Enhancing Dynamic-Viewport Mobile Applications with Screen Scrolling

Lei Zhang, *Member, IEEE*, Feng Wang, *Senior Member, IEEE*, and Jiangchuan Liu, *Fellow, IEEE*

Abstract—The pervasive penetration of mobile smart devices has significantly enriched Internet applications and undoubtedly reshaped the way that users access Internet services. Different from traditional desktop applications, mobile Internet applications require users to input via touch screens and view outputs on the displays with considerably limited size. The significant conflict between the limited-size of touch screens and the richness of online media contents widely exists in dynamic-viewport mobile applications, a class of mobile Internet applications that download contents beyond the user's viewing region (referred to as *viewport*). As dynamic-viewport mobile applications usually use HTTP for content downloading, to improve their quality of experience (QoE) and cost efficiency, in this paper, we present a Mobile-Friendly HTTP middleware (MF-HTTP), which can interpret user touch screen inputs and optimize the HTTP downloading of media objects for such applications. We first demystify screen scrolling in mobile operating systems and precisely break down the viewport moving process. We identify the key influential factors for media object downloading and develop an optimal download scheme. Towards building a practical middleware, we further discuss and address the implementation issues in detail. We implement a MF-HTTP prototype based on Android platforms and evaluate the performance of MF-HTTP by conducting concrete case studies on two representative dynamic-viewport mobile applications, namely, web browsing and 360-degree video streaming.

Index Terms—Mobile applications, dynamic-viewport, screen scrolling, middleware

1 Introduction

During the past decade, we have witnessed the pervasive penetration of mobile smart devices such as smartphones, tablets and wearable devices, which significantly enrich Internet applications and improve user experience. Mobile smart devices are predicted to take up over 50 percent of global devices/connections and surpass 4/5 of mobile data traffic by 2021 [1]. Different from traditional desktop applications, in which users interact via input/output devices like large-size displays, keyboards, and mouses, mobile Internet applications require users to input through *touch screens* and allow them to view outputs on the displays of considerably limited size. This distinct feature introduced by mobile hardware interfaces brings both challenges and opportunities to mobile Internet applications. On one hand, the service providers of mobile Internet applications should provide multiple copies of media contents with different resolutions and even multiple versions of application user interface (UI) layouts to fit various sizes of screens on heterogeneous devices. On the other hand, as media contents are usually organized in certain order/layout in mobile Internet applications, it is possible to predict the viewing region (referred to as *viewport* hereafter) given the user inputs and the fixed size of display.

The significant conflict between the limited-size of touch screens and the richness of online media contents requires the mobile Internet applications to download contents way beyond the user viewport. In the mobile applications that host contents beyond (in and out of) the viewports, users have to move their viewports through touch screen interactions to fully access the contents and achieve good experience. In this paper, we identify and term these applications as *dynamic-viewport mobile applications*. Understanding how the viewport moves naturally becomes crucial to optimize the dynamic-viewport mobile applications. Taking web browsing as an example, mobile users usually can only view a limited area of the web page. By tracking user touches, it is possible to identify in which direction the viewport moves and where it stops, as well as the area covered during the deceleration. Conventionally, web browsers download all the elements in the web page by default, as users can easily view the whole web page on a desktop display. However, in the mobile scenario, given the entire screen scrolling process, we are able to tell which media contents need to be downloaded. For instance, in Fig. 1, the user browses the web page and scrolls the viewport from position A to position B. The area bounded by the dashed lines is covered during the deceleration of screen scrolling. In this web browsing event, there is no need to download the images that are entirely out of the scrolling covered area, which does not hurt the user experience as they never appear in the viewport. In this example, we consider a general case for dynamic viewport applications, in which the screen scrolling can happen in two

- L. Zhang is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong 518060, China. E-mail: leizhang@szu.edu.cn.
- F. Wang is with the Department of Computer and Information Science, University of Mississippi, University, MS 38677 USA. E-mail: fwang@cs.olemiss.edu.
- J. Liu is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong 518060, China, and also with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada. E-mail: csljc@ieee.org.

Manuscript received 14 Dec. 2018; revised 12 Nov. 2019; accepted 1 Dec. 2019. Date of publication 18 Dec. 2019; date of current version 4 Mar. 2021. (Corresponding author: Jiangchuan Liu.) Digital Object Identifier no. 10.1109/TMC.2019.2959524

1536-1233 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

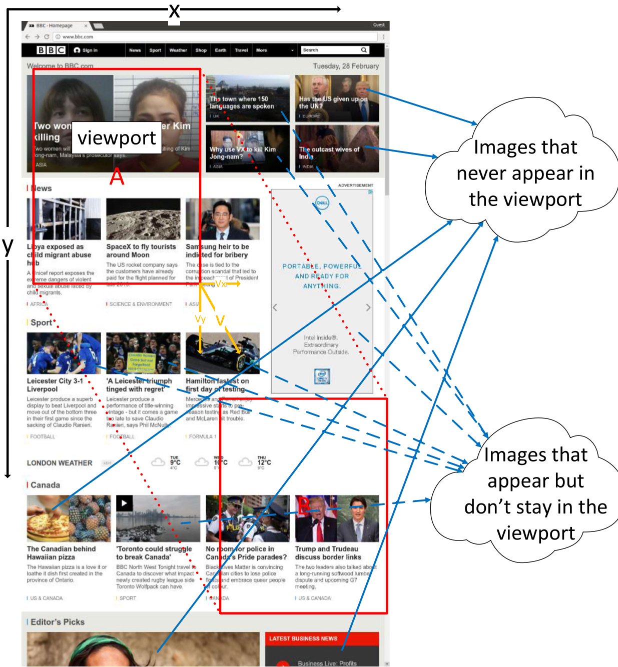


Fig. 1. An example of dynamic-viewport mobile application.

dimensions (vertically and horizontally). In later sections, we will also model and analyze the viewport motions in 2D space with X, Y coordinates. Many websites today have their mobile versions to fit the mobile device's screen width so that users only need to scroll vertically, which can be taken as a special case of Fig. 1.

For dynamic-viewport mobile applications, which now have become mainstream, the current protocol/system designs rarely consider the user-touch screen interactions together with the content organization/represetation in the limited viewport. Despite the limitations brought by the mobile interfaces (e.g., limited displays), we attempt to exploit the opportunities from user-device interactions (e.g., user touches) and add this missing component into the protocol/system design for dynamic-viewport mobile applications. As the screen scrolling animation is mostly affected by user touches, once an input gesture is given based on user touches, the following process of viewport movement is predetermined in mobile operating systems. Therefore, by studying the impacts of user touches on screen scrolling, our work targets to improve quality of experience (QoE) and cost efficiency for the class of dynamic-viewport mobile applications.

As dynamic-viewport mobile applications usually use HTTP for content downloading, in this paper, we showcase the design of Mobile-Friendly HTTP middleware (MF-HTTP), which acts at the application layer, interprets screen scrolling processes on mobile devices by tracking user touch screen operations, and optimizes the downloading of media objects to improve QoE and cost efficiency of such applications. We first demystify screen scrolling philosophy in mobile operating system in depth. With the opportunities of collecting and understanding user touch screen operations, we show how to precisely break down the viewport movement, and identify the media objects involved in the process. By examining the key influential factors for media object downloading, we develop an optimal download scheme. Towards building a practical middleware, we further discuss the implementation details for MF-HTTP, based on which we implement a prototype on Android platforms. We conduct concrete case studies on two typical dynamic-viewport mobile applications, namely, web browsing and 360-degree video streaming, integrate them with our MF-HTTP middleware implementation, and evaluate the performance through extensive experiments. This optimization flow can easily be applied to other protocol/system enhancements for dynamic-viewport mobile applications.

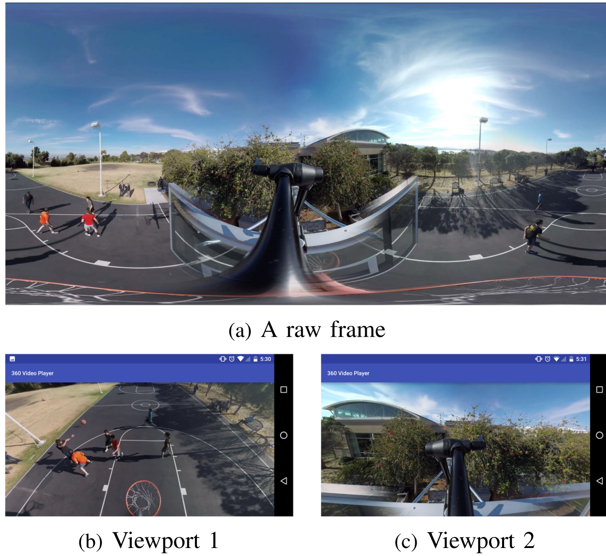
The rest of the paper is organized as follows. Section 2 introduces the class of dynamic-viewport mobile applications, and demonstrates their key features and dominance as a mainstream service type. To enhance dynamic-viewport mobile applications, we present the architecture of the Mobile-Friendly HTTP middleware and reveal its underlying design principles in Section 3. The practical details of MF-HTTP implementation are discussed in Section 4. Section 5 further conducts concrete case studies on two representative dynamic-viewport mobile applications. The performance of MF-HTTP are evaluated in Section 6. Finally, we revisit the related studies in Section 7 and conclude our paper in Section 8.

2 Dynamic-Viewport Mobile Applications

In this section, we first identify the unique characteristics of dynamic-viewport mobile applications, and then show their popular existence. In dynamic-viewport mobile applications, to fully access the contents and achieve good experience, users need to move the viewports through screen scrolling, the unique touch screen interactions on mobile platforms. The operation of screen scrolling in such applications implies that, the viewport is of limited size compared to the contents, and thus the applications need to download their contents beyond the user viewport. The dynamic-viewport mobile applications have been widely seen in real-world, which involve a broad class of mobile Internet applications. Some examples are, to name but a few, mobile web browsing [2], 360-degree video watching [3], and mobile social networking [4]. For mobile web browsing, like the example in Fig. 1, many of today's websites offer web pages that exceed the user viewports. For 360-degree video watching, the raw frames, as shown in Fig. 2a, are downloaded to construct the whole 360-degree panoramic views, while the user viewports can only cover limited areas such as the two snapshots in Fig. 2b and 2c. For mobile social networking, users can only view a very limited amount of social feeds each time in the viewports as shown in Fig. 3, whereas there are usually huge volumes of social contents outside the viewports waiting to be downloaded through user-screen interactions. These examples demonstrate the key features of the daynamic-viewport mobile applications: (1) the users can only access limited contents in their viewports; (2) the applications usually download/prefetch some contents beyond the user viewports; (3) the user viewports are moved/changed through user-screen interactions.

