# On Energy-Efficient Offloading in Mobile Cloud for Real-Time Video Applications

Lei Zhang, *Student Member, IEEE*, Di Fu, *Student Member, IEEE*, Jiangchuan Liu, *Senior Member, IEEE*, Edith Cheuk-Han Ngai, *Senior Member, IEEE*, and Wenwu Zhu, *Fellow, IEEE*

***Abstract*— Batteries of modern mobile devices remain severely limited in capacity, which makes energy consumption a key concern for mobile applications, particularly for the computation-intensive video applications. Mobile devices can save energy by offloading computation tasks to the cloud, yet the energy gain must exceed the additional communication cost for cloud migration to be beneficial. The situation is further complicated by real-time video applications that have stringent delay and bandwidth constraints. In this paper, we closely examine the performance and energy efficiency of representative mobile cloud applications under dynamic wireless network channels and state-of-the-art mobile platforms. We identify the unique challenges of and opportunities for offloading real-time video applications and develop a generic model for energy-efficient computation offloading accordingly in this context. We propose a scheduling algorithm that makes adaptive offloading decisions in fine granularity in dynamic wireless network conditions and verify its effectiveness through trace-driven simulations. We further present case studies with advanced mobile platforms and practical applications to demonstrate the superiority of our solution and the substantial gain of our approach over baseline approaches.**

***Index Terms*— Energy efficiency, mobile cloud, offloading, video.**

## I. Introduction

MOBILE devices, including smartphones and tablets, have become an essential part of our lives. As one of the most effective and convenient tools for communication and entertainment, they are not bound by time and place. By 2019, there will be 11.5 billion mobile-connected devices, including machine-to-machine modules, which exceeds the world's projected population at that time (7.6 billion). In addition, the global mobile data traffic is expected to increase nearly 10-fold between 2014 and 2019, reaching 24.3 EB per month by 2019, a majority of which will be video related [1]. Despite the fast development of the technologies and the effort toward unifying hand-held and desktop computers (e.g., through Windows 8/10, iOS/MacOS, and Android/ChromeOS), it remains widely agreed that mobile terminals will not completely replace laptop and desktop computers in the near future. Migrating popular PC software to mobile platforms or developing similar new software is still confined to the limited computation capability of the mobile devices, as well as their unique and lightweight operating systems and hardware architectures. Further, mobile devices continue to suffer from a limited battery capacity. As the only power source of most mobile devices, the battery has shown relatively slow technological improvement in the past decade. The average capacity has grown only 5% annually [2]. Thus, even though the hardware and the mobile networks continue to evolve, energy is still a major impediment to providing reliable and sophisticated mobile applications that meet the user demands.

Mobile cloud computing, which combines the strength of clouds and the convenience of mobile devices, appears natural and attracts tremendous attention [3]–[5]. We shift the hardware/software requirements and the necessary computing loads from the mobiles to the cloud proxies. Such *computation offloading* prolongs the battery lifetime and expands their computation capability, network bandwidth, and storage space. This is particularly attractive for many video-based applications (e.g., virtual desktop infrastructure and cloud gaming), which are generally computation intensive. For instance, Gaikai and OnLive, which execute video games in the cloud and deliver video streams to end users, have established multimillion user bases in the past few years [3]. Yet moving computation tasks to remote cloud may incur a large volume of extra data transfer. Taking the video game case as an example, our experiment on OnLive shows that the offloaded execution requires a data transfer rate of 288.89 kB/s for the gaming video. This introduces a dilemma for real-time video applications: on the one hand, offloading the computation to the cloud can significantly mitigate the workload on mobile devices, and thus the energy cost for intensive computations can be saved; on the other hand, such real-time data transmission as video streaming, which needs to be guaranteed for quality of service, is very vulnerable to the change in network condition. Naively offloading the tasks regardless of the network condition may cause intolerable performance degradation (e.g., frequent interruptions in video playback). To this end, a smart offloading scheduling scheme

Manuscript received August 15, 2015; revised January 4, 2016; accepted February 20, 2016. Date of publication March 8, 2016; date of current version January 5, 2017. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant and in part by the NSERC Strategic Project. The work of E. C.-H. Ngai was supported in part by the Vinnova GreenIoT Project and in part by the Swedish Foundation for International Cooperation in Research and Higher Education international collaboration in Sweden. This paper was recommended by Associate Editor Y. Wen.

