AN EFFECTIVE SEGMENT PRE-FETCHING FOR SHORT-FORM VIDEO STREAMING

Nguyen Viet Hung[1,2] and Truong Thu Huong[2]
Corresponding author: huong.truongthu@hust.edu.vn
[1]Faculty of Information Technology, East Asia University of Technology, Hanoi, Vietnam
[2]School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, Hanoi, Vietnam

Abstract
The popularity of short-form video platforms like TikTok has increased recently. Short-form videos are significantly shorter than traditional videos, and viewers regularly switch between different types of content to watch. Therefore, a successful prefetching strategy is essential for this novel type of video. This study provides a resource-effective prefetching technique for streaming short-form videos. The suggested solution dynamically adjusts the quantity of prefetched video data based on user viewing habits and network traffic conditions. The results of the experiments demonstrate that, in comparison to baseline approaches, our method may reduce data waste by 21% to 83%, start-up latency by 50% to 99%, and the total time of Re-buffering by 90% to 99%.

Keywords:
Video Sharing, Short Video Streaming, TikTok, Data wastage.

1. Introduction

Mobile video streaming has quickly become one of the most popular types of online media in recent years. According to [1], video traffic accounted for 69% of all mobile data traffic, forecast to increase to 79% in 2027. Fueled by the new popular services about video sharing platforms such as TikTok [2, 3], Douyin [4, 5, 6], YouTube Shorts [7, 8] and so on, streaming short-form videos nowadays is widespread among mobile users. Users can watch, upload, and share various types of user-generated short-form videos with a duration typically a few minutes or shorter.
Short-form video streaming has very different characteristics compared to traditional video streaming. The short-form video streaming platform's algorithm recommends individual video content based on the user's continued interaction with previous videos [9]. Viewers have limited control over video playback of the video material, originality in utilities, and content. In particular, to review the previous video or skip to the following video, a user can scroll the screen whenever he wants because a handy design facilitates finding the content that interests him most.

However, if a viewer scrolls the screen (i.e., video switching) before the current video has been completely played, the downloaded but unwatched video data will be discarded [10]. This mechanism utilizes the HTTP-based protocol, which enables data transfer as quickly as TCP permits. Videos will be continuously downloaded in a list, but it does not allow using more buffers or bitrates than allowed.
As the cached video data can withstand bandwidth fluctuations, such a buffering system could successfully prevent rebuffering events. But, buffering would result in significant mobile data loss when users switch short videos frequently. Moreover, recent measurement studies have revealed that commercial short-form video platforms adopt a straightforward streaming strategy in which videos are downloaded one after another to a user's device. Unfortunately, this approach causes a significant waste of network resources [11]. Therefore, several buffer approaches have been proposed in work [12, 13, 14, 15, 16, 17, 18] with the main idea to limit the amount of prefetched video data. This reduces data waste as a user scrolls to the following video. Practically speaking, when a user scrolls a video, to lose the least amount of data during video playback, he only needs to pre-download up to 1 second of video data. Having to pre-download a video is essential as it can help reduce buffer reloading so that the video playback can be sustained in poor network conditions. But if the number of the prefetched segments is too small, it can lead to many reverse events, thus significantly reducing the overall Quality of Experience (i.e., QoE) of viewers. Previous studies used deep learning to find the optimal value of the required buffer size to adapt to the current network bandwidth. The main issue with these methods is the

Manuscript received March 5, 2023
Manuscript revised March 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.3.8

need to collect a lot of user data to train the deep learning model. Collecting user data is not an easy task when there are more and more user data protection laws are applied, such as GDPR [19].

In this paper, we propose a new resource-efficient pre-fetching scheme for short-form video streaming over the mobile networks with assumed bandwidth fluctuations. Our proposed scheme has the main ideas that can be summarized as follows.

• Firstly, our method adapts and automatically adjusts to network conditions to minimize data waste and re-caching time.

• Secondly, our method dramatically reduces the startup delay when the user scrolls the video with prefetching segments of videos, not only the current video but also the following videos in the playlist.

• Thirdly, we export the algorithm to predict segments in advance, automatically terminating video streams not used by the user, thereby limiting wasted time.

The proposed solution is proven to be able to significantly reduce data wastage, re-buffering time, and startup latency. The remaining of our paper is structured as follows. Section 2 discusses the related work. Section 3 includes some basic concepts and related entity models. Section 4 elaborates the proposed scheme. Section 5 describes the performance evaluation. Finally, Section 6 concludes our findings and future work.