We next take a close look at mobile web browsing to show that dynamic-viewport mobile applications have become a mainstream service type. We examine the Alexa's top 50 global websites [5] on a typical mobile device such as

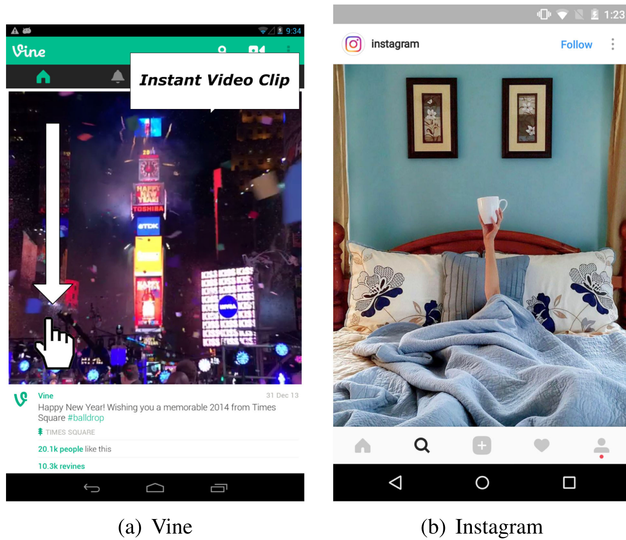
Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.



(a) A raw frame

(b) Viewport 1 (c) Viewport 2

Fig. 2. 360-degree video watching application.



(a) Vine (b) Instagram

Fig. 3. Mobile social networking application.

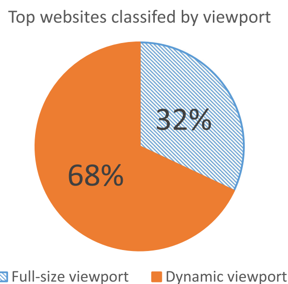


Fig. 4. Top 50 websites' viewport distribution.

Nexus 6 phone. Fig. 4 shows that 68 percent of the top websites belong to the category of dynamic-viewport, while only 32 percent of the websites have full-size viewports. It should be noted that all of the websites with full-size viewports are search or login pages. We further check the normalized viewport size (the ratio of viewport size to web page size) for the websites with dynamic-viewports. Fig. 5 plots the empirical

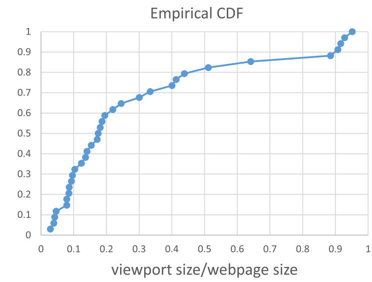


Fig. 5. CDF of normalized viewport size for the top websites with dynamic viewports.

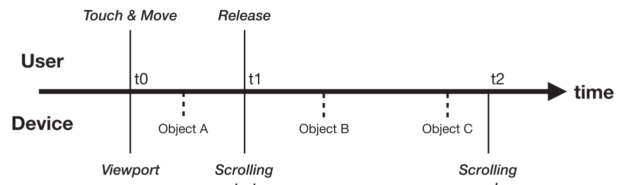


Fig. 6. An example of screen scrolling process.

CDF of the normalized viewport size. For nearly 60 percent of the web sites with dynamic-viewports, the user viewport is smaller than 20 percent of the whole web page. Only a small portion (less than 10 percent) of these websites have comparable user viewports (greater than 90 percent) to the web pages, most of which are also search or login pages. It is worth to mention that, many websites have web pages with indefinite length. For example, when browsing shopping sites or social networking sites, users can always scroll down to view more related items or social feeds. Even for the search or login pages that have full-size viewports, the websites become dynamic-viewport once the search engines return the search results or the contents are returned after completed logins. All these observations further confirm the dominance of the dynamic-viewport applications.

3 MF-HTTP: Architecture and Design

In most of dynamic-viewport mobile applications, HTTP is widely adopted for content downloading due to its simplicity, readiness and ease to use [6]. We thus propose a Mobile-Friendly HTTP middleware (MF-HTTP), acts at the application layer, interprets screen scrolling processes on mobile devices by tracking user touch screen operations, and optimize the downloading of media objects to improve QoE and cost efficiency of dynamic-viewport mobile applications. In this section, we introduce the architecture and design principles of MF-HTTP. To present the details of the design, we first investigate the screen scrolling philosophy, taking the Android OS as an example, next we discuss how to identity the content elements that will be covered during the scrolling, and finally we formulate and solve the optimization problem.

3.1 Middleware Architecture

We first illustrate the opportunities from screen scrolling, the unique user–screen interaction on mobile platforms. Fig. 6 shows an example of screen scrolling process. Along the time axis, the solid line segments indicate the user/device actions, and the dashed line segments indicate when the objects enter

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

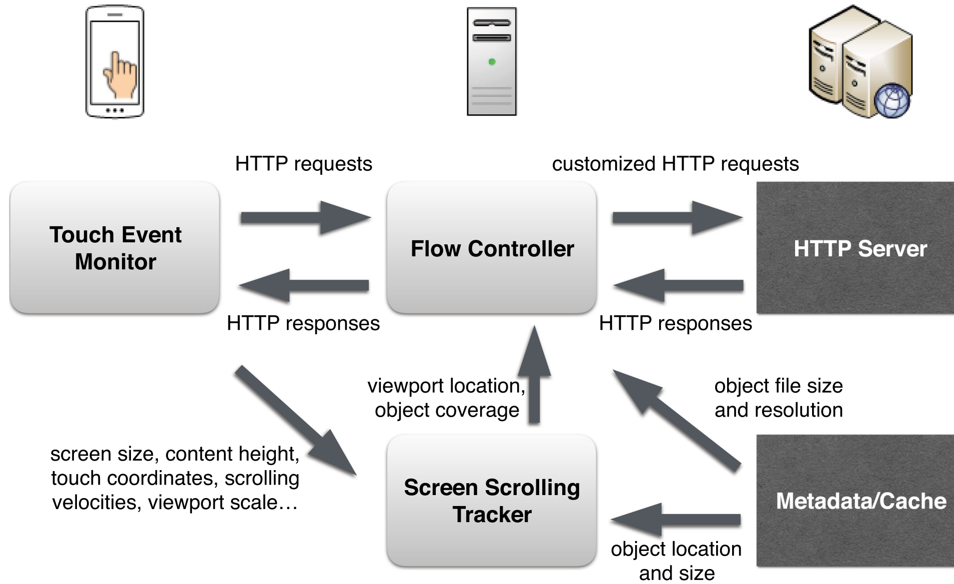


Fig. 7. Middleware architecture.

the user viewport. From $t0$ to $t1$, the viewport changes as the user moves his/her finger. Once the finger is released at $t1$, the following scrolling process is predetermined. Thus, at $t1$, we can accurately calculate the viewport's movement and predict object A's exit and object B and C's entrance in the viewport, which will be addressed in detail in later subsections. Given such information, better download arrangements can be made for the media objects in advance.

Our middleware consists of three modules: touch event monitor, screen scrolling tracker and flow controller, each of which will be elaborated in the following subsections. The main work flow is shown in Fig. 7. The touch event monitor attaches to the target mobile app to collect user touch data, which will be sent to the screen scrolling tracker. The middleware server that holds the other two modules can be either a remote content server or a forward or reverse proxy. With the information of user touch and device configuration, the screen scrolling tracker traces and predicts the viewport's movement. Further, given the location and size of the viewport and those of the media objects, the coverages of the media objects in the viewport can be calculated. Finally, with the full knowledge of the screen scrolling process, the flow controller is able to determine the optimal download policy.

3.2 Touch Event Monitor

The touch event monitor is a light-weight module that collects the device specification and configuration (e.g., screen size, pixel density, viewport scale) as well as the user touch data. As the user touch data can only be obtained from the mobile device, this module is designed to locate on the mobile client and provide interfaces for the mobile app developers to feed the user touch data. The collected information and data are sent to the screen scrolling tracker, which only introduces negligible traffic overhead.

In general, there are 3 types of input user gestures: click, drag, and fling, the last two of which can result in screen scrolling animation. Each gesture can be identified by a series of touch events. By detecting and collecting the information about the user's finger touch and release on the screen, the initial scrolling velocity on x axis (denoted as v_x) and that on y axis (denoted as v_y) can be calculated as the displacement divided by the touch time in two axes, respectively.

3.3 Screen Scrolling Tracker

3.3.1 Scrolling Animation Philosophy

As we will discuss the practical issues for the implementation in the next section, we first investigate the philosophy of animating the screen scrolling for most mobile operating systems, which is to gradually decelerate the scrolling speed until it reaches zero if there is no other finger touch detected during the deceleration.[1] Taking Android OS as an example, we next show how to calculate the viewport movement and the media object coverage during the scrolling process. Given the user touch data, the initial scrolling speed can be obtained as $v = \sqrt{v_x^2 + v_y^2}$. Android OS uses a threshold for the initial scrolling speed to distinguish between a drag and a fling, whose default value is 50 $pixels/second$ and can be scaled under different configurations based on the actual screen resolution.

For *dragging*, the screen scrolling speed will experience a uniform deceleration, which can be easily interpreted given the deceleration parameter and initial speed. As the deceleration of a dragging event is usually short and has very limited impact on viewport movement, we focus on analyzing the case of *flinging*. If a fling is detected, the deceleration will change with the scrolling speed. Given the initial scrolling speed v, the total fling duration $T(v)$ and the total fling distance $D(v)$ (the viewport displacement caused by the fling) can be calculated by using the following equations:

$$l(v) = log[0.35 \cdot v/(Fric \cdot P_{COEF})], \tag{1}$$

$$T(v) = 1000 \cdot exp[l(v)/(D_{RATE} - 1)], \tag{2}$$

$$D(v) = Fric \cdot P_{COEF} \cdot exp[D_{RATE}/(D_{RATE} - 1) \cdot l(v)], \tag{3}$$

1. https://developer.android.com/training/gestures/scroll.html

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

where $D_{RATE} = log(0.78)/log(0.9)$, $Fric$ denotes the friction parameter 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 the 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 pixel density for the specific mobile device. Note that, as the basis of the following analysis, the above equations are obtained from our analysis of Android OS source code.[2,3] Even if the source code cannot be accessed in some cases (for example, a customized OS), the scrolling process should be easy to model, as we only need to fit the relationship between the initial scrolling speed and the scrolling distance, both of which are usually provided in the SDK as available information for mobile app developers.

