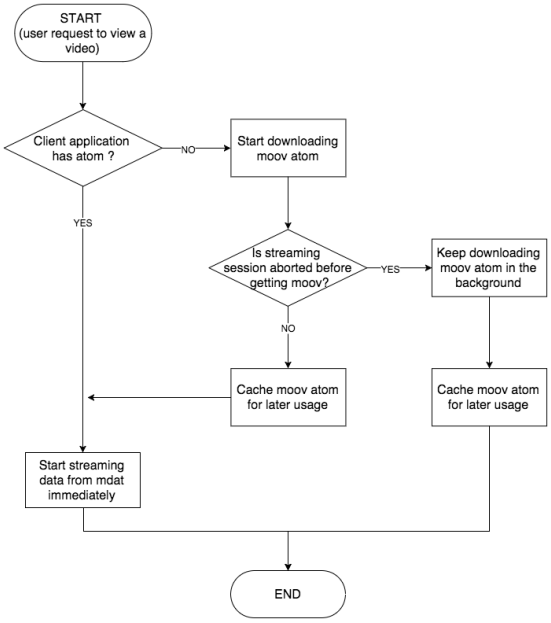


Fig. 4: Atom Caching/prefetching algorithm for startup delay reduction.

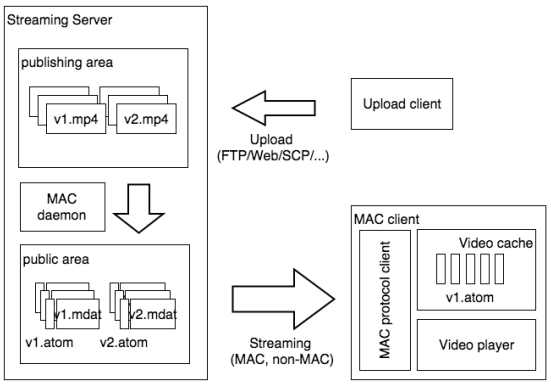


Fig. 5: Movie Atom Caching implementation.

streaming. The web application will conceal atom and mdat files under reference to original MP4 files and provide a layer for providing compatibility with both streaming protocols.

Our MAC client is implemented using FFmpeg [6]. Specifically, we developed a protocol plugin for FFmpeg. Our plugin simply interfaces with streaming service and local video cache for handling atom retrieval, caching and combining with mdat data from server. Thanks to high portability of FFmpeg, our MAC implementation can be built for both Windows and Linux platforms.

## 4. Performance Evaluation

In this section, we present our setups for comparison startup delays between MAC and other protocols including progressive download, and HLS. In our experiments, the following elements are deployed:

- A server machine on which: an FTP service (vsftpd) is running for allowing MP4 file uploads. An instance nginx webserver with supports for PHP 5 over fastCGI [11] are launched for providing HTTP streaming capabilities. And MAC daemon application are running for MAC protocol file preprocessing. All thosecomponents are run on a virtual 32-bit CentOS 6.x machine.
- A client machine on which: required player and its dependencies are installed. Those software components include ffmpeg library and MAC protocol plugin. All of them run on a Ubuntu 14.04 LTS 32-bit virtual machine.
- And a virtual network setup by GNS3 simulator [12]. In our setup, the virtual network contains only a virtual Cisco 2691 router and two virtual links connecting the router with the server and client machines above. Such a simple virtual network is sufficient for us to create different values of end-to-end bandwidth between the client machine and the server machine.

Our experiments for measuring startup delay are conducted as follows: Startup delays are measured on 3 sample videos with the same settings of bitrate, frame-rate, I-frame rate, and other parameters related to audio streams. Table 2 summarizes those parameters. The three sample movies have durations, respectively, 30, 60, and 90 minutes. The sample videos are used for streaming with MAC, HLS, and progressive download protocol under end-to-end bandwidths of 512 Kbps and 2 Mbps. We believe that 2 Mbps is the typical access bandwidth of ADSL subscribers on the internet currently. And 512 Kbps can be used as a representative

Table 2: Parameters of sample MP4 files

| Parameter | Value |
|---|---|
| Video frame rate | 30 frames/s |
| Video Frame size | 1280 x 720 pixels |
| I-Frame rate | 1 I-frame per 50 frames |
| Audio bitrate | 192 Kbps |
| Audio sample rate | 48 kHz |
| Number of audio channels | 2 channels |

residual bandwidth of such internet subscribers during normal working conditions. Ones can say that 512 Kbps is a representative value for typical network condition, while 2 Mbps is the representative one for very good network conditions. For each sample video and a value of the above bandwidths, 5 runs are performed and the final result is obtained by averaging over the 5 results. Since we are interested only in how startup delay can be reduced thanks to caching moov atom, we let the the MAC client has moov atom the requested movie in all those runs. For the first time of requesting a MP4 video, where its moov atom is not cached yet, startup delay of MAC client actually is identical to that of HTTP progressive download.

