Mobile Instant Video Clip Sharing: Modeling and Enhancing View Experience

Lei Zhang, Student Member, IEEE, Feng Wang, Member, IEEE, and Jiangchuan Liu, Senior Member, IEEE

Abstract—With the rapid development of wireless networking and mobile devices, anytime and anywhere data access becomes readily available nowadays. Given the crowdsourced content capturing and sharing, the preferred content length becomes shorter and shorter, even for such multimedia data as video. A representative is Twitter’s Vine service, which, mainly targeting mobile users, enables them to create ultra-short video clips and instantly post and share with their followers. In this paper, we present an initial study on this new generation of instant video clip sharing service enabled by mobile platforms and explore the potentials towards its further enhancement. We closely investigate its unique mobile interface, revealing the key differences between Vine-enabled anytime anywhere data access patterns and that of traditional counterparts. We then examine the scheduling policy to maximize the user watching experience as well as the efficiency on the monetary and energy costs. We show that the generic scheduling problem involves two subproblems, namely, pre-fetching scheduling and watch-time download scheduling, and develop effective solutions towards both of them. The superiority of our solution is demonstrated by extensive trace-driven simulations. To the best of our knowledge, this is the first work on modeling and optimizing the instant video clip sharing on mobile devices.

Index Terms—Mobile Device, Video Clip, Instant Video Sharing, Scheduling.

I. INTRODUCTION

In the past decade, the user-generated content (UGC) available on the Internet (e.g., images, videos, micro-blogs, etc.) has presented an explosive growth trend, which provides significant opportunities for both understanding how users utilize the Internet and enhancing their experience. Video is no doubt a dominant type of media in terms of content generation and sharing. Specifically, for online video sharing, we have witnessed a three-stage evolution with distinct characteristics. The first generation, represented by YouTube [1], [2], [3], allows users to upload and watch videos directly on a number of video sharing sites (VSSes) over the Internet, in which videos are usually recommended by VSSes or found by search results via search engines. Later, such online social network services as Facebook and Twitter [4], [5], [6] emerged to directly connect people through cascaded relations, and information thus spreads much faster and more extensively. They greatly changed the earlier video access patterns, too, through proactively and efficiently sharing among friends the video links from external VSSes [7]. More recently, with the pervasive penetration of wireless mobile networks and the advanced development of smartphones and tablets, video clips can now be ubiquitously generated and accessed [8]. This leads to the emergence of a new generation video sharing services that focus primarily on capturing and sharing short video clips with mobile users; a representative is Twitter’s Vine [9] that was officially launched in the beginning of 2013.

Along with the evolution of video sharing services, users’ video watching habits vary significantly: the users of the first generation video sharing services literally browse videos in the Internet through VSS portals or search engines, and in the second generation, the social ties between friends accelerate video propagation among users who have common interests. The ubiquitous mobile access, on which the third generation video sharing services base, allows users to watch videos much more easily and frequently, which in turn enhances the social ties. The migration towards mobile platforms also makes the content length of individual videos being shorter and shorter, from tens of minutes of YouTube video watching to several seconds of instant video clip watching (e.g., 6-second length limit of Vine videos). Instead of having fine-grained VCR controls on the video playback, mobile users typically browse and watch interesting contents by scrolling their mobile device’s screen over tons of instant video clips, which are organized in playlists and played seamlessly as the user’s viewpoint moves.

Maintaining high-quality user experience during video playback is crucial to all video sharing services, and it becomes an even more challenging task under the mobile scenarios. To put this into perspective, a YouTube video user can tolerate a start-up delay from several seconds to even tens of seconds (advertisements before the playback can be viewed as a kind of delay); an instant video clip itself however is much shorter (e.g., 6-second long for Vine), and a Vine video view can therefore hardly tolerate such typical start-up delays for buffering. Moreover, considering the huge amount of instant video clips consumed during the screen scrolling, the frequent interruptions of single video’s playback may largely hurt the user’s viewing experience for the whole playlist.

In this paper, we for the first time identify and characterize the unique watching behaviors of the new generation mobile video sharing services, namely, batch view, passive view, and screen scrolling. Using Vine as a representative, we closely investigate its unique mobile interface, revealing the key differences between Vine-enabled anytime anywhere data.
access patterns and that of traditional counterparts. We then examine the scheduling policy to maximize the user watching experience as well as the efficiency on the monetary and energy costs. We show that the generic scheduling problem involves two subproblems, namely, pre-fetching scheduling and watch-time download scheduling, and develop effective solutions towards both of them. We suggest novel popularity-based and wireless-aware pre-fetching for the instant video clips, as well as predicting the watching durations of the approaching videos by such user input gestures as dragging and flinging, and then dynamically adapting the schedule for real-time video downloading. Using extensive simulations driven by the real-world traces, we show that our solution can significantly improve the user watching experience while still keeping both the monetary and energy costs relatively low.

The rest of the paper is organized as follows: Section II presents an overview of the new generation of mobile instant video clip sharing services, which reshape the way that mobile users watch videos, and discusses the motivation of our modeling work. In Section III, we formulate the generic scheduling problem. By dividing it into two subproblems and conquering them separately, we then propose a general solution for the generic problem. Section IV evaluates our solution by extensive trace-driven simulations and a further discussion on other open issues is given in Section V. Finally, we conclude our paper in Section VI.

II. BACKGROUND AND MOTIVATION

We now take a close look at Vine, a representative of the new generation mobile instant video clip sharing services. The social tie in Vine is the typical follower-followee relationship, as in Twitter and other similar social network services, e.g., Instagram and Flickr. The users of Vine can follow others whom they are interested in, and then receive updates from them. They can also access the public channels to view promoted videos or videos on specific topics. A typical mobile Vine client has four pages: Home/Feed, Explore, Activity and Profile. As illustrated in Figure 1, a user can view, like, comment, and share (repost) recent posts from others that it follows in the “Home/Feed” page; in the “Explore” page, popular/trending posts, as well as dedicated public channels, are highlighted, together with the interface for the user to search specific videos or people of interest. The other two pages are not designed for video watching; the “Activity” page acts as the notification center showing the recent events, and the “Profile” page allows a user to customize his/her personal settings.