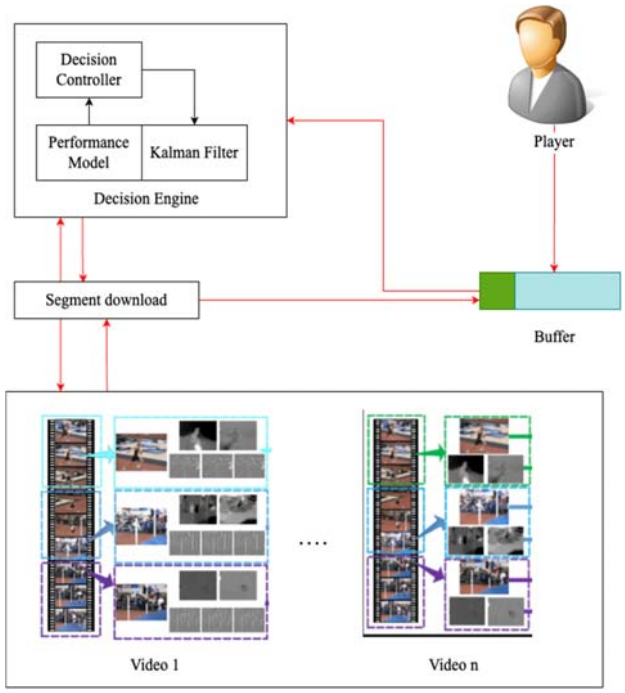
2. Related Work

Several previous studies have delved into the features of short-form video platforms, e.g. [20, 21, 22]. However, they did not offer specific solutions to improve short-form video services. The methods are all limited in terms of fetching data. Previously, in some studies on conventional VoD streaming platforms (e.g., YouTube)., some authors proposed a method to limit the client's buffer capacity to save data usage. According to the theoretical research, the client's request to download the following segment will be put on hold when the client's buffer capacity hits the threshold. That is the way the work [12] recommended the SARA algorithm by setting the buffer threshold value equal to 20s. Besides, in [13], they use the buffer threshold value equal to 30s. Although limiting cached data is an excellent way not to waste too much user data when using video streaming platforms, if you continue reducing the number of downloads before the threshold is 1 second, the user's QoE will be adversely affected. So, challenges still exist for short-form video services, e.g., network [23], data wasted when scrolling through video, or waiting time for the following video to load. And the biggest problem that needs to be solved is how to avoid wasting data and still guarantee the QoE. Up till now, there have been several studies to handle the mentioned issue with positive results. To the best of our knowledge, some academics who had studied commercial short-form video sharing sites discovered that solutions [11], and [24] still waste a lot of video data. Because in [24] said, short-form video streaming platforms such as Douyin use a simple type of video download method (called Next-One strategy). Then videos are downloaded sequentially, and the download of the following video only starts after the download of the current video is complete. Like the Next-One method, WaterFall will only download the next video when the download is finished with the current video, but this method allows downloading the next two videos. Accordingly, we propose a technique called NPM (Network a resource-efficient Prefetching Method) based on deep reinforcement learning. We refer the interested readers to the studies [24, 25, 10, 16, 26, 27, 28], to understand more clearly about DRL and compare similar streaming algorithms.

Research in [24] proposed LiveClip, an adaptive streaming strategy employing the deep reinforcement learning (DRL) technique. They design their algorithms to work in time-varying scenarios based on predictions of the user's interaction with the environment and network conditions. They also estimate the watch duration of subsequent videos. In another study, Ran el at. designed an algorithm called SSR, namely the short-form video streaming and recommendation system, which consists of a Transformer-based recommendation module and a reinforcement learning (RL) based bitrate adaptation streaming module [17]. Their design also includes user behavior, which they allocate to each short-form video to optimize the given QoE objectives. We understand the two examples above that have produced excellent results, but their methods still have some weaknesses. As stated, their systems often rely on predictions of user behavior to suggest a direction for improvement, and this is, in our opinion, very difficult to do precisely. Many other factors, such as the viewers' interests in the video content, the smoothness of playback, and others, affect when

viewers switch between videos [29]. Whereby they improve QoE very well but have not entirely solved the problem of data waste.

Furthermore, in [10], the authors propose a wastage-aware short-form video streaming method (WAS). In the WAS approach, the following segment's download is planned so that the buffer occupancy is never higher than a predetermined threshold at any given time. It can help reduce data wastage in case of the user switches from one video to another during the playback process. In addition, borrowing the idea of Bitrate Adaptive Streaming, WAS dynamically adjusts the video bitrate based on the available throughput. Multiple versions with different bitrates of a video are prepared and stored in the server in advance.



(a) System Architecture

(b) User Session

FIGURE 1: System architecture of the proposed system

In another study [16], a more improved version of the WAS system called DUASVS is a deep learning-based short-form video streaming method. In DUASVC, once the prefetch duration of the current video reaches a threshold, the video player will shift to fetching the following video. They incorporate two models: Integrated Learning, which trains/outputs a set of candidate adaptation models where different members are specialized for different network conditions. The Online Model Selection model chooses the most suitable trained model to be used for each video in the streaming videos online. The common point of these studies is that they all require a large amount of user data. In [16], collecting such a large amount of data set in practice needs to contact an anonymous (short video) streaming vendor to provide datasets collected from their commercial video servers. We are concerned that with today's many laws protecting user data, such data collection is almost impossible to do.

A network-aware prefetching method for short-form video streaming [30] (we called it NaSP), they're trying to find the segment and segment for the video. The selected value is the fixed interval they try to set according to the given network condition. However, this choice is not reasonable in our opinion. Because of constantly changing requirements and changing human behavior, the selection of values to consider is not good. Using the NaSP method, they tried to find the $Bfseg$ and K values according to the segments.

3. Short-form video streaming background and problems

In short-form video streaming, videos are kept on an HTTP server and transmitted to the user's device. Each video is encoded into one or more variations with various levels of quality. Each version is divided into identical-long-play segments that are separated in time. Users can scroll while watching a video to view another one they like. The system must promptly show the user the video.

Figure 1a shows the general architecture of the short-form video streaming system. We consider a scenario where users watch videos on a short video platform on their mobile devices and over wireless networks (e.g., Wifi, 4G/5G). Short videos are stored in an HTTP server where each video is encoded and split into multiple segments with the same playback duration.

To download a video from the server, the user device's segment downloader sends HTTP requests for the individual video segments to the HTTP server. When requests are received, the HTTP server responds with the requested segments to the client via HTTP response messages. A decision engine decides which segments should be downloaded. After sending the request, the downloader receives the requested video segment and stores it in the buffer. The user/player then decodes the incoming video segments and displays them on the user's device screen.

Figure 1b shows a typical user session in our design. Generally, in a current video streaming system, each video is divided into multiple segments at the server side. But the content of those videos are independent and unrelated to each other. However, in our design, whenever a client requests to download segments of next videos, next videos are sorted in the order of related content. For example: if the current video content is about football, then the next video will have football-related content too. A video, after being downloaded, will be removed from the sorted download list, thereby saving the cache and limiting the possibility of a second download when previously downloaded.