3.3.2 Viewport Displacement

Assume that, at time t, which denotes the time elapsed since the scrolling starts, the scrolling speed decreases to v'. From Eqs. (2) and (3), we can have

$$D(v) = Fric \cdot P_{COEF} \cdot (T(v)/1000)^{D_{RATE}}. \quad (4)$$

Given $t = T(v) - T(v')$, the viewport displacement at time t can be calculated as

$$d(t) = D(v) - Fric \cdot P_{COEF} \cdot [(T(v) - t)/1000]^{D_{RATE}}. \quad (5)$$

Upon obtaining $d(t)$, we can further calculate the viewport displacement on x and y axis as $d_x(t) = d(t) \cdot \frac{v_x}{v}$ and $d_y(t) = d(t) \cdot \frac{v_y}{v}$, respectively. Note that, as $d(t)$ can have any direction, which is usually the same as (or opposite to) the direction of the user's finger touch movement, $d_x(t)$ and $d_y(t)$ can be either positive or negative.

3.3.3 Objects Involved in Viewport Movement

As the viewport and the rendered media objects (e.g., objects in a web page) are usually rectangular or bounded by rectangular boxes, let (x_p^0, y_p^0) be the original coordinates of the left-top vertex of a viewport, and w_p and h_p be its width and height. The viewport can be then uniquely defined. Similarly, we define (x_i, y_i), w_i, and h_i as the coordinates of the left-top vertex, the width, and the height of a media object i, respectively.

To identify the media objects covered by a scrolling process, we first determine the area covered by the viewport movement. Given the viewport displacement calculated above, the final location of the viewport's vertices can be obtained. As the viewport can move in any direction in a 2-D plane, the mathematical description of the covered area depends on the specific situation. For simplicity, we study the case of $D_x(v) = D(v) \cdot \frac{v_x}{v} > 0, D_y(v) = D(v) \cdot \frac{v_y}{v} > 0$ (other cases can be studied similarly), in which the covered area is surrounded by the boundary consisting of six intersected line segments as illustrated in Fig. 8.

2. https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/widget/Scroller.java

3. https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/view/ViewConfiguration.java

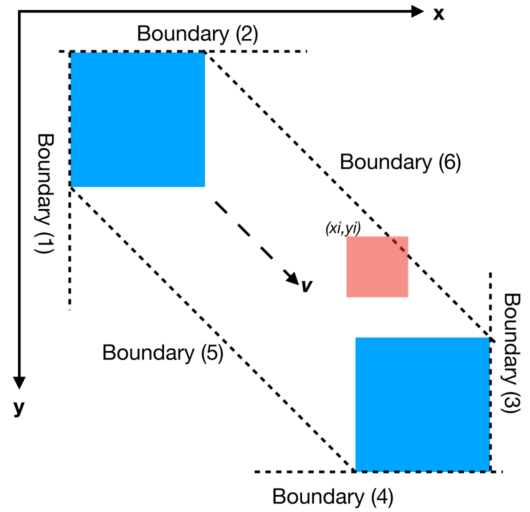


Fig. 8. Viewport movement.

$$(1)\ x = x_p^0;$$
$$(2)\ y = y_p^0;$$
$$(3)\ x = x_p^0 + w_p + D_x(v);$$
$$(4)\ y = y_p^0 + h_p + D_y(v);$$
$$(5)\ y = \frac{D_y(v)}{D_x(v)}(x - x_p^0) + y_p^0 + h_p;$$
$$(6)\ y = \frac{D_y(s)}{D_x(v)}(x - x_p^0 - w_p) + y_p^0.$$

In Fig. 8, the blue rectangles denote the viewport initial and final location, and the dotted lines indicate the corresponding moving boundaries to the six equations above. The area covered by the viewport movement can be determined as the closed area within these boundaries.

To decide whether object i (the red rectangle in Fig. 8) appears in such a bounded area, we check its four vertices to see if it intersects or is located inside. Since the four vertices are correlated, we can further evaluate the case based on the location of one vertex for example the left-top vertex. Specifically, given the boundaries, we can then determine that object i is located in/intersecting the covered area, if (x_i, y_i) meets the following conditions:

$$(1)\ x_p^0 - w_i < x_i < x_p^0 + w_p + D_x(v);$$
$$(2)\ y_p^0 - h_i < y_i < y_p^0 + h_p + D_y(v);$$
$$(3)\ \frac{D_y(v)}{D_x(v)}(x_i - x_p^0 - w_p) + y_p^0 - h_i < y_i < \frac{D_y(v)}{D_x(v)}(x_i + w_i - x_p^0) + y_p^0 + h_p.$$

The three conditions check whether there is a part of the object falls into the area between the three pairs of the parallel boundaries in Fig. 8.

As we are now able to filter the media objects that are involved in a scrolling process, intuitively, the media objects that never appear in the viewport can be omitted for downloading or considered with low priority, which likely causes no difference in user QoE.

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

3.3.4 Object Coverage in Viewport

For those media objects that appear in the viewport, calculating how much area each of them covers is a straight-forward evaluation of its significance to user's multimedia viewing experience. We next show how to compute the coverage of a media object i in the viewport at a given time t. Based on the analysis in the previous subsections, we have that, at time t, the left-top vertex of object i is moved to $(x_p(t), y_p(t)) = (x_p^0 + d_x(t), y_p^0 + d_y(t))$. Similarly, we consider object i appearing in the viewport at time t, if the two following conditions are satisfied:

$$(1)\ x_p(t) - w_i < x_i < x_p(t) + w_p;$$
$$(2)\ y_p(t) - h_i < y_i < y_p(t) + h_p.$$

If object i is identified in the viewport, we can further calculate how much area it covers. Let $s_i(t)$ be the coverage of object i in the viewport at time t, which can be obtained as

$$\begin{aligned} s_i(t) = &[\min(y_i + h_i, y_p(t) + h_p) - \max(y_i, y_p(t))] \cdot \\ &[\min(x_i + w_i, x_p(t) + w_p) - \max(x_i, x_p(t))]. \end{aligned} \tag{6}$$

3.4 Flow Controller

The flow controller determines and executes the optimal download policy for the media objects identified in the last step. We next present the formulation of the download optimization problem, which is solved in this module.

Consider n media objects (such as images in a web page or video segments in a DASH stream) that are involved in a screen scrolling event. To accommodate the heterogeneity of mobile platforms, the service/content providers usually offer multiple versions of media objects, e.g., images/video segments with different qualities. Assume that each object $i \in [1, n]$ have m versions ordered increasingly by resolution. Let t_i be the time when object i first appears in the viewport. Assume that the media objects are indexed based on the order in which they enter the viewport, which implies $t_1 \le t_2 \le \cdots \le t_n$. Let $B(t)$ be the available bandwidth at time t and $f_{i,j}$ be the file size of object i with resolution r_j ($j \in [1, m]$). We further define the cost function as $c(f_{i,j})$, which denotes the cost of download with the given file size. We use $k_{i,j} \in \{0, 1\}$ to denote the download policy for the given object, where the binary variable $k_{i,j} = 1$ indicates the object i of version j will be downloaded, and $k_{i,j} = 0$ otherwise.

3.4.1 Performance Metric Models

We propose two metric models to evaluate the performance gain as well as the download cost for a media object, namely, the QoS model and the cost model.

In practice, user QoE is a subjective metric affected by many factors, and thus it is difficult to model for a broad class of applications such as the dynamic-viewport applications. The actual user QoE can be determined based on the features of a specific application and evaluated accordingly as shown in later case studies. Here our generic QoS model attempts to evaluate the quality of content in the dynamic-viewport applications. Based on Section 3.3.4, object i covers a faction $\frac{s_i(t)}{S}$ of the viewport at time t, where S is the area of the viewport. The quality of content is not only reflected by a media object's coverage and resolution, but also depends on how long the object stays in the viewport. Following this intuition, our QoS model consists of two parts. The first part $Q1(i, j)$ weights the object based on its coverage during the screen scrolling, which can be calculated as the normalized integral of $s_i(t)$ in discrete time with resolution r_j

$$Q1(i, j) = \frac{1}{T(v)} \frac{r_j}{r_m} \sum_{t=1}^{T(v)} \frac{s_i(t)}{S} = \frac{1}{T(v)} \frac{1}{S} \frac{r_j}{r_m} \sum_{t=1}^{T(v)} s_i(t), \tag{7}$$

where $S = w_p \cdot h_p$. The terms in the denominator are used to normalize $Q1(i, j)$ so that its value is between 0 and 1.

The second part $Q2(i)$ is an binary indicator which checks whether the object appears in the final viewport when the screen scrolling stops

$$Q2(i) = \mathbb{1}_{[s_i(T(v)) > 0]}, \tag{8}$$

where $\mathbb{1}_{[\cdot]}$ is the indicator function.

The QoS metric of object i with resolution r_j is defined as a weighted sum of the two parts defined above:

$$Q_{i,j} = a \cdot Q1(i, j) + b \cdot Q2(i). \tag{9}$$

For simplicity, we set $a = b = 1/2$, so that $Q_{i,j}$ is between 0 and 1, and the QoS score of the object in the final viewport will never be lower than that of the object out of the viewport.

The performance gain comes with a price. The download cost of a object can be obtained from the cost function $c(f_{i,j})$ given the file size $f_{i,j}$. We calculate the normalized cost for downloading object i with resolution r_j as

$$C_{i,j} = c(f_{i,j}) / c_M, \tag{10}$$

where c_M is the highest download cost during the scrolling process. As c_M is reached when all the involved media objects are downloaded at the highest resolutions or the bandwidth is completely consumed, it can be calculated as $c_M = c(\min(\sum_{i=1}^{n} f_{i,m}, \sum_{t=1}^{T(v)} B(t)))$. In general, the cost function should be nondecreasing with respect to increasing file size. We keep the cost model generic so that it can be easily adapted to different practical scenarios. For example, the cost function can be defined based on the communication energy model fitting the practical details such as the tail time in 3G/4G communications [7]. In some other cases, the cost function can be a cut-off line describing the monetary cost for a mobile user who pays a fixed fee for a data plan and pays an extra fee (based on usage) when the data traffic exceeds the limit allowed by the data plan.