L. Zhang and D. Fu with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: lza70@cs.sfu.ca; dif@cs.sfu.ca).

J. Liu is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada, and also with South China Agricultural University, Guangzhou, Guangdong, 510642, China. (e-mail: jcliu@cs.sfu.ca).

E. C.-H. Ngai is with the Department of Information Technology, Uppsala University, Uppsala 751 05, Sweden (e-mail: edith.ngai@it.uu.se).

W. Zhu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: wwzhu@tsinghua.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2016.2539690

is needed, in order to: 1) identify whether the offloading can be beneficial considering the reduced computation and the introduced communication overhead and 2) adaptively make offloading decisions based on the current network condition to ensure the quality of service.

In this paper, we investigate energy-efficient mobile offloading for video-based applications. Different from other types of applications, real-time video applications have a stringent delay constraint and dynamic bandwidth requirement. The playback needs to be continuous and the video quality can vary to adapt to the changing network conditions. Through measurements of dynamic wireless network channels in the state-of-the-art mobile platforms, we examine the performance and energy efficiency of migrating representative applications to the cloud. We identify the critical issues in mobile offloading for real-time video applications. We then develop a generic offloading model accordingly in this context and propose a scheduling algorithm that adaptively offloads tasks to accommodate the dynamics of wireless channels in fine granularity. Trace-driven simulation results prove the effectiveness of our solution. We further present two case studies of practical applications with advanced mobile platforms to demonstrate the superiority of our solution and the significant gain of our approach over existing approaches.

## II. Related Work

From the computation perspective, cloud computing offers high availability, high reliability, and virtually infinite resources, which have made offloading promising for mobile platforms. MAUI [6] and CloneCloud [4] partition applications using a framework that combines static program analysis with dynamic program profiling to optimize execution time or energy consumption. Using smartphone VM (virtual machine) image inside the cloud, ThinkAir [7] aims for computation offloading in a commercial cloud scenario with multiple mobile users instead of a single user. It considers not only offloading efficiency and convenience for developers, but also the elasticity and scalability of the cloud for the dynamic demands from users. From the communication perspective, managing energy consumption for networked transactions is a critical issue for mobile devices. Ra *et al.* [8] discuss the trade-off between QoS and delay of data transmission for mobile platforms and present a stable and adaptive link selection algorithm. Catnap [9] exploits the bottlenecks of wireless and wire links and utilizes an application proxy to decouple data units into segments, which are scheduled as bursts during transmission for energy saving. An empirical study of Bartendr [10] demonstrates that a strong signal can reduce energy cost. It then develops energy-aware scheduling algorithms for different workloads, including background synchronization traffic and video stream traffic, based on signal prediction by location and history.

With cloud computing, considerable research efforts have been dedicated to improve the video streaming services. CALMS [5] adaptively leases and adjusts resources in the cloud servers to meet the dynamic demands from users, offering a generic framework for migrating live streaming services. In [11], considering the geographical diversity of cloud resource prices, a Nash bargaining solution is developed for bandwidth provisioning and video placement strategies. An emerging mobile cloud computing paradigm that involves both offloading and video streaming is the mobile remote desktop access (MRDA) [12]. In MRDA, the entire desktop environment is hosted in the remote server while the client is only in charge of receiving and displaying the contents. Another related application is *cloud gaming*, which migrates game execution to the cloud and streams the gaming scenes back to the end users [13]. However, it demands ultralow latency, and the video decoding on the client side may result in excessive use of energy. It has been shown that a naive offloading can incur even higher energy consumption in a state-of-the-art mobile platform [14].