There are two critical challenges in short video streaming. First, the system should ensure high Quality of Experience (QoE). For that, the system should be able to maintain high video quality and, at the same time, avoid playback rebuffering. Playback rebuffering occurs when a video segment is after its playback deadline. This problem has usually caused by significant drops in available network throughput and harms the user's QoE [11]. Startup delay is another critical factor of QoE in fast video streaming. Startup delay refers to the time from the user clicks a video to the time the playback of the video begins. Because users often change the content to watch, providing low startup delay is especially important to ensure a satisfactory user experience.

The system must reduce data waste as the second problem. Data is wasted when a user switches the video they are watching since all the video data that has already been downloaded for that video is thrown away. Recent research [24], [11] indicated that on commercial short video streaming sites, roughly 45% of the downloaded video material is eventually lost. Therefore, it is crucial to minimize data loss when streaming quick videos.

Finally, this suggests the following explanation for the problem with fast video streaming. Choose the best time for the videos in the current playlist to download to enhance user viewing enjoyment and reduce data usage under varied network circumstances and dynamic user behaviors.

In order to estimate the number of prefetching segments, our method must intake the following parameters, which are defined by work [30], including: average throughput, buffer size, and start-up delay. The average network throughput (γ) is given by:

$$\gamma = \frac{1}{W}\sum_{t=t_{\text{curr ent}}-W}^{W} Thrp(t) \quad (1)$$

Where:

- $Thrp(t)$: network throughput at time t
- $t = tcurrent - W$: a time point within the range from the current time to the past W seconds.
- W: monitoring window of W seconds.

The buffer size $Bf(v^t, l^t)$ of video v^t after downloading segment lt is given by:

$$Bf(v^t, l^t) = \tau + \max\left(Bf(v^t, l^t - 1) - \frac{\tau Thr_{v^t}}{Thrp(t)}, 0\right) \quad (2)$$

Where:

- $\max\left(Bf(v^t, l^t - 1) - \frac{\tau Thr_{v^t}}{Thrp(t)}, 0\right)$: Rebuffering time at segment l^t of video v^t.
- τ: the duration of one segment.

Start-up delay ($Delay(i)$) of video v_i:

$$Delay(i) = \max\left(t^b(i, Bf_0) - t^a(i), 0\right) \quad (3)$$

Where:

- $\max\left(t^b(i, Bf_0)\right)$: time when the first Bf_0 segments of video vi are downloaded completely.
- $t^a(i)$: a time a user scrolls to video v_i.

4. Proposed Network-aware Prefetching System - NPM

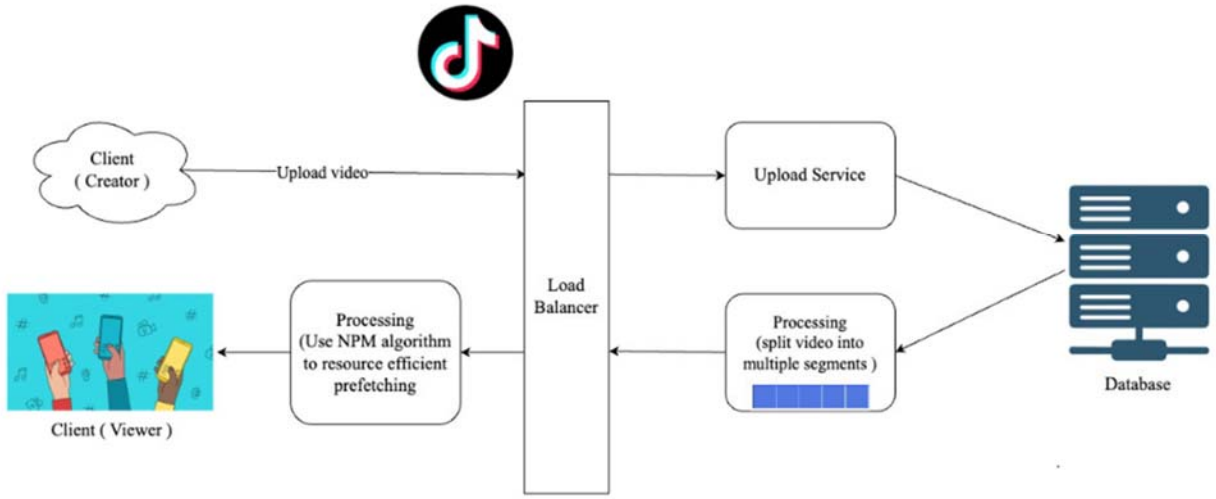


FIGURE 2: Our proposed recommendation system for content conversion models

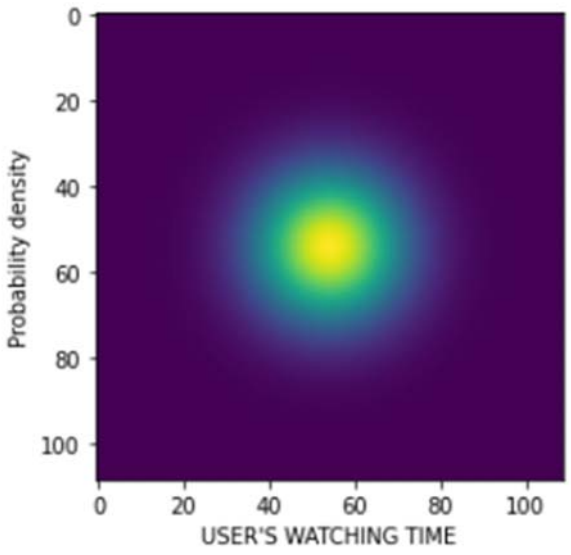


FIGURE 3: Gaussian-distributed User's watching time parameter

4.1. Overall system operation

Figure 2 depicts our proposed design for TikTok's video scrolling system. The overall operation can be described as follows:

- The Client (Creator) uploads the video. And at the same time, there will be a provision for them to upload the video for the client. Uploaded short videos will go through the load balancer, effectively distributing the traffic to the server to improve the best speed and performance and ensure the server not to work with excessive movement.