Vine preserves similar social structure as in other online social networks. A key (and significant) difference is that the media of interest in Vine are now ultra short video clips (with 6 second limit for now), while not the traditional texts or still images. This makes its user experience notably different from that of the traditional mobile social networking and video sharing services. In traditional VSSes and OSNs, the users need to click on the interested item to view or link to a specific video, which only allows users to view one video each time/click. Vine-like services, however, return a playlist of video clips once the user decides to view the updates for certain users, tags, or channels. As the user scrolls the screen of his smartphone/tablet, the mobile app plays a number of video clips from the generated list seamlessly. Given the linear formation of instant video clips in playlists, screen scrolling is the key user action for the mobile instant video clip sharing services, which uniquely differentiates Vine-like services not only from traditional VSSes and OSNs, but also from other mobile VoD or video streaming applications. The small file size of the instant video clip makes it very suitable for the mobile scenario in terms of downloading and watching: the user can easily access the service at anytime and anywhere, and watch a number of instant video clips even for a short duration (e.g., waiting for a bus or during a course break).

We refer to this unique user behavior of viewing multiple video clips with the screen scrolling as Batch View. The batch view behavior implies that mobile users can watch a considerable amount of instant video clips within the playback time of one normal video such as a YouTube video. Besides batch view, another distinct feature is Passive View, as the mobile users have to “passively” watch some of the video clips in the playlist. Similar to a Twitter-like user interface, which is widely adopted in other social networking services, the selected instant video clips that will be viewed are arranged in order, and users have no control over the order of the playbacks. If the user is only interested in two specific video clips that are separated in the playlist, s/he may have to watch all the video clips between them. The only way to proactively skip an instant video clip is to scroll the screen so fast that there is not enough time to play it, which however can hardly be well controlled in practice. If the user scrolls the screen slowly, the following ultra short video clip may have been downloaded and started to play, and thus the user will probably watch it. As the 6-second playback time is so short that it is even not enough to finish reading the description of some video clips (users may have already watched the video clip before making the decision based on the description), the Vine users tend to passively view numerous video clips. The two key user behaviors abstract and emphasize the unique features of this new generation of mobile video clip sharing service.

![Fig. 1. Vine’s interfaces](image-url)
The mobile instant video clip sharing services often lack VCR controls (such as rewind and fast forwarding) that are common in traditional video playback; rather, the mobile users can scroll the screen over the playlist to watch different instant video clips. This mode of instant video clip watching seems to be inflexible, but has its unique advantages in the mobile scenarios. First, as the instant video clips are so short, there is no need to have close and complex controls over each of them. Moreover, given the limited display size of each video, screen scrolling is effective in approaching successive instant video clips in the playlist, enabling users to find interesting contents more easily, and accelerates the propagation of popular videos. Yet, as the key of this video watching mode, screen scrolling, if not being handled properly, may ruin the viewing experience. In the worst case (e.g., a user downloads every instant video clip through 3G with a very limited bandwidth), a vicious circle can happen: the downloading of a just skipped video will take up the network resources and block the downloading of the one that the user intends to watch, which will in turn force the user to give up watching the target video and scroll forward to search for other interesting videos. Conventional approaches, such as deploying proxy servers, may not deal with this issue well, as the bottleneck still exists at the last hop where mobile devices have to download a considerable amount of instant video clips. On the other hand, as a well-known fact, in traditional video sharing services, user interests always focus on a small portion of popular contents. We examine the popularity distribution of Vine videos as shown in Figure 2 (details about the data collection will be introduced in Section IV), where the top 5% popular Vine video clips account for more than 99% reposts. Compared to the first two generations of video sharing services, Vine shows a much higher skewness: the top 10% popular YouTube videos account for nearly 80% of views [3]; whereas the top 2% videos in Renren take up 90% of the total requests, and the top 5% videos attract 95% of requests [9]. This extreme skewness implies the opportunities for pre-fetching popular contents at mobile terminals before the video watching. We accordingly model the unique instant video clip watching mode, and enhance the user experience from the mobile side. To the best of our knowledge, this is the first work on modeling and optimizing the instant video clip sharing from this perspective.