3.4.2 Optimization Objective

The goal is to generate the optimal download policy for all the media objects, which maximizes the QoE gain and minimizes the download cost. The objective function can be formulated as

$$\begin{aligned} &\sum_{i=1}^{n} \sum_{j=1}^{m} k_{i,j} (p \cdot Q_{i,j} - q \cdot C_{i,j}) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} k_{i,j} \left(\frac{p}{2} \frac{1}{2T(v)} \frac{1}{S} \frac{r_j}{r_m} \sum_{t=1}^{T(v)} s_i(t) + \frac{p}{2} \mathbb{1}_{[s_i(T(v)) > 0]} - q \frac{c(f_{i,j})}{c_M} \right), \end{aligned} \tag{11}$$

where p and q are the weighting parameters.

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

For the proposed optimization problem, the following two constraints must be satisfied.

1) Each object is downloaded once at most

$$\forall i \in [1, n], \sum_{j=1}^{m} k_{i,j} \leq 1. \quad (12)$$

2) The bandwidth should be enough to download the objects in time

$$\forall i' \in [1, n], \sum_{i=1}^{i'} \sum_{j=1}^{m} k_{i,j} \cdot f_{i,j} \leq \sum_{t=1}^{t_{i'}} B(t). \quad (13)$$

The first constraint ensures that no more than one copy (with a certain resolution) of each object can be downloaded. The second constraint implies that, when any object i' appears in the viewport at time $t_{i'}$, there should be enough bandwidth to download it and all the other selected objects that enter the viewport before it. Given the download policy, the underlying scheduling scheme hinted by Eq. (13) is to schedule the download in the same order that the objects are requested in the application.

3.4.3 Optimal Solution

We solve the formulated optimization problem by converting it to a variation of the 0-1 Knapsack problem. Define the value of object i with with resolution r_j as $v(i,j) = p \cdot Q_{i,j} - q \cdot C_{i,j}$, its weight as $w(i,j) = f_{i,j}$, and the maximum weight capacity as $W(t') = \sum_{t=1}^{t'} B(t)$. The key difference is that, in our problem, $W(t')$ (the available bandwidth till a given time t') varies with time. Define $M(i,l)$ as to be the maximum value that can be attained with weight less than or equal to l using first i items. Inspired by the solution of 0-1 Knapsack problem, we solve the formulated problem by dynamic programming as shown in Algorithm 1.

Algorithm 1. Optimal Solution by Dynamic Programming

1: Calculate $v(i,j)$ and $w(i,j), \forall i \in [1,n], j \in [1,m]$;
2: **for** l from 0 to $W(t_n)$ **do**
3: $M[0,l] = 0$;
4: **end for**
5: **for** i from 1 to n **do**
6: **for** l from 0 to $W(t_i)$ **do**
7: $M_{temp} = M[i-1, \min(l, W(t_{i-1}))]$;
8: **for** j from 1 to m **do**
9: **if** $M[i-1, \min(l - w(i,j), W(t_{i-1}))] + v(i,j) > M_{temp}$ **and** $w(i,j) \leq l$ **then**
10: $M_{temp} = M[i-1, l - w(i,j)] + v(i,j)$;
11: $k_{i,j} = 1$;
12: $k_{i,j'} = 0, \forall j' \in [1,m], j' \neq j$;
13: **end if**
14: **end for**
15: $M[i,l] = M_{temp}$.
16: **end for**
17: **end for**

The algorithm first initializes $v(i,j)$ and $w(i,j)$ according to the definitions. The maximum weight capacity is carefully updated as it increases with larger i. A variable M_{temp} is further introduced to store the temporary maximum value when evaluating the different versions of a media object. Each time M_{temp} is changed, the download policy $k_{i,j}$ is updated accordingly, which takes $O(m)$ operations. The time complexity of Algorithm 1 is $O(nm^2 W(t_n))$. The proposed algorithm returns the optimal result since it does exhaustive search. However, as we solve the problem as a variation of the Knapsack problem, it runs in pseudo-polynomial time. In practice, $W(t_n)$ usually has higher magnitude than n and m, so it may need to be encoded using $\log W(t_n)$ bits. Although this algorithm is executed whenever a user touch event is detected, given that any user gesture can only affect a limited number of media objects for a very short time, n, m, and $W(t_n)$ are most likely to have small values, and thus Algorithm 1 can run efficiently.

4 Middleware Implementation

In this section, we present and discuss the implementation issues for the MF-HTTP middleware.

4.1 Touch Event Monitor

The touch event monitor is implemented on the mobile side. The middleware should introduce least modifications on mobile clients and HTTP servers for dynamic-viewport mobile services. As we need to collect user touch events from mobile devices, integrating the touch event monitor to the client-side software, typically a mobile app, is however inevitable. It thus should be effortless for general mobile app developers to implement and integrate the touch event monitor, which employs simple and standard APIs to collect and transmit data.

The user interface for an Android app is built using a hierarchy of layouts (`ViewGroup` objects) and widgets (`View` objects). Layouts are invisible containers that control how its child views are positioned on the screen. Widgets are UI components that can be displayed on screen, such as buttons and text boxes. The widget that occupies (a part of) the device's screen can listen to and handle user touch events on it. The idea is to find the proper View object class in the application's source code, which can also be provided by developers, and attach this module to the scrollable View objects that display the scrolling effect in response to touch gestures. Next, we override the `onTouchEvent` method of the target View objects to handle touch screen motion events. The customized `onTouchEvent` method focuses on three types of motion events: `ACTION_DOWN`, `ACTION_MOVE`, and `ACTION_UP`, which denote the start, the ongoing process, and the end of a pressed gesture, respectively. When the first two types of motion events are detected, the touch coordinates and the timestamp are reported. When `ACTION_UP` motion events are detected, the initial scrolling velocities on x and y axes are calculated and reported, based on which the input gesture can be identified as a fling or a drag. We further decouple the scrolling animation from the original mobile application to produce a well-controlled scrolling process, by employing the `Scroller` class to animate scrolling over time using platform-standard scrolling physics (friction, velocity, etc.). The corresponding scrolling offsets for both drag and fling events are calculated and sent to the screen scrolling tracker.

4.2 Screen Scrolling Tracker

As the touch event monitor is designed to be as light as possible, the screen scrolling tracker should be able to collect all

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

the related information to user input, hardware platform, metadata of the media objects in dynamic-viewport mobile applications, and provide accurate and timely feeds for the flow controller. Some of these information can only be obtained in runtime, while others should be retrieved in advance, which requires us to carefully identify and process.

Prior to user consuming any mobile Internet services, the screen scrolling tracker requires certain knowledge about the (multimedia) services provided by dynamic-viewport mobile applications. As such knowledge can be hardly collected from client side, we implement the screen scrolling tracker on the middleware server. This module can thus access the related data on the cache of the middleware server, which is generated during the previous usages of the target service from the same user or from other users. If a miss occurs at the cache, it can retrieve the required metadata directly from the server of the target dynamic-viewport mobile application with very low cost. Advanced caching schemes [8], [9] can be applied by the middleware server to reduce the cache misses as well as the caching cost.

During the consumption of mobile Internet services, the screen scrolling tracker maintains a TCP socket connection with the touch event monitor to collect the data related to screen scrolling. First, this module retrieves the device specification and configuration information from the touch event monitor, e.g., screen size, pixel density, initial viewport location, viewport size, viewport scale, platform scrolling physics, etc. Second, the user touch data is constantly transmitted to the module through the TCP socket connection, including touch coordinates, timestamp, velocity, and scrolling offsets along with total duration if a fling is detected. Based on the analysis in Section 3.3, it is able to calculate the viewport locations and object coverages during the scrolling process. Whenever a touch event with a newer timestamp arrives, the emulation of current/unfinished scrolling is aborted.

Sending all user input events away from the mobile users may cause some user privacy issue. To avoid that, we actually do not send all user input events to the middleware. Rather than collecting all the information of user touches (i.e., where and how the user touches the screen), MF-HTTP only requires the information of the scrolling speed when the user releases his/her touch (i.e., how quick user finger leaves the screen). Therefore, besides emulating the viewport movement, the middleware cannot reproduce the exact user touch given the limited information, which can address the user privacy concern in certain degree. In addition, during the system deployment the middleware can be placed at a trustful proxy/server, which can reside in the same (and safe) internal network as the mobile client.

4.3 Flow Controller

The flow controller is also implemented on the middleware server and runs in a separate thread from the screen scrolling tracker. During its execution, it communicates with the screen scrolling tracker by sharing global variables, and collects the related information from the previous downloading sessions.

The flow controller should have certain control over the download of media objects without modifying the content server of the dynamic-viewport mobile application or breaking down the hardcore of the mobile app. To this end, we adopt the *mitmdump*[4] tool, run MF-HTTP as a man-in-the-middle proxy, and redirect the mobile client's HTTP traffic to the middleware server. As the tool offers a powerful scripting API that allows us to control many aspects of HTTP traffic being proxied, we develop a Python script to run with *mitmdump* on the middleware server to identify and handle the HTTP traffic generated by the target mobile multimedia service. By default, the tool's script mechanism is single threaded, and the proxy blocks while script handlers execute, which can easily cause a performance issue as multiple HTTP requests may be initiated simultaneously, e.g., in one web browsing session. We thus modify the script with the `@concurrent` setting, and let MF-HTTP proxy work in a non-blocking mode so that the flow controller can process multiple HTTP requests at the same time. The control of media object downloading is realized by modifying, deferring, or blocking the target HTTP headers, requests and responses.

The flow controller executes the optimization logic presented in Section 3.4. It is worth noting that, our optimization model of MF-HTTP can adapt to various user requirements and different practical scenarios, as the cost function and the weights of performance metrics are adjustable. Moreover, as the inputs, the outputs, and the interfaces employed by MF-HTTP are simple and straight-forward, users of MF-HTTP can design and implement their own optimization logics.

5 Case Studies

MF-HTTP targets to optimize dynamic-viewport mobile applications, a class of mobile Internet applications that can make HTTP downloads outside user viewport. For different applications, the knowledge assumed from the last section can be carefully obtained or bypassed. We next present concrete case studies on two representative applications, web browsing and 360-degree video streaming, and discuss the light and practical adjustments for the MF-HTTP prototype.