- Next, the short videos are passed through the upload service and saved to the database. To playback the short videos for the clients to watch, the server divides the video into several segments. Those video segments continue to be passed through the load balancer. Still, before the clients can enjoy those short videos, they will go through one processing step called - NPM. NPM is designed to significantly reduce the start-up delay for users when scrolling videos (described in the following subsection).
- Finally, the video reaches the Client (Viewer), who can experience the short-form videos without interruption.

4.2. Network resource-efficient Prefetching Method – NPM

NPM - Network resource-efficient Prefetching Method is designed for short-form video streaming to improve the quality of user experience by optimizing the resource network.

4.2.1. Selection of number of pre-fetched segments

Intuitively, to reduce the waiting time for a user in each specific interval, we need to find a good way to buffer (or prefetch) fewer data. It leads to the fact that we should specify Bf_{seg} - the threshold of the number of segments that can be pre-fetched. Bf_{seg} is to be the function of the present network bandwidth, bit rate, user's watching time, since downloading the videos in the current playlist is influenced by variable network throughput conditions, user's bitrate conditions, and dynamic user behaviors. Although, the smaller the value of Bf_{seg}, the lower the loss. But choosing Bf_{seg} too small can lead to re-buffering under a reduced state of the network throughput condition.

Therefore, we suggest dynamically adjusting the number of prefetched segments based on current network conditions. The whole process will finally result in a certain data waste, rebufferings, and start-up delays that, in turn, have a certain impact on the overall service experience of users. Bf_{seg} is fitted by Polynomial Regression from the 3 variables:

Let BW_x denote the bandwidth variations trace, Thr_y denote the throughput variation trace, and UT_z denote the user's watching time trace.

The bandwidth trace is used from [31], while the throughput trace is taken from [30]. The throughput trace files can be illustrated in Figure 4. For the user watching time, because identifying user behavior is very challenging [32], a classical strategy to deal with the situation of having no prior information in the implementation is to use a Gaussian distribution for the parameters of interest. Therefore, we use Gaussian distribution to create user behavior traces for user scrolling behaviors, as illustrated in Figure 3.

The resulted Bf_{seg} is then fitted to the function as follows:

$$Bf_{seg} = BW_x + 4.5523Thr_y + 2.072UT_z - 2 \quad (4)$$

with: confidence interval of 95% and constant of 0.13. The found $Bfseg$ will be, then, used in the NPM algorithm in next step.

4.2.2. NPM algorithm

Finding a new video to load into the system is essential because it affects the overall experience of users over the service. For example, downloading a new video with different content will take us a long time to reload the new video.

In general, if a new video with different content is prefetched to the buffer, it may be likely not the content a user desires to watch. So it will take time for this user to find another video of the same content type and reload it to the system (i.e. long video switching time). As the result, the long video switching time will badly affect on the user's satisfaction while using the video streaming service.

Therefore, NPM automatically finds videos with the same content as the current videos and downloads them based on user behavior. Those found videos will

be saved to the counter before being displayed on the client's device. When a user scrolls to the next video, NPM automatically ends the previous video stream to release the buffer for the following video.

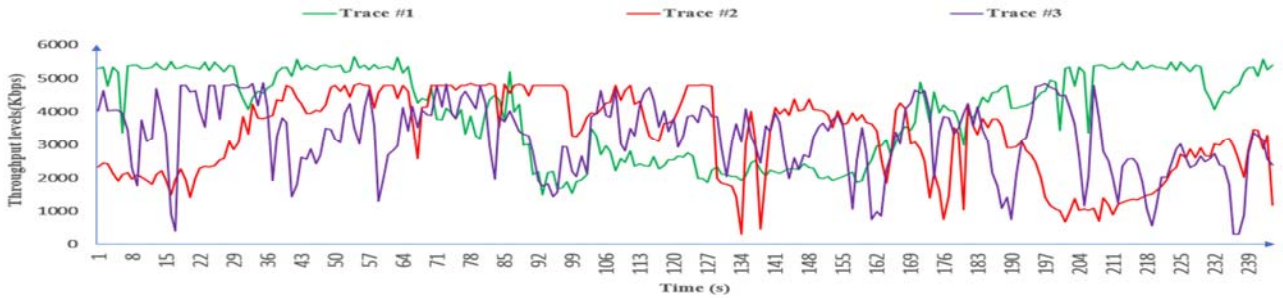


FIGURE 4: Throughput variation on each throughput trace file.

Algorithm 1 NPM Algorithm

Input: $n, \gamma, tx_v, BW_x, UT_z, Thr_y, t_{step}, Bf_{seg}, K, Bf_0, Q_v, P_v$

```
while t <= Σ_{v=1}^{n} t_v do
    Update Bf_seg Using Eq. 4 and γ using Eq. 1;
    a = 1
    if tx_v == UT_z then
        v++ (User scroll to the next video)
    end
    if v == n then
        break
    end
    else if v < (n − 1) then
        i = v + 2
        while i < (v + K) && i < n do
            (Choose the video v+a with the smallest
             amount of buffer in the next K video to buffer)
        end
    end
    if ((Bf_v − Q_v) < Bf_seg) && (Bf_v ≤ P_v) then
        Buffer video v Using Eq. 2
        if Bf_v − Q_v < Bf_0 then
            Rebuffer video v using Eq. 3
        end
    end
    else if ((Bf_{v+a} < Bf_seg) && (v < n) then
        Buffer video v+a Using Eq. 2
        if tx_v == UT_z then
            a− = 1
        end
    end
end
else
    The buffer already has the current video and forwards
     the next K video.
    tx_v+ = t_step; t+ = t_step ; Q_v+ = t_step
end
```

Provided a user scrolls to the next video while watching the current video, in that case, Bf_{seg} - the prefetched segments - is also considered waste data.