### III. Download and Watch Scheduling: Problem and Solution

As mentioned, Vine videos are organized in different playlists, which can be characterized into three types: the list of video updates from followees (social videos), the list of promoted videos in popular sections (popular videos), and the list of user uploaded videos in recent sections (recent videos). Only the playlist of social videos changes with different users, and the other two types of playlists remain the same across users. Denote the playlist of instant video clips that will be watched by the user as $V = \{v_1, v_2, ..., v_n\}$. As illustrated in Figure 3 according to different user input actions, each instant video clip may remain in the user’s viewport for different durations. We use $U = \{u_1, u_2, ..., u_n\}$ to denote such durations, where $u_i$ corresponds to the duration that the user watches the video $v_i$. Also, we let $u_0$ denote the time that the user starts watching the playlist. We consider two types of network connections in this formulation: mobile connection (e.g., 3G) and wireless connection (e.g., WiFi). We use $B(t)$, $C(t)$, and $E(t)$ to denote the available bandwidth, the monetary cost, and the energy consumption at a given time $t$, respectively, where $B(t) \in \{B_{\text{wifi}}, B_{3G}\}$, $C(t) \in \{C_{\text{wifi}}, C_{3G}\}$ ($C_{\text{wifi}} = 0$, since the cost for WiFi connections is usually negligible), and $E(t) \in \{E_{\text{wifi}}, E_{3G}\}$. As in practice both the major mobile communication protocols and wireless communication protocols work in some basic time units, in our problem, we also divide the time evenly into discrete time slots, where one time slot may contain an integer multiple of such basic time units. Let $R$ be the video streaming rate of the mobile video sharing service, and $L$ be the video length (in Vine’s case, 6 seconds). It is worth noting that given the video length limit is only several seconds, most users create and post videos that fulfill the length limit and thus the file size of Vine videos is almost the same after transcoding to a certain resolution. Therefore, we use the same $R$ and $L$ for different instant video clips. Define a video downloading schedule as $S = \{(v_1, t_1, l_1), (v_2, t_2, l_2), ..., (v_k, t_k, l_k)\}$, where a tuple
the downloading of this video misses the deadline for the continuous playback, which calculates how many time slots back discontinuity naturally reflects the user experience for a discontinuities: where as well as maximize the efficiency, which is to minimize the further discuss more specific assignments of w
Discontinuity consumption by their corresponding maximum values, where [0
our objective is thus to minimize the playback discontinuity, as well as maximize the efficiency, which is to minimize the total monetary cost:
\[ C_{\text{total}} = \sum_{(v_i, t_i, t_j) \in S} \sum_{t = t_i}^{t_j + 1} C(t) \]
and the total energy consumption:
\[ E_{\text{total}} = \sum_{(v_i, t_i, t_j) \in S} \sum_{t = t_i}^{t_j + 1} E(t) \]
It is easy to see that these objectives may contradict with each other, as downloading more portions of the playlist can reduce the playback discontinuity but will also inevitably consume more energy and may increase the monetary expense. We thus adopt the following linear combination form to align them together:
\[ p \cdot \text{Discontinuity} + q \cdot \frac{C_{\text{total}}}{C_{\text{max}}} + r \cdot \frac{E_{\text{total}}}{E_{\text{max}}} \tag{1} \]
where p, q and r are the parameters to assign different weights to the three goals. As Discontinuity is a ratio between [0, 1), we also normalize the monetary cost and the energy consumption by their corresponding maximum values, where \(C_{\text{max}}\) is the maximum total cost of the case that all the videos in the playlist are downloaded through 3G links, and \(E_{\text{max}}\) can be obtained similarly. We then have the following theorem:

**Theorem 1.** The decision version of the modeled generic downloading scheduling problem is a NP-complete problem.

**Proof:** We first state the corresponding decision problem: given all the required parameters, is there a schedule with the objective value given by Equation \(\text{[1]}\) at most \(M\)? Given an instance of this decision problem, a certificate that it is solvable would be a specification of the downloading schedules for each video. We could then check each video’s playback discontinuity, downloading monetary cost, energy consumption and whether the objective value is no greater than \(M\).

We now show that the Subset Sum problem is reducible to this decision problem. Consider an instance of Subset Sum with numbers \(l_1, \ldots, l_n\) and a target \(W\). To construct an equivalent scheduling instance, one may be struck initially by the fact that we have so many parameters to manage. The key is to sacrifice some of the flexibility, producing a simpler “skeletal” instance of the problem that still encodes the Subset Sum problem. Assume that \(p/q\) and \(p/r\) are large enough so that any increase of discontinuity will increase the objective value. Let \(B_{\text{wifi}} \geq B_{3G} \geq R\), and thus, for each video, the playback discontinuity can be guaranteed to be zero as long as it is scheduled for downloading no later than its playback. We also assume that for any time slot, \(C_{3G} > C_{\text{wifi}}, E_{3G} > E_{\text{wifi}}\). Given the type of connection, the monetary cost and energy consumption are fixed for each time slot. Therefore, the objective value given by Equation \(\text{[1]}\) can be taken as a function of number of time slots for a certain type of connection (defined as \(O_{\text{wifi}}(m)\) and \(O_{3G}(m)\), where \(m\) is the number of time slots).

Given videos 1, 2, ..., \(n\) and video \(k\) has a watch duration of \(u_k\). Define \(l_k\) as the time slots taken to download video \(k\) \((l_k = u_k \cdot R/B, \text{where } B \in \{B_{\text{wifi}}, B_{3G}\})\). The corresponding full Subset Sum instance is to find whether there is a nonempty subset that sum to \(W\) given a set of integers \(l_1, l_2, \ldots, l_n\). Assume that there are \(W\) time slots of WiFi links before the first video playback, and during the playback from video 1 to video \(n\), only 3G connection is available. Given any value of \(M\), \(W\) can be determined such that \(O_{\text{wifi}}(W) + O_{3G}(R \cdot \sum_{k=1}^{n} u_k - W \cdot B_{\text{wifi}})/B_{3G}) = M\).

Now consider any feasible schedule to this instance of the decision problem. To limit the objective value no greater than \(M\), there must not be any idle time slot of WiFi downloads before the first video playback according to the definition of \(W\). If the first \(W\) WiFi slots have not been fully utilized, there must be some videos that have not been downloaded or more than appropriate amount of videos are downloaded through 3G connection instead of using WiFi, both of which would cause the objective value greater than \(M\) according to our assumptions. In particular, if videos \(i_1, \ldots, i_k\) are the ones that are scheduled to be downloaded in the first \(W\) WiFi slots with durations \(\hat{l}_{i_1}, \ldots, \hat{l}_{i_k}\), then the corresponding numbers \(\hat{l}_{i_1}, \ldots, \hat{l}_{i_k}\) in the Subset Sum instance add up to exactly \(W\).

Conversely, if there are numbers \(\hat{l}_{i_1}, \ldots, \hat{l}_{i_k}\) that add up to exactly \(W\), then we can schedule these intervals to be download videos in the first \(W\) WiFi time slots, and the
remainder are downloaded during the playback, which is a feasible solution to the decision problem. This finishes the proof that the decision version of the original optimization problem is NP-complete.