Figure 6 and figure 7 show comparisons between MAC and progressive download in terms of startup delay under typical access bandwidth of 512 Kbps and large access bandwidth of 2Mbps. Using results from those figures and models described in 2 (equation 9, 10), we can conclude the followings:

- If waiting time threshold $T_0$ that a user can wait is not higher than 20 seconds, progressive download is not acceptable to users in most of the cases. Meanwhile, MAC provides relatively good user experience with streaming failure rates are less than 40%, 20%, 10%, 5% for patience factor $k$ equals 0, 1, 2, 3 respectively.

- If some users are sufficiently patient to wait until moov atoms are retrieved ($T_0$ is as large as 60 seconds), MAC results in much smaller streaming failure rates in all cases. Specifically, streaming failure rates are less than 10%, for $k = 0$, and less than 1% for $k > 0$.

- Startup delay of MAC is slidely smaller than HLS. The reason that HLS has low startup delay is as follows. HLS is a combination of smaller segments. Each small segment will require only small header atom, thus incurs relatively small delay to start. Although reencoding the original file into multiple smaller files help in reducing startup delay, it incurs more pre-processing cost. Total data of all segment files are higher than the original file. And data rate of the video streaming will also higher.

## 5. Related Work

In the past, most streaming solutions used streaming protocols such as RTP as a multimedia transport protocol and RTSP as session control protocol. Today, popular video streaming services are exclusively based on HTTP. Major advantages of HTTP-based streaming includes simplicity in deployment, firewall-friendly traffics, and already available at almost all client platforms. HTTP progressive download [2] best illustrates those advantages. A content provider who wants to provide video streaming simply host MP4 files on its website, and then clients can enjoy view video streams only with web browsers.

For MP4 streaming over HTTP, it is crucial to download moov atom of MP4 files in the first place. To achieve that, a couple of methods exist. Firstly, software tools, such as FFmpeg [6] and MP4 FastStart [13] can be used to relocate moov atom to the begining of the file. When movie files are progressively downloaded by clients, naturally, moov atoms are downloaded first and video playback can be started right away. Another method for downloading moov atoms in the first place is used by some video players, e.g. VLC and iOS's video player. Following this

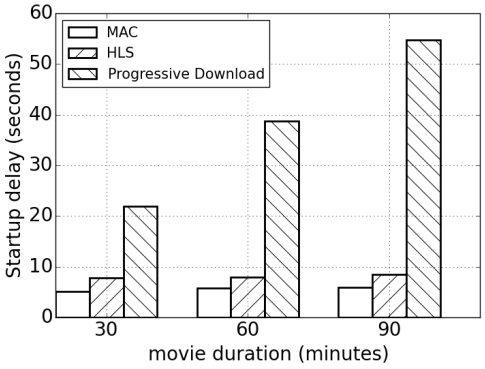


Fig. 6: Startup delay comparisons (512 Kbps access bandwidth).

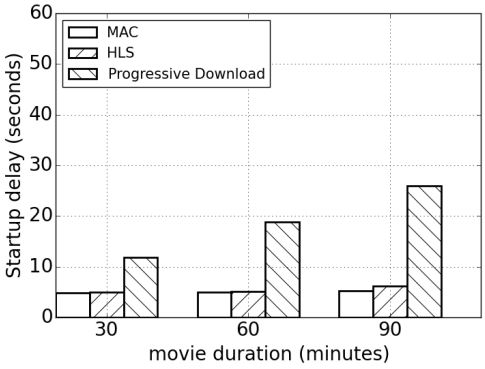


Fig. 7: Startup delay comparisons (2.0 Mbps access bandwidth).

method, a client opens a http connection with a standard HTTP Range request [14] to download the mp4 file from the begining of the file. As long as it detects that those first bytes are not moov atom, another HTTP Range request is sent to server for requesting moov atom at the end of the file.

Although the two methods improve HTTP progressive download for MP4 streaming, they do not solve the long startup delay problem as pointed out in this paper. By caching and prefetching moov atoms, MAC can far more improve user experience over progressive download while keeping simplicity and ease of deployment for the system. It is especially true for streaming of full-length movies.