It can also be noted that the higher the value of γ, the lower the value of Bf_{seg} with a minimum buffer size of two segments. This is to facilitate no startup delay when the user scrolls from one video to another one. Specifically, the algorithm considers the following K videos to the current video in the playlist.

Let Q_v denote the play progress of the video, P_v denote the playback of the video, n denote the total number of videos users watch in a viewing session, and tx_v is the current video watch time of video v.

Step 01:

NPM defines the average throughput γ according to formula (1) and the buffer size or the max number of prefetched segments Bf_{seg} according to formula (4).

Step 02:

In this step, during the viewing session, NPM continuously compares the buffer size of the current video with Bf_{seg}. If the current buffer size is less than f_{seg}, NPM will download 1 second of the following segments of the current video until the buffer size exceeds threshold $Bfseg$.

If the amount of video data v left in the buffer is larger than the segment Bf_0, the video can continue to be played back. After that, NPM will continue prefetching Bf_{seg} segments of the next K videos into the current playlist.

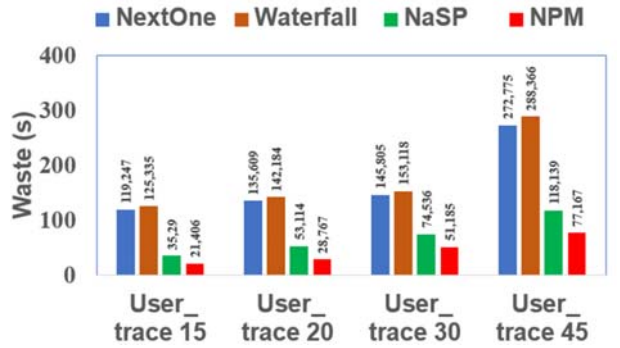
Step 03:

In this step, before buffering, if the case of step 2 is not satisfied, NPM will find new videos of similar content to load into the list, continuing to segment and

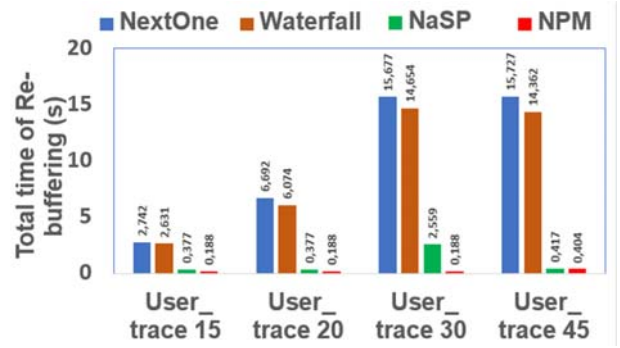
to load these segments as specified Step 2, until the end of the video is loaded into the buffer.

The pseudo-code of NPM can be found in Algorithm 1.

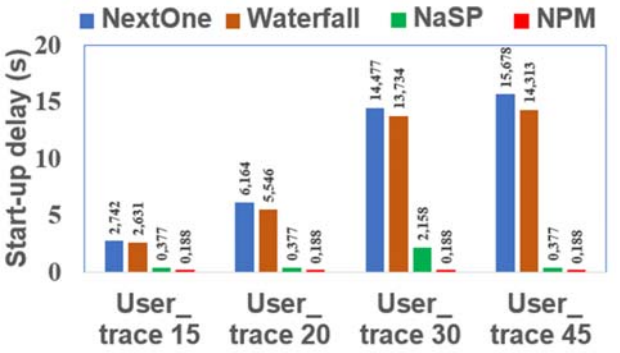
5. Performance Evaluation



(a) Waste (s)



b) Total time of Re-buffering (s)

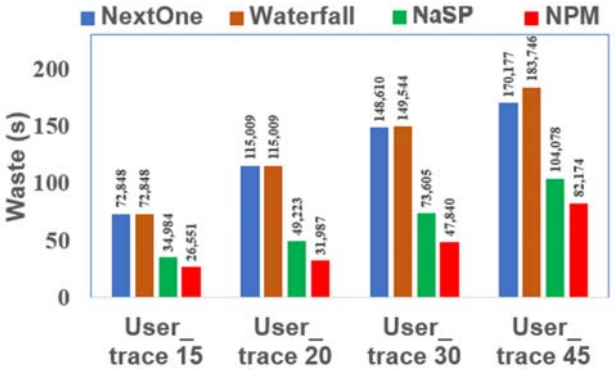


(c) Start-up delay (s)

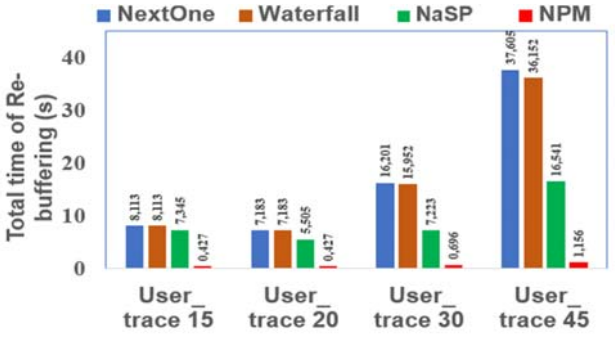
FIGURE 5: NPM vs. reference methods under trace #1