We now present a practically effective solution that takes the unique structure of the problem into account. It divides the problem into two cases depending on whether the scheduling part is before or during a user watches the playlist. The former demands a pre-fetching scheduling, whereas the latter demands a watch-time download scheduling.

A. Pre-fetching Scheduling

We first consider the pre-fetching scheduling. It happens well before the user starts watching the playlist, i.e., without a stringent time constraint; hence we can offload the mobile traffic to the wireless network to reduce the transmission cost. The objective is to find a schedule $S_{pf}$ to pre-fetch the videos, subjecting to the following constraints:

1) **Storage Constraint:**
$$\sum_{(i_j, i_j, i_j) \in S_{pf}} B(t) \leq \text{StorageSize};$$

2) **Cost Constraint:**
$$\forall (i_j, i_j, i_j) \in S_{pf}, \sum_{t=i_j}^{i_j+i_j-1} C(t) = 0.$$

The storage constraint ensures that the total amount of pre-fetched video will not exceed the limited local storage. And the cost constraint implies that the pre-fetching is performed only through WiFi links. Note that the playlist $V$ during the pre-fetching may only be a subset of that during the watch-time, as the pre-fetching occurs before the video watching and new videos may be added to the playlist after the pre-fetching, which will be handled by our watch-time download scheduling to be discussed in the next subsection.

Given that the user behavior during the video watching is unknown at this stage (nor the watching duration $U$), we thus introduce $P = \{p_1, p_2, ..., p_n\}$ to denote the user preference on each video in the playlist, which can reflect the potential lengths of the watch durations. In practice, $P$ can be determined in different ways: for example, it may be a specific metric such as video popularity, video timeliness or the social distance between the publisher (the user who reposts the video) and the consumer (the user who may watch the video); it can be also modeled as a function which considers all of the above factors. In this work, we use the video popularity as the metric of user preference. In addition, we introduce parameter $\alpha \in [0, 1]$ to represent the aggressiveness of the pre-fetching. In particular, for each instant video clip, we pre-fetch $\alpha$ of the total video, instead of downloading the whole clip. Given the limited storage on mobile devices, pre-fetching a part of the video, not all of it, indicates that more videos can be pre-fetched, and thus the start-up delay can be greatly reduced, which also makes the pre-fetching scheduling can better work with the watch-time download scheduling. The playback discontinuity of a single video $v_i$ can then be rewritten as
$$\text{discontinuity}(v_i) = 1 - \frac{1}{\alpha \cdot L} \cdot \frac{pf(v_i)}{R},$$
where $pf(v_i)$ defines how much of this video has been pre-fetched:
$$pf(v_i) = \sum_{(i_j, i_j, i_j) \in S_{pf}, t=\hat{t}_j} B(t).$$

As stated in the generic formulation, the next step is to find a proper assignment of $w_i$. For this subproblem, we define $w_i$ as
$$w_i = \frac{1}{\sum_{k=1}^{n} p_k \cdot \text{discontinuity}(v_k) \cdot p_i \cdot \text{discontinuity}(v_i)},$$
which considers both the user preference for $v_i$ and its current playback discontinuity. It is worth noting that $w_i$ decreases more of $v_i$ has been pre-fetched. The intuition is that, given the batch view behavior, it is not reasonable to allocate all the resources to a tiny portion of extremely popular videos. In practice, the first several units of a video clip are usually requested with a much higher probability than its later part. Together with the pre-fetching aggressiveness $\alpha$, this assignment of $w_i$ allows us to pre-fetch more instant video clips with the video preference still being considered.

As the monetary cost for WiFi links is usually negligible, our goal here is to minimize $\text{Discontinuity}$ with the following form:
$$\text{Discontinuity} = \frac{1}{\sum_{k=1}^{n} p_k \cdot \text{discontinuity}(v_k)} \cdot \sum_{v_i \in V} p_i \cdot \text{discontinuity}(v_i)^2.$$

One may notice that, different from Equation[1] this objective function does not directly involve the energy consumption of pre-fetching. In this formulation, we use $\alpha$ to control the trade-off between the energy consumption and the playlist playback discontinuity. As $\alpha$ gets larger, more videos would be pre-fetched, which causes more energy consumption; on the contrary, if $\alpha$ is small, only a small portion of videos will be pre-fetched, and thus little energy is consumed. Therefore, the above objective function can still represent the overall performance.

This pre-fetching scheduling subproblem is a variation of the knapsack problem with a total weight limit
$$W = \min(\text{StorageSize}, \sum_{t \text{ such that } C(t)=0} B(t)),$$
where an object is one time slot length of video playback, and its value is the amount of decrease of $p_i \cdot \text{discontinuity}(v_i)^2$ after pre-fetching one more time slot, if the object belongs to video $v_i$. It is easy to see that while the weight of each object is the same, the value changes as the decisions are made, i.e., as one object of video $v_i$ is downloaded, the value of all the remained objects of video $v_i$ decreases as now $\text{discontinuity}(v_i)$ decreases. To this end, we design a
greedy algorithm by searching and downloading one object that currently has the greatest value in each iteration. Recall that in our scenario all the objects have the same weight. Therefore, as our solution can find the optimal result in each iteration, it can actually return the final optimal pre-fetching schedule.

B. Watch-time Download Scheduling

Unlike the pre-fetching, during the watch-time, the video watching durations can be largely determined from the input user gestures. In general, the mobile instant video clip sharing services allow three types of user gestures: click, drag and fling, where the last two gestures can cause screen scrolling. Once an user gesture is given, the following process of screen moving is predetermined. Given the fixed display size of each instant video clip (specifically, the fixed height), the motion of screen scrolling can be modeled and calculated, and the details of the scrolling process can be obtained accurately (e.g., how many videos are present, how long each video will stay in the viewport), which can hardly be done in VoD or video streaming applications. Although different operating systems have different technical details for implementation, the philosophy for animating the screen scrolling is generally the same, which is to gradually decelerate the scrolling speed until it reaches zero if there is no other finger touch detected during the deceleration. We next show how to calculate the video watching durations according to user gestures, by taking the Android OS as an example.