The two applications in our case studies involves two major types of multimedia experience and various kinds of user behaviors. From the multimedia experience side, web browsing offers a one-time download-and-view experience, while 360-degree video streaming provides a continuous download-and-view experience. From the user behavior side, the two applications both support touch-based interactions, as well as other types of user inputs, e.g., gyroscope readings from end devices in 360-degree video streaming. To examine different user behaviors, we analyzed the data from a touch-based user behavior dataset for social media browsing [4] and three sensing-based user behavior datasets for 360-degree video watching [10], [11], [12]. As shown in Fig. 9, the touch-and-scroll user behaviors usually have high fling (the specific type of user touches that cause fast screen scrolling) speed in browsing-based applications: 86.1 percent of the flings are over 500 pixel/ms and 45.8 percent of them are over 1,000 pixel/ms, which suggests great optimization opportunities in fast browsing events. On the contrast, the head-turn user behaviors in video watching applications exhibits a more stable pattern: in Fig. 10, the probability of the angular speed being less than $10^{\circ}/s$ is about 62.8 percent, and the probability for less than $30^{\circ}/s$ is around 83.1 percent,

4. https://mitmproxy.org/

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

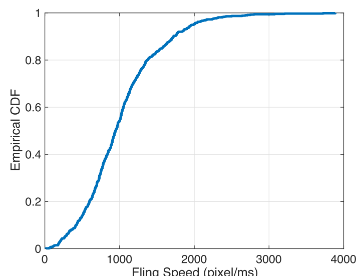


Fig. 9. CDF of user-touch fling speed.

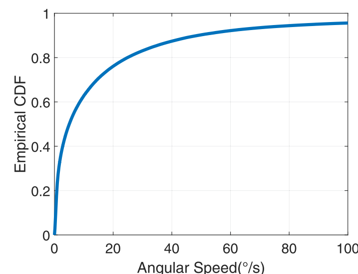


Fig. 10. CDF of head-turn angular speed.

which implies that the dynamics of unpredictable head turns are bounded to certain limits. Given the above considerations, the two target applications are representative for viewpoint tracking and media objects downloading.

5.1 Mobile Web Browsing

For mobile web browsing, the media objects that are critical to user viewing experience are the images in the web page (the videos are often marked by their thumbnails before being selected to play). Therefore, our scrolling-aware HTTP middleware can be adjusted for the download of images.

5.1.1 For Touch Event Monitor

In this case study, we develop a light-weight web browser based on the `WebView`[5] class from Android API, whose `onTouchEvent` method is customized as presented in the last section. Note that `WebView` share the same rendering engine as Chrome for Android, where both are based on the same code.

5.1.2 For Screen Scrolling Tracker

As the web page layouts and the resource dependencies are usually stable [13], the screen scrolling tracker can collect necessary information about the web page from the middleware server's cache. If the web page has never been requested before, our middleware server starts a *WebDriver*[6] (Chrome) instance and downloads the web page. A reference between web objects' locations and source URLs can be then built accordingly. We use Chrome's developer tool to emulate the web page layout under different screen sizes. Every time the web page is requested, this reference is built and updated, so that the middleware keeps refreshing the information of web page layouts proactively.

The "load-before" relationship between the web page's contents such as HTML, CSS, JavaScript, and image objects, usually referred to as the content dependency [14], is one of the key factors of web page loading optimization. For example, the browser should first download HTML as the root file, and download CSS/JavaScript files next to specify the layouts and the contents. The dependency exists because that HTML, CSS, and JavaScript need to be parsed/executed. This parse/execution process decides what contents (multimedia objects or other HTML, CSS and JavaScript files) to be downloaded and where (multimedia objects) to be displayed in the web page. Therefore, those contents can only be requested and downloaded after parsing/executing the depended HTML, CSS and JavaScript. Although the dependencies between web objects can be profiled using tools such as Wprof [15], we choose to not violate any dependency by obeying the download sequence/order of HTML, CSS and JavaScript. As HTML, CSS and JavaScript files constitute only a quarter of the bytes on the average mobile web page [16], MF-HTTP focus on modifying/skipping the download of the rest multimedia objects (still the majority of the downloaded bytes), e.g., images, among which dependencies rarely exist.

5.1.3 For Flow Controller

As bandwidth is rarely the bottleneck for web browsing [2], we release the bandwidth constraint from the formulated problem in Section 3.4. Rather than modifying the hardcore of the web engine to have fine-grained control over the download of web page objects, MF-HTTP adopts simple but effective approaches. The flow controller is adjusted to execute the following work process. (1) When a web page is requested, as the images' source URLs are already collected, the flow controller maintains a block list of source URLs for the images outside the initial viewport. (2) For each data flow, it checks the header to see if the requested URL is in the block list. If so, it blocks the HTTP request. (3) By receiving the updates of viewport location, viewport displacement, and object coverage from the screen scrolling tracker, the flow controller is able to determine whether an image appears in the viewport in the scrolling process. If the image is never involved in the scrolling, it remains in the block list. For web browsing, the images in the viewport before and after its moving are the most crucial to user QoE. Thus such images in the current viewport or in the final viewport when the scrolling stops are identified and removed from the block list. For the images that appear but fail to stay in the viewport, the flow controller evaluates their values $p \cdot Q_{i,j} - q \cdot C_{i,j}$ as in Eq. (11). The images with positive values are allowed to download, while others with negative values are kept in the block list. (4) Whenever a new user touch event is detected, the flow controller receives the updates from the screen scrolling tracker and reacts in the same logic as described above.

5.2 360-Degree Video Streaming

Different from web browsing, video streaming is bandwidth-sensitive and -intensive, which can also benefit from MF-HTTP. 360-degree videos provide users with panoramic views and create unique viewing experience, which are now popular on major video sharing platforms such as YouTube and Facebook. It is worth noting that, 360-degree videos are commonly seen and consumed from various platforms,

5. https://developer.android.com/reference/android/webkit/WebView.html
6. https://seleniumhq.github.io/selenium/docs/api/py/api.html

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

which do not necessarily require virtual reality hardware to play. In this case study, 360-degree videos are consumed as navigable videos from mobile clients with dynamic viewports. As shown in Fig. 2, the user's viewport is significantly confined by the device's size of display, while the whole raw frame is streamed back with large portions outside the viewport. We next discuss how to enable the key idea of MF-HTTP for 360-degree video streaming.

5.2.1 Modification on Mobile Side

The touch event monitor is implemented and attached to an open source 360-degree video player.[7] In this case, we directly pin the touch event monitor to the player's main View object class, which extends the `TextureView` class from Android API, to handle the touch events and output the user gestures and the scrolling offsets.

It worth noting that for 360-degree video playback on smartphones, current major service providers such as YouTube allow different ways of user interactions: user touch input and gyroscope sensing are both supported. Our proposed MF-HTTP should also be able to adapt to other types of user inputs as long as the required information can be extracted. The essential information needed from the user in MF-HTTP is how quickly the viewport moves in which direction (and where it will stop based on the calculation), which should be easily collected or transformed from the given user inputs such as sensor readings. For instance, gyroscope readings on the smartphone can directly tell the angular speed of the user head turn, which can be used to calculate the viewport moving speed and indicate where it stops for the specific video player. MF-HTTP can then work as normal.

5.2.2 Tile-Based DASH Streaming

Although major video sharing platforms like YouTube have already adopted progressive and adaptive download over HTTP to delivery 360-degree videos, they still largely inherit the delivery scheme from traditional Internet videos, which is apparently inefficient for 360-degree videos and provides no flexibility to adapt to the change of user's Region Of Interest (ROI). An adaptive video streaming technique that can smartly respond to viewport movement is demanded for MF-HTTP. To this end, we adopt the tile-based approach [17], [18] to adapt user's ROI, the viewport.

Bandwidth prediction is a widely existed issue in most video streaming studies. If the bandwidth drops drastically in short time, it may cause playback stall/freeze and trigger rebuffering. Inspired by the start-of-the-art 360-degree video streaming systems [19], [20], our MF-HTTP takes two approaches to mitigate this issue: (1) we use the recent history to predict the short future—predict the future bandwidth for the next one or several video segments based on the observed bandwidth during the download of last one or several video segments; (2) to further tolerate bandwidth prediction errors in dynamic network conditions, we set a damping coefficient $\alpha \in (0, 1]$ and conservatively use $\alpha \cdot predicted_bandwidth$ as the available bandwidth in the flow controller.

7. https://github.com/fbsamples/360-video-player-for-android

5.2.3 Adjustments for 360-Degree Video Watching

As the spherical view for 360-degree videos is built from the rectangular raw frame, we adopt the widely used equirectangular projection [21] as the sphere-to-play mapping scheme, which unwraps a sphere with a radius of r on a 2D rectangular plane with the dimensions of $(2\pi r, \pi r)$. Given the initial field of view (FOV) and the viewport size obtained from the mobile client, the radius of the spherical view can be calculated, which enables the translation between longitudes and latitudes of the sphere and x, y coordinates of the 2D plane. The screen scrolling tracker can then map the viewport to the tiles of the raw video frame.

User behaviors for video watching exhibit distinct patters. In particular, user interest for video contents is usually coherent in one viewing session, and thus users produce much more drag events than fling events if there are any. Given that a DASH segment's duration is usually much longer than a scrolling, instead of interpreting viewport movement, the screen scrolling tracker only keeps a close track of the viewport's current location by monitoring the user drag events. The tiles are thus classified into two categories: tiles that appear in the viewport and tiles that have no overlap with the viewport. In the original formulation, media objects with different resolutions are evaluated and selected separately, which can be simplified here by setting $Q_{i,j}$ to be binary, as the tiles that appear in the viewport should be of the same quality so as to provide better and consistent QoE for video watching. As the design of more sophisticated algorithms specifically for 360-degree video DASH streaming optimization is out of the scope of this paper, therefore, for illustration purpose, here the flow controller adopts the following principle for tile-based 360-degree video DASH streaming: given the available bandwidth, minimize the quality of the tiles that have no overlap with the viewport and maximize the quality of the tiles that appear in the viewport.

To accommodate different types of user inputs, MF-HTTP can work in different modes. For more stable touch-based user inputs, MF-HTTP can work in an aggressive mode by skipping the download of no-show tiles as stated above. On the other hand, for more dynamic sensor-based user inputs that may change frequently and dramatically, MF-HTTP can work in a conservative mode: (1) to avoid missing any tile during the playback, it downloads all the tiles; (2) the tiles are downloaded with different resolutions; (3) the resolution selection is affected by the viewport-staying time/viewing probabilities based on the user inputs—tiles with higher viewing time/probabilities are downloaded in higher resolutions. In this conservative mode, although bandwidth is more evenly utilized for all the tiles (with lower resolutions), the user perceived video quality may not be hurt: when the viewport moves very fast, users often cannot tell the difference in video resolution.