Different from the previous studies on computation offloading that have focused on service partitioning and correspondingly the low-level details such as program profiling and state migration, we target and investigate the energy-aware scheduling for the offloading requests and the required data transmission based on the application demands and the network condition. We then apply the proposed scheduling algorithm to real-time video streaming applications, particularly those involving rich user interactions and tightened time constraints, such as MRDA and cloud gaming, to achieve more efficient mobile device power usage. To the best of our knowledge, this is the first work that jointly considers mobile computation offloading and energy efficiency for computation and transmission in the context of real-time video.

## III. Why Is Offloading Over Wireless Challenging for Real-Time Video?

To understand the challenges of offloading from mobile terminals to the cloud through wireless channels, we have conducted a series of measurement studies on a 32 GB Google Nexus 7 tablet. We are interested in both the service quality and the energy consumption of the tablet. To this end, we used a digital multimeter to record the current transferred between the battery and the tablet, since the supply voltage remains stable for a long period of time. The current is sampled every second and recorded by a software with an accuracy of 10 mA. We unplugged the battery from the device and wired them up with the digital multimeter using serial connection in the circuit. A desktop computer running Linux OS is used to bridge the router with access to the Internet. The tablet is connected to a wireless access point. Fig. 1 shows a snapshot of the test devices that are used in our measurement. To emulate different network properties, we use the `netem` tool on the desktop PC, which provides the network emulation functionality and is widely available in existing Linux systems. Its features include wide area network delays with different delay distribution, packet loss, packet duplication, packet corruption, and packet reordering.

Two real-world mobile applications are selected as test applications in this experiment. The first application is OnLive, a mature and pioneering commercial cloud gaming platform. It is also a representative mobile offloading video application. In OnLive, the gamer's commands are sent from the thin client

Fig. 1. Snapshot of the test devices.

TABLE I
DELAY TOLERANCE IN ONLINE GAMING

| Example Game Type | Perspective | Delay Threshold |
|---|---|---|
| First Person Shooter (FPS) | First-Person | 100 ms |
| Role Playing Game (RPG) | Third-Person | 500 ms |
| Real Time Strategy (RTS) | Omnipresent | 1000 ms |

TABLE II
AVERAGE NUMBER OF FRAMES UPDATED BEFORE ARRIVAL OF RESPONSE

| Added delay | Avg Num of frames updated | Response time |
|---|---|---|
| 0 | 5 frames | 0.33 s |
| 50 ms | 5.7 frames | 0.38 s |
| 100 ms | 6.5 frames | 0.43 s |
| 150 ms | 7.5 frames | 0.50 s |

to the cloud gaming platform over the Internet. Once the commands reach the cloud gaming platform, they are converted into appropriate in-game actions, which are interpreted by the game logic and turned into changes in the game world. The game world changes are then processed by the graphical processing unit (GPU) to form a rendered scene. The rendered scenes are compressed by the video encoder and sent to a video streaming module, which delivers the video stream back to the thin client. Finally, the thin client plays the video frames to the gamer. To test OnLive, we run a cross-platform game through an OnLive mobile client as well as on our test device locally. The game is available with a mobile version in Google Play Store.

The second application is an open source chess game. It is a delay-tolerant and computation-intensive application, which acts as a counterpart of real-time video application. In the chess game, the server is able to configure a network chess engine to analyze chess positions and make decisions on the best chess moves. To test this application, we set the chess game to computer versus computer mode, in which no human input is required. The program communicates with the chess engine and uses it to search for the best moves.

### A. Impact of Delay

We first measure the impact of delay on energy consumption and user experience in the experiment. When the network latency increases, the power consumption does not vary much in the two test applications. Although the energy efficiency of computation offloading remains stable, we argue that the increasing delay can hurt the performance of real-time video applications that are offloaded to the cloud. Longer delay means that more time is needed to get the response when the application interacts with the cloud. For some applications such as email, the delay can be masked so that it does no harm to the user experience. In some chess games, each player has a fixed time interval to think and make the move. If the network delay increases, the network chess engine has less time to search for the best move (less thinking time). It does not hurt the user experience as long as the total delay does not exceed the permitted time interval.