5.1. Experimental Settings

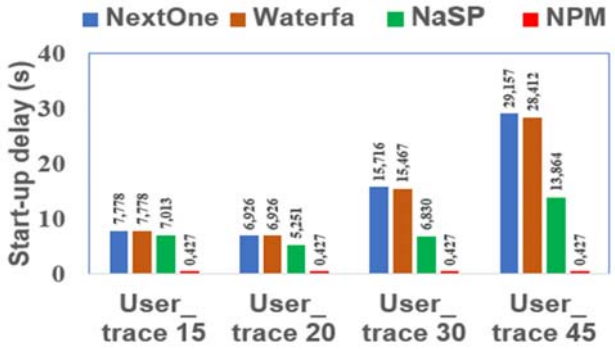
In our experiment, the experimented videos (or films) are 15 seconds long and have a constant bitrate of 1Mbps (so that each video will have 15Mb in size),



(a) Waste (s)



b) Total time of Re-buffering (s)

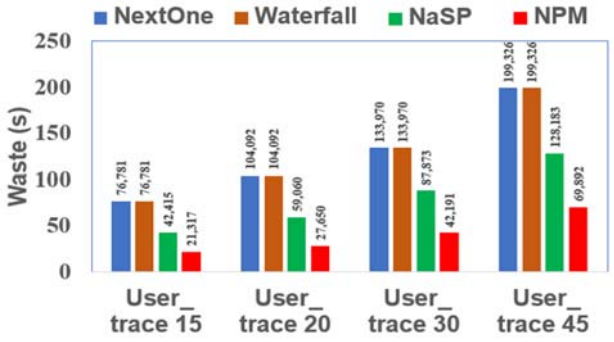


(c) Start-up delay (s)

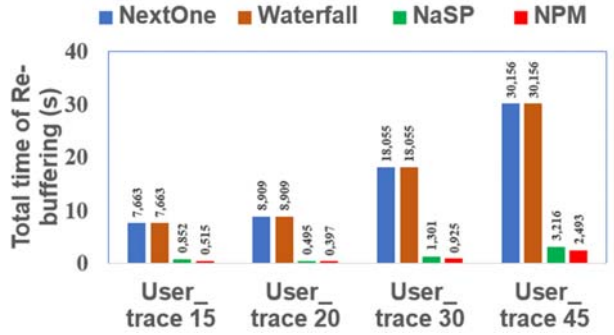
FIGURE 6: NPM vs. reference methods under trace #2

and the threshold of the number of next videos that can be pre-buffered $K = 4$. The videos are then each split into 15 successive one-second chunks.

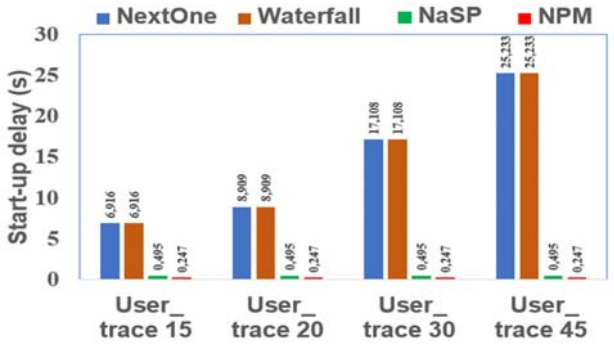
In our evaluation, we do not address additional factors like resolution. The performance of our solution and



(a) Waste (s)



b) Total time of Re-buffering (s)



(c) Start-up delay (s)

FIGURE 7: NPM vs. reference methods under trace #3

other existing methods are measured under various network conditions (i.e., network throughput per second) emulated by the three network traces shown in Figure 3.

The experiment is written in Python and tested on a computer running 64-bit Windows 10 with the following configuration: 16384MB of RAM and an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz (4 CPUs), 2.6GHz processor.

NPM is compared with the three reference methods with the main characteristics and operation briefly described as follows:

- Next-one [33]: This method separately buffers each segment of the video to be viewed. The following video will not start buffering until the previous one has finished downloading. The maximum number of subsequent videos that will automatically be buffered and cached is 1.
- Waterfall [33]: The following videos will only be downloaded if the current video has finished downloading, just like the Next-one technique. The main distinction is that two additional videos can now be buffered, which may cut re-buffering time but increase the waste time.
- NaSP [30]: The suggested technique works similarly to our method by prefetching sections of the current video and the videos that will follow it in the playlist. The technique, however, also dynamically modifies the number of buffered segments following the current network circumstances but with given conditions. Not suitable for network fluctuations that change continuously.

In contrast to NPM, the Next-one and Waterfall techniques have a set amount of segments and films that will be buffered. This is unreasonable and cannot be applied dynamically to the non-constant user and network throughputs. Besides, NaSP uses a similar strategy to ours. However, NaSP segmentation is inefficient, and the video fetching is inconsistent because this method uses a specific range of conditions to deal with large bandwidth changes. At

TABLE 1: Performance of NPM vs. reference methods under different throughput traces

Method		NetOne	Waterfall	NaSP	NPM
Throughput trace #1	**Waste**	168.36	177.25	70.27	**44.63**
	Total time of Re-buffering	10.21	9.43	0.93	**0.24**
	Start-up delay	9.77	9.06	0.82	**0.19**
Throughput trace #2	**Waste**	126.66	130.29	65.47	**47.14**
	Total time of Re-buffering	17.28	16.85	9.15	**0.68**
	Start-up delay	14.89	14.65	8.24	**0.43**
Throughput trace #3	**Waste**	128.54	128.54	79.38	**40.26**
	Total time of Re-buffering	16.2	16.2	1.47	**1.08**
	Start-up delay	14.54	14.54	0.5	**0.25**

the same time, our approach will automatically adapt to changing network conditions.

To simulate actual network conditions and verify how NPM works in reality. We use three network bandwidth traces from mobile networks [30], as shown in Figure 4 with 250s.

- Throughput trace #1 emulates a relatively high throughput averaging close to 5000Mbps with slight fluctuations.
- Throughput trace #2 emulates the condition in which the network throughput fluctuates considerably, with an approximate average speed of about 3000Mbps.
- Throughput trace #3 emulates a network condition with moderate fluctuation at a speed of about 2000Mbps.