Apple's HTTP Live Streaming (or HLS), Microsoft Smooth Streaming (MSS) [15], and Dynamic Adaptive Streaming over HTTP (or DASH) [16] are video streaming procotols that allows delivery of video content over HTTP. They can also provide advanced features like adaptive bitrate streaming [17]. In all those solutions, the original media is splitted into segments. Each segment can be seen as a small file with atom headers and data. Additional "packaging formats" are introduced to combine all segments into a playlist to form a complete video content. In case of Apple HLS, multimedia segments have MPEG-2 transport stream (TS) files, and packaging format is simply a text file whose extension is ".m3u8". In MSS and DASH, a more advance file format called fragmented MP4 is used for multimedia segments and XML-based format is used to packaging segments together. Since segments are small video chunks with about dozens of seconds in durations, segment atom headers are small in size and can be retrieved shortly. Startup delay is thus not as high as progressive download. However, those solutions are not as popular as MP4 format because of their newly introduced standards and less adopted in existing hardware and software. Also, a large number of video chunks could lead to several disadvantages including difficult in managing multimedia assets and higher input/output operations on harddisks.

In comparisons with fragmentation approach (like HLS, MSS, and DASH), MAC provides a easy-to-use yet efficient improvement to an already popular scheme. It does not require reencode video into multi-file media assets, thus, it produces less complexity and overheads. The disadvantage is that, it is relatively limited in providing advanced feature like streaming with adaptive bitrate. However, we believe that MAC can be used in combinations with fragmented streaming solutions, e.g., caching and prefetching atoms of segments, to improve further streaming experience.

## 6. Conclusions

In this paper, we have shown that in streaming schemes where MP4 files are used as multimedia assets, time for downloading moov atoms, a data segment in MP4 files which is important for decoding process, is noticeable. This amount of time leads to long startup delays and poor quality of experience. User behavior models were presented to show how startup delay affects the streaming system. Analysis on the models reveals that reducing startup delay can greately improve user quality of experience and keep users using the system service. In order to cut down startup delay, a caching mechanism, called MAC, was proposed and implemented. Our performance evaluations showed that MAC improves startup delay significantly in MP4 file streaming applications and have competitive startup delay with modern fragmented streaming schemes with very little complexity and overhead.

## Acknowledgments

This work was supported by the project CN.14.02 funded by VNU University of Engineering and Technology.

## References

[1] N. Färber, S. Döhla, J. Issing, Adaptive Progressive Download Based on the MPEG-4 File Format, Journal of Zhejiang University SCIENCE A 7 (1) (2006) 106–111.

[2] P. Gill, M. Arlitt, Z. Li, A. Mahanti, Youtube Traffic Characterization: A View From the Edge, in: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, ACM, 2007, pp. 15–28.

[3] Information Technology – Coding of Audio-visual Objects – Part 14: MP4 File Format (2003).

[4] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, H. Zhang, Understanding the Impact of Video Quality on User Engagement, ACM SIGCOMM Computer Communication Review 41 (4) (2011) 362–373.

[5] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, H. Zhang, A Case for a Coordinated Internet Video Control Plane, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, ACM, New York, NY, USA, 2012, pp. 359–370. doi:10.1145/2342356.2342431.

[6] F. Bellard, M. Niedermayer, et al., Ffmpeg, http://ffmpeg.org.

[7] V. Organization, VLC Media Player, http://www.videolan.org/vlc/ (2006).

[8] G. Lawton, LAMP Lights Enterprise Development Efforts, Computer 38 (9) (2005) 0018–20.

[9] R. Pantos, W. May, HTTP Live Streaming draft-pantos-http-live-streaming-05, Published by the Internet Engineering Task Force (IETF).

[10] W. Reese, Nginx: the High-Performance Web Server and Reverse Proxy, Linux Journal 2008 (173) (2008) 2.

[11] M. R. Brown, FastCGI: A High-performance Gateway Interface, in: Fifth International World Wide Web Conference, Vol. 6, 1996.

[12] Y. WANG, J. WANG, Use gns3 to Simulate Network Laboratory, Computer Programming Skills & Maintenance 12 (2010) 046.

[13] MP4 FastStart, `http://www.datagoround.com/lab/`, Accessed: 2015-12-04.

[14] R. Fielding, Lafon, Y., Ed., and J. Reschke, Ed.," Hypertext Transfer Protocol (HTTP/1.1): Range Requests, Tech. rep., RFC 7233, June (2014).

[15] A. Zambelli, IIS smooth Streaming Technical Overview, Microsoft Corporation 3.

[16] T. Stockhammer, Dynamic Adaptive Streaming Over HTTP–: Standards and Design Principles, in: Proceedings of the Second Annual ACM Conference on Multimedia Systems, ACM, 2011, pp. 133–144.

[17] Melnyk, Miguel A and Stavrakos, Nicholas J and Penner, Andrew and Tidemann, Jeremy and Breg, Fabian, Adaptive Bitrate Management for Streaming Media Over Packet Networks (2011).