6 Performance Evaluation

We have conducted extensive evaluations to examine the performance of our MF-HTTP middleware for both case studies. We will discuss their results in the following two subsections, respectively.

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

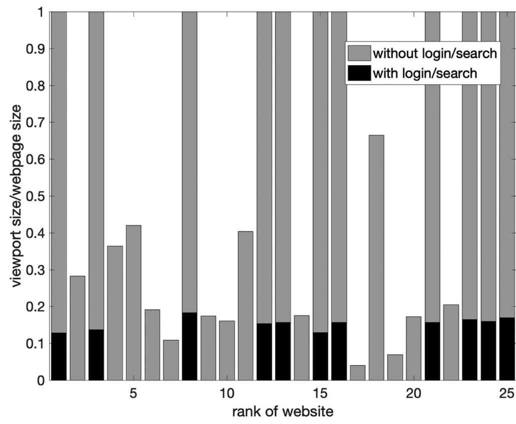


Fig. 11. Normalized viewport size.

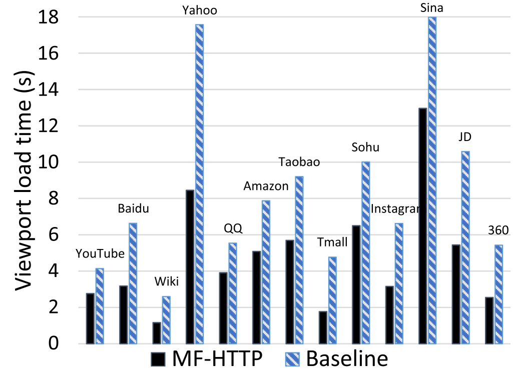


Fig. 13. Viewport load time.

6.1 Experiments for Web Browsing

6.1.1 Test Platforms and Settings

In this subsection, we evaluate the performance improvement of our MF-HTTP middleware for the mobile web browsing case study. We use a Nexus 6 phone running Android 7.0 as the mobile client, and a desktop computer with Intel Core i7-3770 CPU @ 3.40 GHz $\times$ 8 and 16 GB memory running Ubuntu 14.04 LTS as the MF-HTTP middleware. As the touch interface has no dramatic change across different generations of hardware and software platforms, similar experiment results are observed with other phones. The mobile client is connected to MF-HTTP through an IEEE 802.11 WLAN router. Both of the middleware and the router locate in the university campus network, and the network condition is good and stable. We use the browser to access the Alexa's top 25 global websites [5]. Each browsing session consists of default viewport loading followed by a random scrolling touch. We set the weight of cost metric $q = 0$ to maximize the viewing experience. To better trace the loading performance, we add a timer to the browser. We compare the performance of browsing with and without MF-HTTP enabled. The baseline approach downloads all the media objects in the default order with no consideration of the user viewport, which is commonly used in most browsers.

6.1.2 Results

We first check the default viewport size against the web page size, where Fig. 11 shows the ratio of top 25 websites (the gray bars) . In particular, there are 11 websites having full-size viewports and 14 websites having dynamic viewports. Those 14 dynamic-viewport websites stand for more general and various types of websites, from which mobile users can only view a small portion of the whole page (as low as 4.1 percent in the case of Sohu). The 11 websites with full-size viewports are mainly search engines (e.g., Google global and 4 other regions, Microsoft Live) and login pages (e.g., Facebook, Twitter, and Linkedin). We further check their viewport sizes after logging into user accounts or entering search keywords. As shown in Fig. 11 (the black bars), with the social contents and search results returned, the user viewports only cover 15.4 percent of the web pages in average. It is worth noting that, some websites (e.g., YouTube and Yahoo) have pages of varying length, which will always load new contents when users hit the bottom. In theory, these websites can have unlimited length of contents, and thus the impacts of limited-size viewports become even more notable. We further examine the distribution of media object size. In particular, we check the ratio of image height/webpage height and the ratio of image width/webpage width, respectively, and plot the results in Fig. 12. The majority of the images (over 60 percent) have medium sizes, which cover from 5 to 15 percent of the web page's width and from 10 to 25 percent of the web page's height, respectively. There are also over 10 percent large images with width and height greater than 22.6 percent of web page's width and 31.6 percent of web page's height, and 20 percent small images with width and height smaller than 4.9 percent of web page's width and 8.8 percent of web page's height.

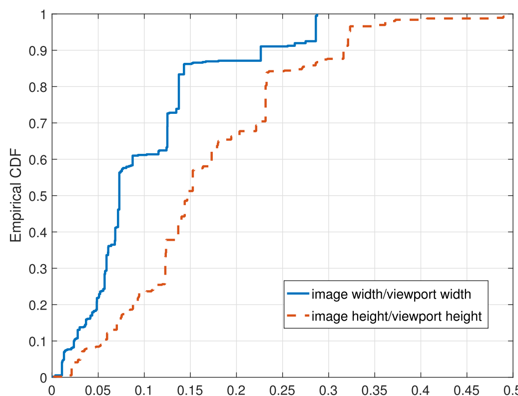
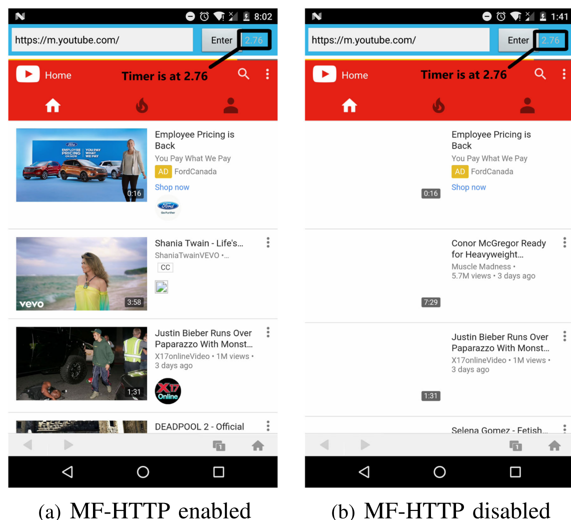


Fig. 12. Image size distribution.

Rather than using page load time, one of the major performance metrics for web browsing, we use a new metric, *viewport load time*, which is the elapsed duration when the viewport is fully loaded. We record the screen of the test smartphone and replay the video to track the loading process as well as the timer. As shown in Fig. 13, MF-HTTP significantly improves the loading performance for the websites with dynamic viewports as it prioritizes the downloads of the objects in the viewport. In average, MF-HTTP reduces the viewport load time by 44.3 percent. The high loading time of some webpages mainly caused by the large number of images and the relatively large image size. Another reason may be that our test browser is developed merely based on the WebView API. The advantage of our implementation is lightweight and having increased control over advanced configuration options, while the disadvantage is that the test browser is less-optimized and lacks some features of fully-developed browsers. The test browser thus has poorer

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.



(a) MF-HTTP enabled (b) MF-HTTP disabled

Fig. 14. Screenshots of two browsing sessions with the same timestamp.

performance than the commercial products such as Chrome, Safari and Firefox. As the experiments for MF-HTTP and baseline were done on the same test browser, the VLT comparison can demonstrate the superiority of MF-HTTP. Fig. 14 further shows two screenshots taken at the same time for two YouTube browsing sessions using different approaches. In this example, MF-HTTP finishes loading the viewport, while the baseline approach still struggles in downloading objects disregarding whether they are in the viewport.

We next examine the amount of traffic generated during the browsing sessions using the two approaches. Fig. 15 shows that MF-HTTP generally requires less data transmissions than the baseline approach, with 15.3 percent traffic saving in average. It is worth noting that, as q is set to be 0, MF-HTTP works in the most aggressive download mode, and only omits downloading the objects that never enter the viewport. Hence, when $q > 0$, more traffic saving can be expected. We further break down the traffic constitution in Table 1. With MF-HTTP enabled, the mobile client sends comparable amount of data (100.34 percent), while receives 16.65 percent less data, which suggests that the traffic saving comes from less media downloads. As the mobile client needs to report user touches to the middleware server, the communication overhead for MF-HTTP is mostly outgoing traffic from the client, which only accounts for a small portion of the total traffic as shown in Table 1. The communication overhead for MF-HTTP is thus negligible (less than 2 percent).

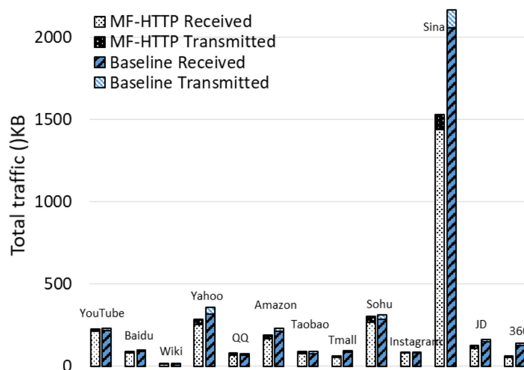


Fig. 15. Amount of traffic.

TABLE 1
Comparison and Proportions of Two-Way Traffic

	MF-HTTP/Baseline	% in MF-HTTP	% in Baseline
Received	83.35%	88.83%	90.63%
Transmitted	100.34%	11.17%	9.37%

6.2 Simulations for 360-Degree Video Streaming

6.2.1 Data Collection

In this subsection, we evaluate the performance improvement of our MF-HTTP middleware for the 360-degree video streaming case study. We obtain three test videos from YouTube[8] at 4 different resolutions/quality levels: 1080s (quality level 4), 720s (quality level 3), 480s (quality level 2), and 360s (quality level 1), where "s" stands for spherical. We recruit 10 volunteers to watch each video on the Nexus 6 phone and modify the 360-degree video player to record user touches during the video watching. Each video watching session lasts for 1 minute. To support tile-based DASH streaming, we use the *GPAC*[9] toolbox to slice and package the 360-degree videos into into 4×4 tiles. We further do a segmentation on the encoded tile-based videos and generate segments with duration of 1 second as well as the MPD files, which are ready to be DASHed. The viewport movement and the resulting tile and rate selection are generated by MF-HTTP based on the collected traces of user touches.

6.2.2 Results

First, we examine the effect of parameter selection by varying the ratio of p/q from 10^0 to 10^{-3}. We calculate the average quality level (QL) of the 3 test videos and normalize their delivery cost (NC) against that of the baseline approach at 1080s. The cost model adopted is a linear model: 10 dollar per 100 MB traffic, which is close to the major mobile operators data add-on prices.[10] Fig. 16 shows the clear tradeoff between two optimization sub-objectives, where higher quality level comes with higher cost. The result suggest an appropriate setting of p/q may be 10^{-1}, where a good balance can be achieved for both sub-objectives.