In contrast, for delay-sensitive applications such as real-time video applications, a longer network delay can result in significant performance degradation. Studies on traditional online gaming systems have found that different types of games have different thresholds of maximum tolerable delay [15]. Table I summarizes the maximum delay that an average player can tolerate before the quality of experience (QoE) begins to degrade. To measure the actual impact of network delay in OnLive, we recorded the video on the tablet's screen. The frame rate was approximately 15 frames/s in OnLive. By analyzing the gaming video frame by frame, we were able to measure the response time of each user's input action under different network delays. As shown in Table II, the average response time increases with the network delay, which indicates that OnLive has significantly degraded the user experience.

### B. Impact of Packet Loss

We next examine the influence of packet loss rate, which is another important parameter in network channels. From our measurements, we find that the instant power consumption of the two test applications does not change significantly with the packet loss rate. Nevertheless, it does not mean that the energy efficiency of computation offloading remains unchanged. The packet loss may have great negative effects on the quality of network communication. It may result in higher energy consumption for transferring the same amount of data. In other words, the effective throughput on the wireless link drops when the packet loss rate increases. We captured the number of transferred packets during computation offloading in the two test applications.

The chess game's GUI communicates with the network chess engine through a transmission control protocol (TCP) connection. We classify packets that do not appear in successful data transfer as unnecessary packets, including duplicate ACKs, timeout retransmissions, fast retransmissions, ACKs of unseen segment, and other special packets. In each run of the chess game, we counted the total number of packets and the number of unnecessary packets and calculated the percentage of unnecessary packets. As shown in Fig. 2(a), there are only 0.1% unnecessary packets in a lossless wireless communication. The unnecessary packets take

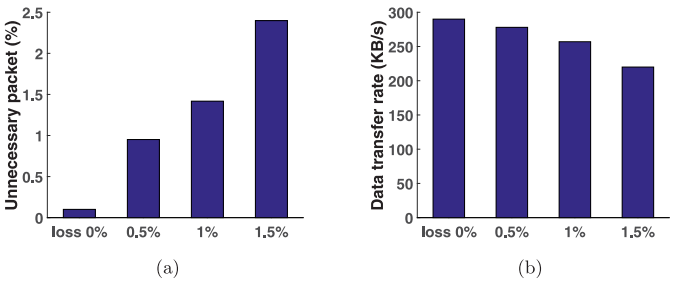



Fig. 2. Impact of packet loss. (a) Percentage of unnecessary packets. (b) Transfer rate of OnLive.

up 2.4% of the total traffic when the packet loss rate increases to 1.5%. Such a traffic overhead caused by packet loss may be tolerable for offloading applications with small data transfer rates [8], such as the chess game. However, in other cases where real-time video applications are operating, packet loss can result in significant degradation in user experience. Let us take OnLive as an example. Different from a traditional client–server gaming, which is limited to transmitting game engine data/commands, OnLive streams the gaming scene videos to the users via UDP. We measured the data transfer rate between the mobile client and the OnLive's cloud sever instance under varying packet loss rates. Fig. 2(b) presents the mobile client's average downloading data rate under different packet loss rates. It shows that the downloading rate drops 24.1% when the wireless connection loses more packets, even though the data rate remains higher than 200 kB/s. It is worth noting that the decreasing data rate implies not only lower energy efficiency but also poorer user experience. For instance, the quality of the gaming video can be noticeably degraded with a low data rate.