By detecting and collecting relevant information about the user’s finger touch, the initial scrolling speed \( s_0 \) can be calculated as the dragging distance divided by the touch time in the unit of \( \text{pixels/second} \). As the screen only scrolls vertically in mobile video sharing services, we denote the height of each instant video clip as \( h \). By default, the threshold \( s_T \) for the initial scrolling speed to distinguish between a drag and a fling in the Android OS is 50 \( \text{pixels/second} \), which can be scaled under different configurations based on the actual screen resolution.

Dragging: In the case of dragging, the screen scrolling speed will experience a uniform deceleration with the default deceleration \( d = 2000 \text{ pixels/second}^2 \). Given the initial speed \( s_0 \), there will be \( \frac{s_0^2}{2dh} \) video clips covered by this drag gesture. In the deceleration process, for the \( m \)-th video clip showing in the dragging animation, we have

\[
mh = s_0 t_m - d t_m^2 / 2,
\]

where \( t_m \) is the time that the \((m+1)\)-th video clip starts to enter the viewport \((t_0 = 0)\). Solving Equation \(2\) gives us

\[
t_m = (s_0 - \sqrt{s_0^2 - 2mh}/d) (1 \leq m \leq \lfloor s_0^2/2dh \rfloor).
\]

Flinging: If a fling is detected, the deceleration will change with the scrolling speed. Given the scrolling speed \( s \), the total fling duration \( T(s) \) and the total fling distance \( D(s) \) can be calculated by using the following equations:

\[
I(s) = \log(0.35 \cdot s/(Fric \cdot P_{COEF})),
\]

\[
T(s) = 1000 \cdot \exp[I(s)/(D_{RATE} - 1)],
\]

\[
D(s) = Fric \cdot P_{COEF} \cdot \exp[D_{RATE}/(D_{RATE} - 1) \cdot I(s)],
\]

where \( D_{RATE} = \log(0.78)/\log(0.9) \), \( Fric \) denotes the parameter of the friction with the default value as 0.015, and \( P_{COEF} = G \cdot 39.37 \cdot ppi \cdot 0.84 \). To compute \( P_{COEF} \), \( G \) is the gravity of Earth with a constant value of 9.80665 \( m/s^2 \), 39.37 is used for the conversion between meters and inches, and \( ppi \) denotes the parameter of pixels per inch for the specific mobile device. From Equation \(5\) and \(6\) we can derive

\[
D(s) = Fric \cdot P_{COEF} \cdot (T(s)/1000)^{D_{RATE}},
\]

and

\[
T(s) = 1000 \cdot [D(s)/(Fric \cdot P_{COEF})]^{1/D_{RATE}}.
\]

Given the initial speed \( s_0 \), there will be \( |D(s_0)/h| \) video clips covered by this fling gesture. Assume \( s_m \) is the scrolling speed at time \( t_m \). In the deceleration process, the following equation is also satisfied:

\[
D(s_0) - D(s_m) = mh (1 \leq m \leq |D(s_0)/h|).
\]

By combining Equation \(7\) \(8\) and \(9\) we have

\[
t_m = T(s_0) - T(s_m)
\]

\[
= T(s_0) - 1000 \cdot [D(s_m)/(Fric \cdot P_{COEF})]^{1/D_{RATE}}
\]

\[
= T(s_0) - 1000 \cdot [(D(s_0) - mh)/(Fric \cdot P_{COEF})]^{1/D_{RATE}}
\]

\[
= T(s_0) - 1000 \cdot [D(s_0)/(Fric \cdot P_{COEF}) - mh/(Fric \cdot P_{COEF})]^{1/D_{RATE}}
\]

\[
= T(s_0) - 1000 \cdot [(T(s_0)/1000)^{D_{RATE}} - mh/(Fric \cdot P_{COEF})]^{1/D_{RATE}}
\]

(1 \leq m \leq |D(s_0)/h|)

(10)

It is worth noting that, as the basis of this analysis, Equation \(2\) \(3\) \(5\) and \(6\) are obtained from our analysis of the Android OS source code. Assume that the user will focus on one video at any given time. From Equations \(3\) and \(10\) the watching duration of \(m\)-th video clip showing in the screen scrolling animation can be obtained as \( u_m = t_m - t_{m-1} \). Based on the above analysis, we can now tell how many videos are scrolled by a user gesture and how long each video can stay in the viewport. Therefore, the video watching duration \( U = \{u_1, u_2, ..., u_n\} \) is available once the input user gestures are given. This subproblem of watch-time download scheduling is thus to find a proper real-time download schedule \( S_{rd} \), so as to minimize our objective:

\[
p \cdot \text{Discontinuity} + q \cdot \frac{C_{total}}{C_{max}} + r \cdot \frac{E_{total}}{E_{max}}.
\]

Note that Discontinuity here needs to consider the result of the pre-fetching schedule with the updated
Algorithm 1 Playlist Scheduling

1: while true do
2:   if a video \( v \) is shown on screen then
3:     Remove \( v \) from \( V_p \);
4:   else if a new user input gesture comes then
5:     Set all slots in \( Q \) to empty;
6:     Obtain \( V_w \) based on the user input gesture model;
7:     Sort \( V_w \) in descendant order according to \( u_1^2 \cdot \text{discontinuity}(v_i) \);
8:     Delete the videos that cannot reduce the objective value if downloaded from \( V_w \);
9:     while \( V_w \) is not empty do
10:        Pick the 1st video \( v_1 \) out of \( V_w \);
11:     while \( v_1 \) is not fully downloaded do
12:         if there exist available slots in \( Q \) then
13:             Update \( Q \) to assign the closest-to-deadline slot to \( v_1 \);
14:         else
15:             break;
16:         end if
17:     end while
18:     end while
19:     if there are available WiFi slots in \( Q \) then
20:         Update \( Q \) to move later scheduled downloadings forward to fill the WiFi slots;
21:     end if
22:     end if
23:   else
24:     Update \( V_p \) for newly arrived videos;
25:     if cache is not full and WiFi is available then
26:         Search \( V_p \) to find the video \( v \) with the largest decrease amount of \( p_i \cdot \text{discontinuity}(v_i)^2 \) assuming one more unit of the video is pre-fetched;
27:     end if
28:     Update \( Q \) to schedule downloading one unit of \( v \);
29:   end if
30: end while