We next check the bandwidth consumption for MF-HTTP at different resolutions. As shown in Fig. 17, MF-HTTP significantly reduces the bandwidth consumption at each resolution (52 percent average bandwidth saving at 360s, 59 percent at 480s, 60 percent at 720s, and 56 percent at 1080s, respectively), compared to the baseline approach, streaming the whole frame with a fixed resolution without considering the viewport. The result suggests that, with the same video quality, MF-HTTP is much more cost-efficient in terms of data transmissions than the blind downloading. We further plot a sample trace of one video watching session in Fig. 18, which shows that MF-HTTP does not necessarily share network load peaks with the baseline steaming approach. On the other hand, the bandwidth consumption of MF-HTTP is closely affected by the number of tiles that appear in or overlap the viewport, as the valleys of the two curves match in Fig. 18.

8. YouTube IDs of the three test videos are: -xNN-bJQ4vI, rG4jSz_2HDY, wXeKxY3F0sE.
9. https://gpac.wp.imt.fr/home/
10. https://www.telus.com/en/mobility/prepaid/add-ons

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

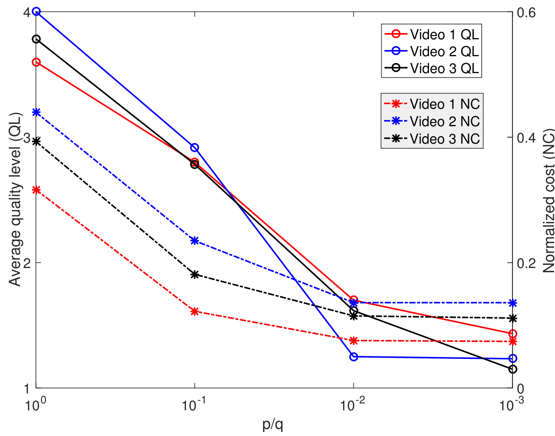


Fig. 16. Effect of p/q.

We further vary the available bandwidth from 250 to 1,000 KB/s to examine the streaming quality of MF-HTTP, and compare its performance with a greedy DASH scheme that maximizes bandwidth usage and streams at the highest possible resolution. Fig. 19 shows how much time (in percentage) the test videos are played at different resolutions using two streaming approaches. It worth noting that, in MF-HTTP, we track user viewport and calculate its location for a known and short future. Since we make no prediction of user viewport during this process, MF-HTTP has full knowledge of user viewport when selecting the tiles and their bit-rates. Therefore, in our MF-HTTP optimization, it does not miss tiles and thus avoids playback stall time as long as the bandwidth can afford the streaming with the lowest resolution. When there is not enough bandwidth for the lowest resolution, playback stall occurs, which is denoted as "NA" In Fig. 19. As shown, MF-HTTP constantly outperforms the greedy DASH scheme under all bandwidth conditions for all test videos. MF-HTTP can maintain good video quality when the bandwidth is low, and it quickly responds to the increase of the bandwidth. This result suggests that MF-HTTP can more efficiently utilize the network resource to focus on downloading the high quality video segments in the viewport.

7 Related Work

A serial of studies have been conducted to optimize web browsing, an application that is largely affected by user viewport. Prior work [2] suggested that client-only approaches have significant limitations for mobile users: caching [22] web contents does not remove the true bottleneck of web page

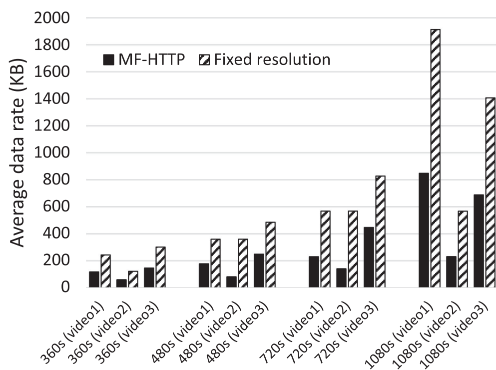


Fig. 17. Bandwidth consumption with fixed resolution.

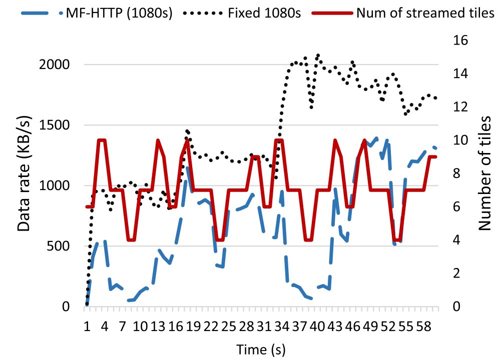


Fig. 18. A sample trace of one video watching session.

loading–RTT, and predictive prefetching [23] cannot work well either since most of the pages will only be requested once by a user. A recent measurement study [24] showed that only a few web sites have fully deployed HTTP/2 (the state-of-the-art standard in industry) servers, and few of them have correctly realized the new features in HTTP/2, which implies the necessity of research efforts on optimizing web performance. Scheduling network requests is a widely exploited approach to reduce page load time, which is designed base on the dependency between web page elements [14]. Butkiewicz *et al.* [13] proposed KLOTSKI, a system that prioritizes the contents most relevant to the user preference and with least rendering time. By collecting the traces of user gaze fixation during web browsing, Kelton *et al.* [25] examined the focus of user attention and reordered the loading of web objects accordingly. To achieve the best performance-energy tradeoff, Ren *et al.* [26] adopted a machine learning based approach to predict the optimal processor configurations at runtime for heterogeneous mobile platforms.

Video streaming is another killer application influenced by user viewport. The rate adaptation scheme is one of the fundamental research issues for video steaming. By studying the responsiveness and smoothness trade-off in DASH, Tian *et al.* [27] showed that client-side buffered video time is a helpful feedback signal to guide rate adaptation. Instead of constantly predicting future capacity, Huang *et al.* [28] proposed to use simple capacity estimation only in the startup phase and then choose the video rate based on the current buffer occupancy in the steady state. Novel techniques, e.g., deep learning [29] and emerging computing architectures, e.g., edge computing [30], [31], [32] are also adopted to improve the rate adaptation for video streaming. Recently, MPEG DASH standard [33] has included a new Spatial Representation Description (SRD) [34] feature, to support the streaming of spatial sub-parts of a video to display devices, in combination with adaptive multirate streaming that is intrinsically supported by DASH. Following this advance, DASH has been further exploited to stream zoomable and navigable videos [35], virtual reality videos [36], and multi-view videos [37]. For 360-degree video streaming, Qian *et al.* [19] designed a viewport prediction mechanism based on the analysis of user head movement traces to optimize the rate-adaptation, and reworked other related components in the streaming pipeline to further boost the performance against non-viewport-adaptive approaches. He *et al.* [20] identified that viewport prediction error can result in significant video quality degradation, and thus proposed a novel

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

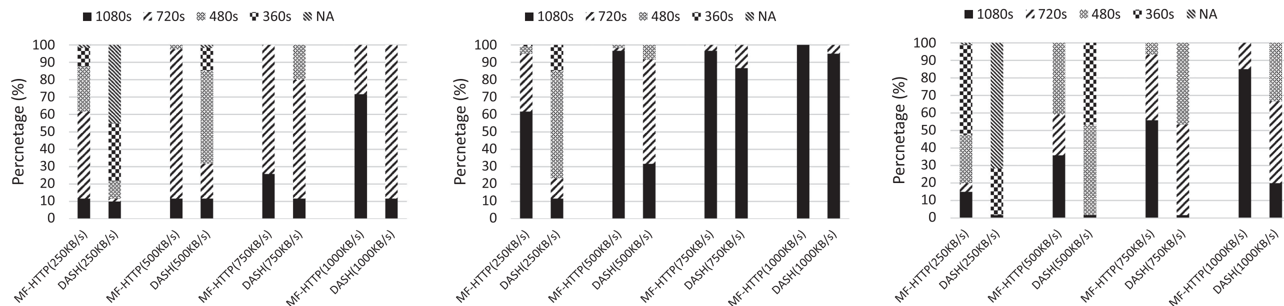


Fig. 19. Video quality constitutions with different bandwidth (Video 1 to 3 from left to right).

tile-based layered approach to adaptively stream 360-degree content on smartphones.

Such mobile smart devices as smartphones, phablets, and tablets, undoubtedly reshape the way that users access Internet services, and therefore attract tremendous attentions from academia. Existing studies have tackled the challenges brought by the intrinsic mobile nature and enhanced network protocols to accommodate seamless mobility [38], [39], inefficient retransmission [40], unstable channel quality [41], [42], [43], and unexpected interference [44], [45] in wireless and mobile networks. Yet, very few of them have attempted to improve network protocols for multimedia applications by utilizing rich interfaces and user interactions on mobile smart devices. To this end, taking the example of the most commonly used network protocol–HTTP, we proposed our middleware design to make it more suitable for mobile multimedia applications. Furthermore, rather than optimizing one specific application, our work strives to enhance dynamic-viewport mobile applications, a class of mobile Internet applications that make HTTP downloads for media contents outside user viewports.

8 Conclusion

In this paper, we presented the Mobile-Friendly HTTP middleware (MF-HTTP) to enhance dynamic-viewport mobile applications that usually use HTTP to download media contents beyond the users' viewing regions on mobile devices. MF-HTTP acts at the application layer and interprets screen scrolling processes on mobile devices by tracking user touch screen operations. Based on the information from the screen scrolling processes, MF-HTTP further optimizes the downloading of media objects to improve QoE and cost efficiency. To achieve this, we first demystified the detailed screen scrolling philosophy in mobile system and showed how to precisely break down the viewport movement. We then identified the key influential factors for media object downloading, and developed an optimal downloading scheme. We further discussed practical issues towards the implementation of MF-HTTP. Finally, we implemented a prototype based on Android platforms and conducted concrete case studies on two typical dynamic-viewport mobile applications, namely, web browsing and 360-degree video streaming, to demonstrate the superior performance of MF-HTTP.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61902257, and in part by Tencent "Rhinoceros Birds" — Scientific Research Foundation for Young Teachers of Shenzhen University.