5.2. Experimental Results

In this paper, three evaluation metrics are taken into account:

- Waste time: the time spent caching a video that the user has never watched.
- Start-up delay: the interval between when a user selects the following video and when that video is ready for playback.
- Total time of Re-buffering: The time the user must wait until the video starts playing.

The performance of NPM is measured in three different network throughput conditions (i.e., three different trace files), summarized in Table 1 and visualized in Figure 5, Figure 6, and Figure 7.

As seen in Figure 5, with the Throughput trace #1, NPM has waste time lower than NaSP by about 31 to 46%, and lower than NextOne and Waterfall by about 64 to 83%. The total time of re-buffering and Start-up delay of NPM is reduced by up to 99% compared to the three reference methods.

As seen in Figure 6, with the Throughput trace #2, NPM has a Waste time lower than NextOne and Waterfall by 55 to 72%, and lower than NaSP by about 21 to 35%. Furthermore, NPM shows a 90 to 97% reduction in total buffer time compared to the three methods. In this network condition, the start-up delay is about 94 to 99% lower than the three reference methods.

As seen in Figure 7, with the Throughput trace #3, the first 20 seconds make the start-up delay inevitable. However, NPM can still reduce start-up delay from 96 to 99% compared to NextOne and Waterfall, and 50% compared to NaSP. Waste time of our NPM method is still reduced by 65 to 73% compared to NextOne and Waterfall method, 45 to 53% reduction compared to NaSP method. The total Re-buffering time of NPM is still 91 to 96% lower than NextOne and Wastefall and 50% lower than NaSP.

As summarized in TABLE 1, we can conclude that NPM is proven to outperform the referenced methods under three typical network conditions in terms of Waste time, Total time of Re-buffering, and Start-up delay. The reduction of those 3 parameters will

improve the overall user satisfaction over the video streaming service.

In addition, the findings prove two points:

- First, there is time wastage caused by the buffered videos that users never watch because they switch to the next video. If this time wastage can be reduced, the user satisfaction could be increased.
- The second is the re-buffering period, during which the user must wait for the viewing section to buffer fully. Therefore, this period should be decreased as well.

6. Conclusion

In this paper, we have demonstrated the way to send short videos to viewers via time-varying networks. By pre-downloading portions of videos, including the current video and the following videos in the playlist, our suggested solution drastically reduces startup delay when the user scrolls the video and minimizes data waste and recaching time. Compared to the 3 most recent reference methods, the experimental results show that our method reduces data waste by 21% to 83%, startup latency by 50% to 99%, and overall re-buffering time by 90% to 99%.

Although NPM has improved the three concerned metrics significantly, the metric - quality of experience (QoE) from clients' perspective – has not been fully considered. This metric opens a new challenge for our future work when all methods should finally receive a high QoE score ranked by users' subjective experience.

Acknowledgment

This research is funded by Hanoi University of Science and Technology (HUST) under Project number T2022-PC-012. Also, we would like to thank Mrs. Phan Thi Yen, Ms. Bui Anh Thu, Ms. Nguyen Mai Cham of the East Asia University of Technology for supporting us in the experiments.

References

[1] "Ericsson mobility report," ttps://www.ericsson.com/4ad7e9/assets/ local/ reports-papers/mobility-report/documents/2021/ ericsson-mobility-report-november-2022.pdf, June 2022.

[2] "Tiktok," https://www.tiktok.com, accessed: 2022-08-10.

[3] D. Klug, Y. Qin, M. Evans, and G. Kaufman, "Trick and please. A mixed-method study on user assumptions about the tiktok algorithm," in 13th ACM Web Science Conference 2021, ser. WebSci '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 84–92. [Online]. Available: https://doi.org/10.1145/3447535.3462512

[4] "Douyin," https://www.douyin.com, accessed: 2022-08-09.

[5] L. Sun, H. Zhang, S. Zhang, and J. Luo, "Content-based analysis of the cultural differences between tiktok and douyin," in 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 4779–4786.

[6] T. Shao, R. Wang, and J.-X. Hao, "Visual destination images in user-generated short videos: An exploratory study on douyin," in 2019 16th International Conference on Service Systems and Service Management (ICSSSM), 2019, pp. 1–5.

[7] "Youtube short," ttps://www.youtube.com/shorts, accessed: 2022-08-09.

[8] A. M. Putri, D. A. P. Basya, M. T. Ardiyanto, and I. Sarathan, "Sentiment analysis of youtube video comments with the topic of starlink mission using long short term memory," in 2021 International Conference on Artificial Intelligence and Big Data Analytics, 2021, pp. 28–32.

[9] M. E. D. Klug, Y. Qin and G. Kaufman, ""trick and please. a mixedmethod study on user ssumptions about the tiktok algorithm," Virtual Event, United Kingdom, p. 84–92, 2021.

[10] G. Zhang, K. Liu, H. Hu, and J. Guo, "Short video streaming with data wastage awareness," in 2021 IEEE International Conference on Multimedia and Expo (ICME), Shenzhen, China, 2021, pp. 1–6.

[11] Y. Zhang, Y. Liu, L. Guo, and J. Y. B. Lee, "Measurement of a large-scale short-video service over mobile and wireless networks," IEEE Transactions on Mobile Computing, pp. 1–1, 2022.

[12] H. K. Yarnagula, P. Juluri, S. K. Mehr, V. Tamarapalli, and D. Medhi, "Qoe for mobile clients with segment-aware rate adaptation algorithm (sara) for dash video streaming," ACM Trans. Multimedia Comput. Commun. Appl., vol. 15, no. 2, jun 2019. [Online]. Available: https://doi.org/10.1145/3311749

[13] X. Chen, T. Tan, and G. Cao, "Energy-aware and context-aware video streaming on smartphones," in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 861–870.