\( \text{discontinuity}(v_i) \):

\[
\text{discontinuity}(v_i) = 1 - \frac{1}{\min(t_i, L)} \cdot \sum_{t \in \{ \sum_{k=0}^{i-1} u_k, \min\{ \sum_{k=0}^{i-1} u_k + k \} \}}^{\sum_{k=0}^{i-1} u_k + L} \min(t_i + t_{i+1} - 1, L) \]

\[
\Pi \left( \sum_{(v_j, t_j, t_j') \in S_{rd}, v_j = v_i, t_j' \leq t} B(t) + pf(v_i) \geq (t - \sum_{k=0}^{i-1} u_k) \cdot R \right),
\]

where the amount of the video \( v_i \) that has been pre-fetched \((pf(v_i))\) is also considered in the calculation. For the watch-time downloading subproblem, we define \( w_i \) as

\[
w_i = \frac{1}{\sum_{k=1}^{i} u_k^2},
\]

which emphasizes the importance of watch durations.

The problem is essentially to find a best trade-off between playback discontinuity, monetary cost and energy consumption during the watch-time. Given the objective function, we are able to tell whether downloading one unit of \( v_i \) is beneficial to the overall result. Therefore, only those videos that can reduce the objective value if downloaded need to be considered for scheduling. As mentioned in the generic problem, videos with higher watching durations have higher impacts on the objective function, which should be scheduled for downloading with higher priorities. To this end, we propose a heuristic that always tries to download one time slot of the video \( v_i \) that has the highest value of \( u_1^2 \cdot \text{discontinuity}(v_i) \) in the unscheduled set, and schedules its downloading interval as late as possible (i.e., closest to but still before its playback deadline) so as to only introduce the minimal impact on other videos to be scheduled later. If there still exist available WiFi downloading slots after the scheduling finishes, we move the video downloadings scheduled in the later time slots forward to these WiFi slots so as to fill all the time slots with lower cost.

We integrate and summarize our solutions for both subproblems in Algorithm 1. \( V_p \) is the set of videos that are considered for pre-fetching, which can be initialized as the whole playlist. \( V_w \) is the set of videos that are scheduled during watch-time according to the dragging/flinging model given an user gesture input. \( Q \) is the schedule queue, which denotes whether each time slot is available or assigned to download one unit of a video. We analyze the time complexity of Algorithm 1 by examining the two parts that solve the the corresponding subproblems, respectively. For the watch-time downloading subproblem, the first part of our algorithm (line 2-22) first sorts \( V_w \) accordingly \((O(\log |V_w|))\), and then search \( Q \) for a proper schedule for each video \((O(|V_w|\log |Q|))\). Although this part is executed every time a user gesture is detected, given that any user gesture can only affect a limited number of videos that will show on the screen, \(|V_w|\) is hence quite small, and so is the searching space in \( Q \), which implies that our algorithm can work efficiently during the watch-time. For the pre-fetching subproblem, the second part of our algorithm (line 23-28) searches for the next video to pre-fetch. Assume that each video is divided into \( M \) units. The searching process will thus be performed at most \(|M||V_p|\) rounds. As each round of the search process only has the complexity of \( O(|V_p|) \), its efficiency is also acceptable especially given that the pre-fetching happens well before the video watching and is often with a much longer time span.

IV. PERFORMANCE EVALUATION

In this section we evaluate our proposed approach by conducting extensive trace-driven simulations, which exploit the real-world data of Vine videos and the user gestures recorded during the Vine watching experience.

A. Data Sets and Traces

1) Vine Video Measurement: Our measurement on Vine is mainly on the video uploading rate and the video popularity
in terms of the number of reposts. In particular, we develop a customized crawler that imitates the behaviors of the mobile application by sending the same requests and substituting the header information with the headers of mobile browsers. The collected data set consists of the videos from popular sections and recent sections in Vine’s user channels during the week of November 21-27, 2013, where we recorded the exact time that each instant video clip was uploaded to the user channels. We further track how many times each video has been reposted as the metric of popularity. Figure 4 plots the video popularity versus its rank for the two types of videos, in which videos from recent sections exhibit a higher skewness. The average numbers of hourly video uploads during a day for popular and recent sections are shown in Figure 5.

2) User Gesture Traces: As we have emphasized in previous sections, Vine-like services introduce the revolutionary change of video watching mode for mobile users. To obtain the real-world user gesture traces that directly reflect the unique user experience of the new generation of mobile instant video clip sharing services, we implement a mobile app on Android platforms to record the exact user touch events. We recruit 10 volunteers to watch Vine videos using the official client on Android under 3G connections and WiFi connections, respectively. Each experiment is conducted around 5 minutes. We plot two important characteristics of user behavior from the collected traces: Figure 6 shows the probability density distribution of the user gestures’ inter-arrival time, and its curve fitting result; and Figure 7 plots the histogram of the measured initial scrolling speed of the triggered flinging animations. The user gesture traces are used to simulate video watch durations by applying the dragging/flinging model.