## IV. Energy-Efficient Offloading for Real-Time Video: Generic Model and Solution

The measurement mentioned earlier suggests that offloading in the wireless mobile environment has its distinct challenges, particularly for video applications. First, real-time video applications have a stringent requirement in terms of delay. Although our primary goal is to save energy on the mobile devices, we do need to guarantee continuous video playback. Besides rate adaptation, the context of offloading in mobile cloud provides us another option: aborting the transmission and keeping the computation locally. When offloading is no longer applicable or beneficial, a decision can be made to keep task execution and video rendering local. Second, both the uplink and downlink traffic should be taken into consideration when scheduling offloading in mobile cloud. There are basically three steps for offloading a task: 1) sending the data from the client; 2) processing the task and encoding the video in the cloud; and 3) receiving and decoding the video at the client side. The schedule of downlink transmission is therefore cascaded with that of uplink transmission. Given the playback deadlines, not only the sending and receiving intervals should be specified, but also the time between them must be long enough for task execution in the cloud.

We now consider the general mobile offloading scenario for real-time video applications and develop a generic scheduling model that adaptively selects the offloaded tasks and energy-efficiently schedules the offloading transmissions between mobiles and the cloud.

### A. Generic Offloading Problem

Assume that the application has a set of $n$ tasks $J = \{j_1, j_2, \ldots, j_n\}$, and $m$ time slots $\{t_1, t_2, \ldots, t_m\}$ in total to execute them. Each task stands for a possible user input and the task execution and rendered video corresponding to this input, which can be executed either locally or in the cloud. Taking cloud gaming as an example, in a continuous gaming scene, the rendering of each object/in-game character can be taken as a separate task, since each in-game character can have an independent game logic of responding to the gamer's input. We denote a task with a tuple of five elements $j_i = (c_i, t_i^a, t_i^d, d_i^s, d_i^r)$, where $c_i$ is the computation load (e.g., the computation for task execution and video rendering), $t_i^a$ is the task's start time (e.g., when the user input occurs), $t_i^d$ is the deadline for the task to be completed and sent back (e.g., the deadline for the continuous playback), $d_i^s$ is the amount of data to be sent (e.g., the user input and the current state), and $d_i^r$ is the amount of data to be received (e.g., the execution results and the video fragments). These parameters can be predicted by some forecast algorithms [16].

Given the knowledge of a wireless channel state for a given time span, the function $R(t_i)$ maps the channel state to the effective throughput at each slot $t_i$, which denotes the useful bandwidth considering network delay and packet loss during data transmission. We will address how to model the wireless channel states and acquire $R(t_i)$ later. We consider that the energy cost function can be obtained based on network measurements with real hardware. We are then able to calculate the energy consumption $E_c(c)$ for the computation of $c$ and the energy consumption $E_t(d)$ for the wireless transmission of $d$. It is worth noting that, in the current formulation, rather than specifying a certain wireless access technology, we employ a generic energy model, which can be further extended based on various features of different wireless access technologies.

Different network interfaces have different characteristics in power consumption. Moreover, the rapid development of multicore multithread processors has significantly changed the characteristics of energy consumption, not to mention the countless hardware models and various implementations in commercial mobile products. As such, we do not rely on a specific energy model here, but consider a generic energy cost function, which allows our formulation and solutions to be easily integrated with different hardware platforms and wireless interfaces [17], [18]. We will examine its impact in our case studies later.

Our goal is to find an optimal schedule for the client to execute (locally) or offload (to the cloud) each task and minimize the energy consumption with delay constraints. Let $s_i = (d_i, [t1_i, t2_i], [t3_i, t4_i])$ be the schedule for $j_i$, where $d_i$ is the offloading decision. $d_i = 0$ indicates that the client decides to execute $j_i$ and render the video locally, while $d_i = 1$ indicates that the client offloads $j_i$ to the cloud and receives the encoded video. $[t1_i, t2_i]$ and $[t3_i, t4_i]$ are the time intervals for sending and receiving data, respectively. Note that we

only consider the sending and receiving intervals for the tasks that are chosen to be offloaded. If a task is decided to be run locally, we do not consider its schedule. Let $F_c$ be the processing capability of the cloud server that executes the offloaded tasks. Note that scheduling is performed at the client side. The problem can be formulated as finding an optimal schedule $S = \{s_1, s_2, \ldots, s_n\}$ that minimizes the total energy consumption

$$E_{\text{total}} = \sum_{i=1}^{n} \left\{ (1 - d_i) E_c(c_i) + d_i E_t\left(d_i^s + d_i^r\right) \right\} \tag{1}$$

and the following constraints should be satisfied.