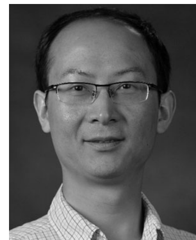
References

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update," 2016-2021 white paper, San Jose, CA, USA, 2017.
[2] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "How far can client-only solutions go for mobile browser speed?" in *Proc. ACM Int. Conf. World Wide Web*, 2012, pp. 31–40.
[3] C. Zhou, Z. Li, and Y. Liu, "A measurement study of oculus 360 degree video streaming," in *Proc. ACM Multimedia Syst. Conf.*, 2017, pp. 27–37.
[4] L. Zhang, F. Wang, and J. Liu, "Mobile instant video clip sharing with screen scrolling: Measurement and enhancement," *IEEE Trans. Multimedia*, vol. 20, no. 8, pp. 2022–2034, Aug. 2018.
[5] The top 500 sites on the web, Accessed: 2017. [Online]. Available: http://www.alexa.com/topsites
[6] Y. Ma, X. Liu, Y. Liu, Y. Liu, and G. Huang, "A tale of two fashions: An empirical study on the performance of native apps and web apps on android," *IEEE Trans. Mobile Comput.*, vol. 17, no. 5, pp. 990–1003, May 2018.
[7] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. ACM Int. Conf. Mobile Syst. Appl. Services*, 2012, pp. 225–238.
[8] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.
[9] J. Li, C. Shunfeng, F. Shu, J. Wu, and D. N. K. Jayakody, "Contract-based small-cell caching for data disseminations in ultra-dense cellular networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 5, pp. 1042–1053, May 2019.
[10] E. J. David, J. Gutiérrez, A. Coutrot, M. P. Da Silva, and P. L. Callet, "A dataset of head and eye movements for 360° videos," in *Proc. ACM Multimedia Syst. Conf.*, 2018, pp. 432–437.
[11] C. Wu, Z. Tan, Z. Wang, and S. Yang, "A dataset for exploring user behaviors in VR spherical video streaming," in *Proc. ACM Multimedia Syst. Conf.*, 2017, pp. 193–198.
[12] W.-C. Lo, C.-L. Fan, J. Lee, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "360 video viewing dataset in head-mounted virtual reality," in *Proc. ACM Multimedia Syst. Conf.*, 2017, pp. 211–216.
[13] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar, "KLOTSKI: Reprioritizing web content to improve user experience on mobile devices," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, 2015, pp. 439–453.
[14] R. Netravali, A. Goyal, J. Mickens, and H. Balakrishnan, "Polaris: Faster page loads using fine-grained dependency tracking," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, 2016, pp. 123–136.
[15] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with WProf," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, 2013, pp. 473–485.
[16] V. Ruamviboonsuk, R. Netravali, M. Uluyol, and H. V. Madhyastha, "Vroom: Accelerating the mobile web with server-aided dependency resolution," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 390–403.

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.

[17] N. Quang Minh Khiem, G. Ravindra, A. Carlier, and W. T. Ooi, "Supporting zoomable video streams with dynamic region-of-interest cropping," in *Proc. 1st Annu. ACM SIGMM Conf. Multimedia Syst.*, 2010, pp. 259–270.
[18] M. Xiao, C. Zhou, Y. Liu, and S. Chen, "OpTile: Toward optimal tiling in 360-degree video streaming," in *Proc. ACM Int. Conf. Multimedia*, 2017, pp. 708–716.
[19] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 99–114.
[20] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han, "Rubiks: Practical 360-degree streaming for smartphones," in *Proc. ACM Int. Conf. Mobile Syst. Appl. Services*, 2018, pp. 482–494.
[21] Equirectangular projection, Accessed: 2017. [Online]. Available: https://en.wikipedia.org/wiki/Equirectangular_projection
[22] F. Qian *et al.*, "Web caching on smartphones: Ideal versus reality," in *Proc. ACM Int. Conf. Mobile Syst. Appl. Services*, 2012, pp. 127–140.
[23] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 22–36, 1996.
[24] M. Jiang, X. Luo, T. Miu, S. Hu, and W. Rao, "Are HTTP/2 servers ready yet?" in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1661–1671.
[25] C. Kelton, J. Ryoo, A. Balasubramanian, and S. R. Das, "Improving user perceived page load times using gaze," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, 2017, pp. 545–559.
[26] J. Ren, L. Gao, H. Wang, and Z. Wang, "Optimise web browsing on heterogeneous mobile platforms: A machine learning based approach," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.
[27] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic HTTP streaming," in *Proc. ACM 8th Int. Conf. Emerg. Netw. Experiments Technol.*, 2012, pp. 109–120.
[28] 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," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 187–198, 2015.
[29] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 197–210.
[30] Y. Im *et al.*, "FLARE: Coordinated rate adaptation for HTTP adaptive streaming in cellular networks," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 298–307.
[31] T. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 1965–1978, Sep. 2019.
[32] A. Mehrabi, M. Siekkinen, and A. Yla-Jaaski, "Edge computing assisted adaptive mobile video streaming," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 787–800, Apr. 2019.
[33] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, Apr. 2011.
[34] O. A. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S. Y. Lim, "MPEG DASH SRD: Spatial relationship description," in *Proc. ACM Multimedia Syst. Conf.*, 2016, Art. no. 5.
[35] L. D'Acunto, J. van den Berg, E. Thomas, and O. Niamut, "Using MPEG DASH SRD for zoomable and navigable video," in *Proc. ACM 7th Int. Conf. Multimedia Syst.*, 2016, Art. no. 34.
[36] M. Hosseini and V. Swaminathan, "Adaptive 360 VR video streaming: Divide and conquer," in *Proc. IEEE Int. Symp. Multimedia*, 2016, pp. 107–110.
[37] K. Diab and M. Hefeeda, "MASH: A rate adaptation algorithm for multiview video streaming over HTTP," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.
[38] A. Yadav and A. Venkataramani, "msocket: System support for mobile, multipath, and middlebox-agnostic applications," in *Proc. IEEE 24th Int. Conf. Netw. Protocols*, 2016, pp. 1–10.
[39] L. Zhang, F. Wang, and J. Liu, "Dispersing social content in mobile crowd through opportunistic contacts," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2276–2281.
[40] M. O. Khan, L. Qiu, A. Bhartia, and K. C.-J. Lin, "Smart retransmission and rate adaptation in WiFi," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols*, 2015, pp. 54–65.
[41] A. Aqil, A. O. Atya, S. V. Krishnamurthy, and G. Papageorgiou, "Streaming lower quality video over LTE: How much energy can you save?" in *Proc. IEEE 23rd Int. Conf. Netw. Protocols*, 2015, pp. 156–167.
[42] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Trans. Mobile Comput.*, vol. 14, no. 12, pp. 2516–2529, Dec. 2015.
[43] L. Zhang, D. Fu, J. Liu, E. C.-H. Ngai, and W. Zhu, "On energy-efficient offloading in mobile cloud for real-time video applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 1, pp. 170–181, Jan. 2017.
[44] C.-J. Liu and L. Xiao, "RMIP: Resource management with interference precancellation in heterogeneous cellular networks," in *Proc. IEEE 24th Int. Conf. Netw. Protocols*, 2016, pp. 1–10.
[45] Y. Wu, Y. He, L. P. Qian, J. Huang, and X. S. Shen, "Optimal resource allocations for mobile data offloading via dual-connectivity," *IEEE Trans. Mobile Comput.*, vol. 17, no. 10, pp. 2349–2365, Oct. 2018.

Lei Zhang (S'12–M'19) received the BEng degree from the Advanced Class of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, China, in 2011, and the MS and PhD degrees from Simon Fraser University, Burnaby, BC, Canada, in 2013 and 2019, respectively. He is a recipient of C.D. Nelson Memorial Graduate Scholarship (2013) and Best Paper Finalist at IEEE/ACM IWQoS (2016). He is currently an assistant professor with the College of Computer Science and Software Engineering, Shenzhen University. His research interests include multimedia systems and applications, mobile cloud computing, edge computing, social networking, and Internet of Things. He is a member of the IEEE.



Feng Wang (S'07–M'13–SM'18) received the bachelor's and master's degrees in computer science and technology from Tsinghua University, Beijing, China, in 2002 and 2005, respectively, and the PhD degree in computing science from Simon Fraser University, Burnaby, British Columbia, Canada, in 2012. He is currently an associate professor with the Department of Computer and Information Science, University of Mississippi, University, Mississippi. He is a recipient of IEEE ICME Quality Reviewer Award (2011) and ACM BuildSys Best Paper Award (2018). He is a technical committee member of Elsevier Computer Communications. He served as program vice chair in International Conference on Internet of Vehicles (IOV) 2014, and as TPC co-chair in IEEE CloudCom 2017 for Internet of Things and Mobile on Cloud track. He also serves as TPC member in various international conferences such as IEEE INFOCOM, ICPP, IEEE/ACM IWQoS, ACM Multimedia, IEEE ICC, IEEE GLOBECOM, and IEEE ICME. He is a senior member of the IEEE.

Jiangchuan Liu (S'01–M'03–SM'08–F'17) received the BEng (cum laude) degree in computer science from Tsinghua University, Beijing, China, in 1999, and the PhD degree in computer science from the Hong Kong University of Science and Technology, in 2003. He is currently a university professor with the School of Computing Science, Simon Fraser University, British Columbia, Canada and a visiting professor with the College of Computer Science and Software Engineering, Shenzhen University, Guangdong, China. He is an EMC-endowed visiting chair professor of Tsinghua University, Beijing, China and an adjunct professor of Tsinghua-Berkeley Shenzhen Institute. In the past, he worked as an assistant professor with the Chinese University of Hong Kong and as a research fellow with Microsoft Research Asia. He is a co-recipient of the Inaugural Test of Time Paper Award of IEEE INFOCOM (2015), ACM SIGMM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), and ACM Multimedia Best Paper Award (2012). His research interests include multimedia systems and networks, cloud computing, social networking, online gaming, big data computing, RFID, and Internet of Things. He has served on the editorial boards of the *IEEE/ACM Transactions on Networking*, the *IEEE Transactions on Big Data*, *IEEE Transactions on Multimedia*, *IEEE Communications Surveys and Tutorials*, and *IEEE Internet of Things Journal*. He is a steering committee member of the *IEEE Transactions on Mobile Computing* and Steering Committee chair of IEEE/ACM IWQoS (2015-2017). He is a fellow of the IEEE and an NSERC E.W.R. Steacie memorial fellow.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: SIMON FRASER UNIVERSITY. Downloaded on January 15,2022 at 10:03:13 UTC from IEEE Xplore. Restrictions apply.