B. Methodology

We conduct extensive simulations to evaluate our proposed pre-fetching and watch-time downloading solution with the first-hand measurement results as well as data traces from Vine and the mobile users who participated in our data collection experiments. For comparison, we implement another two video downloading schemes. **Sequential Downloading** (SeqD) downloads all the videos according to their order in the playlist, and disregards all the user actions, which takes the playlist with instant video clips as a single long video, and is the most aggressive downloading scheme with the least flexibility. **Next-one Downloading** (NextD) always attempts to download the next video that will enter the viewport during the current playback, which emulates the current caching strategy of Vine. If the current watching duration is not long enough to download an instant video clip, the next playback will be interrupted. Unlike our proposed solution, both of SeqD and NextD are not capable to obtain and utilize the information of watching durations, which implies that they always try to download the entire target video rather than a fraction of it. In addition, we use playback discontinuity, monetary cost, energy consumption as the three metrics in our evaluation, which directly relate to the three goals of our optimization problem. We randomly introduce 20 video watching events of Vine during the daytime from 9 a.m. to 9 p.m. in the week-long data set, emulate the mobile user’s video watching behaviors by using the user gesture traces, and produce the average results. Each video watching event consumes 50 videos from popular sections and 150 videos from recent sections. We assume that, during the watch-time only 3G links are available with 1 MB/s bandwidth, and WiFi is available for pre-fetching once every hour. The application local storage is set as 100 MB, which is about half of the total amount of video consumption in one watching event, given the file size of an instant video clip is around 1 MB. The monetary cost model is 1 dollar per 100
goals in our objective function, two of which are designed
C. Impacts of MB traffic. The energy model is adopted from [10].

Fig. 8. Impacts of $p$, $q$ and $r$

MB traffic. The energy model is adopted from [10].

C. Impacts of $p/q$ and $p/r$ Ratios

We first investigate the impact of different parameter settings by varying the values of $p$, $q$, and $r$. As we have three goals in our objective function, two of which are designed for efficiency, we vary the ratio of $p/q$ with $r = 0$, and vary the ratio $p/r$ with $q = 0$, respectively. The results are shown in Figures 8(a) and 8(b) which demonstrate how the playback discontinuity, the cost efficiency and the energy efficiency change with different parameter values. As $p$, $q$ and $r$ only affect our optimization goal during the watch-time, we disable pre-fetching in this experiment, and only focus on studying the proper settings of $p/q$ and $p/r$ for watch-time downloading (WT). Since changing the ratios between $p$, $q$ and $r$ is essentially the trade-off between efficiency and performance, it is not surprising that the plots in Figures 8(a) and 8(b) are very similar. Note that, as we adopt both a linear monetary cost model and a linear energy model, the normalized results for monetary cost and energy consumption are nearly the same when pre-fetching has been disabled. When $p/q$ ($p/r$) is small ($< 0.5$), very limited number of videos have been downloaded and the video playback is severely affected, so as to reduce the monetary cost (energy consumption). On the other hand, if $p/q$ ($p/r$) increases to a certain degree ($> 3$), the video watching experience is optimized but results in a much higher monetary cost (energy consumption). Moreover, there is a small interval near 1.5 in the figure, in which the playback quality is acceptable with a relatively good cost efficiency (energy efficiency). We thus pick $p/q = 1.5$ and $p/r = 1.5$ as the default setting for the remaining evaluation.

It is worth noting that where the best trade-off achieves for the ratio of $p/q$ ($p/r$) may change when different monetary cost model and energy consumption model are applied. Figure 8 implies that we can have close controls over the performance of our watch-time scheduling scheme by varying the values of $p/q$ and $p/r$, which can be flexible and adjusted based on the user needs in real-time. For example, if a user watches Vine videos at home or in office, s/he probably does not care about the cost and wants to maximize the watching experience by setting $p/q$ and $p/r$ to be large values; while, for a user who accesses Vine service with a limited data plan or a low battery, small values should be assigned to $p/q$ and $p/r$, so that the monetary cost/energy consumption for the video watching can be reduced.

D. Impact of Pre-fetching Aggressiveness $\alpha$

We next study how the overall performance of our proposed solution changes with $\alpha$, which, as discussed in the previous section, represents how aggressively our pre-fetching (PF) acts. We vary the value of $\alpha$ from 0.1 to 0.9, and compare the major performance metrics of our proposed solution with pre-fetching (WT+PF) and those of watch-time downloading without pre-fetching (WT). We normalize the performance metrics of WT+PF by using WT as the baseline, and plot them in Figure 9. As shown in Figure 9(a) when $\alpha$ grows large, the playback discontinuity decreases gradually, while the monetary cost first decreases and then becomes stable. The reason is that when the pre-fetching becomes more and more aggressive, more videos will be downloaded through WiFi, thus the monetary cost is saved; however, as one cannot accurately predict which video (and which portion of it) will be consumed and the application local storage is also limited, there are still some parts of videos need to be downloaded during the watch-time. At the same time, as more and more videos are pre-fetched, some of the videos, which originally will not be downloaded during watch-time according to the performance-efficiency trade-off, are pre-fetched and make the playback discontinuity continue to decrease. On the other hand, Figure 9(b) shows that the normalized energy consumption keeps growing (from 1.2x to 3.8x) until $\alpha$ reaches 0.5, where the application local storage is used up and limits the amount of pre-fetched videos even if $\alpha$ gets larger than 0.5 (and thus the energy consumption). It is worth noting that here the comparison is only between WT+PF and WT, not WT+PF and SeqD (WT+PF and NextD), where, as will be discussed in the next subsection, the energy saving can achieve as much as over 90%. Figure 9 suggests us setting $\alpha = 0.2$ as the default value, where WT+PF consumes about 2x energy of WT to save around 45% monetary cost and improve the playback discontinuity by over 60%.