[14] G. Huang, W. Gong, B. Zhang, C. Li, and C. Li, "An online buffer-aware resource allocation algorithm for multiuser mobile video streaming," IEEE Transactions on Vehicular Technology, vol. 69, no. 3, pp. 3357–3369, 2020.

[15] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in Proceedings of the 2014 ACM conference on SIGCOMM, 2014, pp. 187–198.

[16] G. Zhang, J. Zhang, K. Liu, J. Guo, J. Lee, H. Hu, and V. Aggarwal, "Du-asvs: A mobile data saving strategy in short-form video streaming," IEEE Transactions on Services Computing, pp. 1–1, 2022.

[17] D. Ran, Y. Zhang, W. Zhang, and K. Bian, "Ssr: Joint optimization of recommendation and adaptive bitrate streaming for short-form video feed," in 2020 16th International Conference on Mobility, Sensing and Networking (MSN), 2020, pp. 418–426.

[18] D. Ran, H. Hong, Y. Chen, B. Ma, Y. Zhang, P. Zhao, and K. Bian, "Preference-aware dynamic bitrate adaptation for mobile short-form video feed streaming," IEEE Access, vol. 8, pp. 220 083–220 094, 2020.

[19] P. Voigt and A. Von dem Bussche, "The EU general data protection regu-lation (GDPR)," A Practical Guide, 1st Ed., Cham: Springer International Publishing, vol. 10, no. 3152676, pp. 10–5555, 2017.

[20] S. McLachlan, "Instagram reels algorithm: Everything you need to know," https://blog.hootsuite.com/instagram-reels-algorithm/, 2022-05-18.

[21] F. W. Lei Zhang and J. Liu, "Mobile instant video clip sharing with screen scrolling: Measurement and enhancement," in IEEE Transactions on Multi-media 20. IEEE, 2018, p. 2022–2034.

[22] Z. Chen, Q. He, Z. Mao, H.-M. Chung, and S. Maharjan, "A study on the characteristics of douyin short videos and implications for edge caching," in Proceedings of the ACM Turing Celebration Conference-China, 2019, pp. 1–6.

[23] Y. Zhang, P. Li, Z. Zhang, B. Bai, G. Zhang, W. Wang, and B. Lian, "Chal-lenges and chances for the emerging short video network," in IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 1025–1026.

[24] J. He, M. Hu, Y. Zhou, and D. Wu, "Liveclip: Towards intelligent mobile short-form video streaming with deep reinforcement learning," in Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, ser. NOSSDAV '20, Istanbul, Turkey, 2020, p. 54–59.

[25] J. Guo and G. Zhang, "A video-quality driven strategy in short video streaming," in Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Alicante Spain, 2021, p. 221–228.

[26] S. Sengupta, N. Ganguly, S. Chakraborty, and P. De, "Hotdash: Hotspot aware adaptive video streaming using deep reinforcement learning," in 2018 IEEE 26th International Conference on Network Protocols (ICNP), 2018, pp. 165–175.

[27] A. Singh, N. Thakur, and A. Sharma, "A review of supervised machine learning algorithms," in 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). Ieee, 2016, pp. 1310–1315.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," nature, vol. 518, no. 7540, pp. 529–533, 2015.

[29] G. Zhang, K. Liu, H. Hu, V. Aggarwal, and J. Y. Lee, "Post-streaming wastage analysis–a data wastage aware framework in mobile video streaming," IEEE Transactions on Mobile Computing, vol. 22, no. 1, pp. 389–401, 2021.

[30] D. Nguyen, P. Nguyen, V. Long, T. T. Huong, and P. N. Nam, "Network-aware prefetching method for short-form video streaming," in 2022 IEEE 24th International Workshop on Multimedia Signal Processing (MMSP), 2022, pp. 1–5.

[31] H. T. Le, N. P. Ngoc, A. T. Pham, and T. C. Thang, "A probabilistic adaptation method for http low-delay live streaming over mobile networks," IEICE TRANSACTIONS on Information and Systems, vol. 100, no. 2, pp. 379–383, 2017.

[32] D. Deshpande and S. Deshpande, "Analysis of various characteristics of online user behavior models," International Journal of Computer Applications, vol. 161, no. 11, pp. 5–10, Mar 2017. [Online]. Available: http://www.ijcaonline.org/archives/volume161/number11/27190-2017913127

[33] J. He, M. Hu, Y. Zhou, and D. Wu, "Liveclip: Towards intelligent mobile short-form video streaming with deep reinforcement learning," in Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, ser. NOSSDAV '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 54–59. [Online]. Available: https://doi.org/10.1145/3386290.3396937

NGUYEN VIET HUNG received the B.Sc.degree in Bachelor of Informatics pedagogy from the faculty of Engineering Technology of Ha Tinh University, Vietnam, in 2012, the M.Sc. degree Master of Information Technology from the Faculty ofInformation Technology of Ho Chi Minh City University of Technology, Vietnam, in 2016, andhas been learning the Ph.D. degree in Telecommunications Engineering from the Hanoi University of Science and Technology, Vietnam. His research interests include multimedia communications, network security, artificial intelligence, traffic engineering in next-generation networks, QoE/QoS guarantee for network services, green networking, and applications.

TRUONG THU HUONG received the B.Sc.degree in Electronics and Telecommunications from the Hanoi University of Science and Technology (HUST), Vietnam, in 2001, the M.Sc. degree in information and communication systems from the Hamburg University of Technology, Germany, in 2004, and the Ph.D. degree in telecommunications from the University of Trento, Italy, in 2007. Her research interests are oriented toward network security, artificial intelligence, traffic engineering in next-generation networks, QoE/QoS guarantee for network services, green networking, and development of the Internet of Things ecosystems and applications.