E. Performance Enhancement

Mobile users can access the instant video clip sharing services under any circumstances. We next show the overall performance gain of our proposed solution (WT+PF) compared to Sequential Downloading (SeqD) and Next-one Downloading (NextD) under different operating conditions. To simulate different levels of connection quality, we vary the downloading bandwidth from 150 KB/s to 3 MB/s. Figure 10 shows the change in downloading efficiency and playback discontinuity as the bandwidth varies.
shows the results with $p/q = 1.5$ and $p/r = 1.5$, which plots the playback discontinuity in Figure 10(a) and the normalized monetary cost and the normalized energy consumption in Figure 10(b). As we keep $\alpha$ fixed in this experiment, the monetary cost and the energy consumption exhibit similar patterns due to their linear models, and thus we see overlapping plots for NextD in Figure 10(b). As SeqD naively downloads videos and disregards their playbacks, which no doubt introduces the highest monetary cost/energy consumption, we use it as the baseline, and normalize the costs of the other two approaches. As shown in the figure, our proposed approach (WT+PF) has a very stable performance with different downloading bandwidths, in terms of both playback discontinuity and cost efficiency. On the contrary, the playback discontinuity of NextD and that of SeqD increasing dramatically when the bandwidth is low (< 0.5 MB/s). The reason is that, as the bandwidth becomes lower, it is more and more difficult for these two downloading schemes to finish each downloading before the playback, while our proposed approach can still keep the playback discontinuity at a low level by smartly managing the downloading according to user actions and efficient pre-fetching. On the other hand, the monetary cost/energy consumption of NextD quickly increases after reaching the minimal at 0.5 MB/s. The reason for both NextD and SeqD introducing high monetary cost/energy consumption is that, when the bandwidth is low, both NextD and SeqD keep the downloading link busy almost all the time; whereas if the bandwidth becomes high enough, both of the downloading schemes attempt to download all the videos, especially given the fact that, both NextD and SeqD disregard the video watch duration and download the entire video instead of part of it. Figure 10(b) shows that our proposed approach can save at least over 40% monetary cost and 30% energy consumption, and the cost/energy saving under high bandwidths can be higher than 90%. This result implies that our solution can efficiently offload the traffic from the watch-time downloading to the pre-fetching, and reduce a significant amount of unnecessary watch-time downloads, as it only downloads the part of the video that will be watched given the input user actions. One may notice that the other two caching schemes may achieve the similar performance to our proposed approach in terms of playback discontinuity, when the bandwidth is high enough so that most downloads can be simply finished before the playback deadline. This is because the setting of $p/q$ and $p/r$ asks for a balance between cost efficiency and playback discontinuity. As previously discussed, if a user is more aggressive on the video watching experience, s/he can further increase of the ratio of $p/q$ and $p/r$, e.g., to $p/q = 3.5$ and $p/r = 3.5$. We plot the corresponding results in Figure 11 which shows that our solution can always achieve the best playback discontinuity at diverse bandwidths, and still with huge amounts of (over 90%) cost/energy savings. To sum up, the results demonstrate that, compared to the other two caching schemes, our proposed approach achieves much more stable performance, which is a crucial requirement for practical designs.

V. FURTHER DISCUSSION

In this section, we briefly discuss several open issues that are worth further exploring. The increasing availability of various sensors integrated in mobile devices provides new opportunities for understanding the operating conditions of users and the surrounding environment. First, extracting patterns from mobile users’ daily activities can assist us to find proper locations for pre-fetching. Specifically, for a mobile user, there are a number of locations that s/he would stay as a daily routine, such as office, classroom and home. These routine locations, which normally provide dedicated WiFi connections, can act as regular pre-fetching sites, as the WiFi access is normally free and users may also be able to charge their mobile devices at these places. Besides the free daily routine WiFi connections, mobile users are also exposed to unforeseen wireless networks from time to time (e.g., when having coffee at a cafe with free WiFi), where opportunistic pre-fetching can be performed.

Moreover, understanding and predicting the operating conditions allows us to make full use of the opportunities that would be normally ignored by mobile users. Existing works have been done on energy-efficient mobile context sensing [11], [12]. For example, Acquisitional Context Engine [12] infers user’s current context by dynamically learning relationships among various context attributes. One simple rule that has been learned by ACE can be: if the user is static, WiFi is connected, and the phone is muted, then the user is probably having a meeting, in which case opportunistic pre-fetching can be done without affecting other applications. Further integrating such sensing abilities with a human mobility model based on periodic travels [13] can bring us the chances to make our proposed approach even smarter, i.e., automatically
adjusting parameter values of $p$, $q$ and $r$ according to the currently obtained operating context.

Finally, our work has also touched the interests of some other research fields such as popularity prediction and human-computer interaction. A number of typical solutions of popularity prediction for online contents have been widely studied in the literature, most of which focus on predicting the trend based on time series with regression models [14], [15] and classification models [16], [17]. On the other hand, measurement studies [18], [9] also show that the related statistics of video sharing services may be well fit by certain distributions. In our work, video popularity is used to estimate the level of user interests in the future accesses. Although numerous papers have studied how users express interests by examining and understanding various user behaviors [19], [20], we may push it one step further – predicting future user behaviors based on the potential user interests, where machine learning techniques can be applied.

VI. Conclusion

In this paper, we presented the first initial study on the new generation of instant video clip sharing services enabled by mobile platforms and explored the potentials for its further enhancement. Taking Vine as an example, we closely investigated its mobile user interface, identified and characterized the unique watching behaviors of the new generation mobile video sharing services, namely, batch view, passive view and screen scrolling. We formulated a generic scheduling problem to maximize the viewing experience as well as the efficiency on the monetary and energy costs. We showed that the formulated generic problem is NP-complete and to better solve it, we further divided the problem into two subproblems, specifically, the pre-fetching scheduling problem and the watch-time download scheduling problem, conquered them separately and then developed a general solution for the generic problem. Using extensive simulations driven by the real-world traces, we showed that our solution can significantly improve the viewing experience while still keeping both the monetary and energy costs relatively low.