1) Causality constraint:

$$\forall i \in [1, n], \quad t1_i \geq t_i^a. \tag{2}$$

2) Playback continuity constraint:

$$\forall i \in [1, n], \quad t4_i \leq t_i^d. \tag{3}$$

3) Streaming rate constraint:

$$\forall i \in [1, n], \quad \sum_{t_k = t1_i}^{t2_i} R(t_k) \geq d_i^s \quad \text{and} \quad \sum_{t_k = t3_i}^{t4_i} R(t_k) \geq d_i^r. \tag{4}$$

4) Time gap constraint:

$$\forall i \in [1, n], \quad t3_i - t2_i \geq c_i / F_c. \tag{5}$$

The causality constraint (2) implies that a task cannot be scheduled before its start time (when the corresponding user input occurs). The playback continuity (3) follows that computation offloading should not cause performance degradation in terms of user experience even though our goal is to save energy. In other words, each video fragment must be received before the playback deadline. Meanwhile, the start times and the deadlines can be set to meet the playback sequence of video fragments. For real-time video applications, the required data rate is affected by both the encoding settings and the video contents. The streaming rate constraint (4) ensures that the necessary amounts of data are transferred during the scheduled intervals given the expected data rate and available bandwidth. To execute an offloaded task in the cloud, some processing time is required. Taking cloud gaming as an example, when the user actions are input to the cloud server, it takes time to render the corresponding gaming scenes according to the gaming logic and encode the gaming video. The time gap constraint (5) guarantees that the cloud server has enough time to process the offloaded task and generate the encoded video.

Besides the previous constraints, there are also some implicit constraints. First, a task can only be considered for offloading if it leads to certain energy savings according to the computation and transmission cost functions

$$\forall i \in [1, n], \quad \text{if } d_i = 1, \ E_t\left(d_i^s + d_i^r\right) < E_c(c_i).$$

Second, the schedules of different tasks cannot conflict with each other, which guarantees the feasibility of our sequential scheduling

$$\begin{aligned} \forall i \neq j, \quad [t1_i, t2_i] \cap [t1_j, t2_j] &= \emptyset \\ [t1_i, t2_i] \cap [t3_j, t4_j] &= \emptyset \\ [t3_i, t4_i] \cap [t1_j, t2_j] &= \emptyset \\ \text{and } [t3_i, t4_i] \cap [t3_j, t4_j] &= \emptyset. \end{aligned}$$

This problem is challenging as each task has two intervals, the sending interval and the receiving interval, to schedule with the restriction of the start time, the deadline, and the processing time in between. The criterion for making offloading decision for a specific task is whether there will be energy saving if the task is completed before the deadline. This can vary significantly for different schedules under a dynamic network environment. Moreover, it is possible for a single task to appear on more than one schedule, which can save the same amount of energy. Considering the scheduling of multiple tasks, this problem becomes even harder, as each scheduled task directly affects the available time slots for the following tasks. The decision version of the formulated problem can be proved to be NP complete, to which the subset sum problem [19] is reducible.

**Algorithm 1** Greedy Offloading Scheduling

```
Set the time array T as all available;
Sort the current task queue J by descendant order of c/(d^s + d^r)
while true do
    Select the first task j_1 in the sorted queue;
    Search the interval [t_1^a, t_1^d] in T for the valid schedules that
    satisfy all the constraints (1)-(4)
    if there is no valid schedule then
        d_1=0
    else
        d_1=1;
        Select the schedule with the highest energy saving E_c(c_1) -
        E_t(d_1^s + d_1^r) (if there are multiple schedules, select the latest
        one);
        Update T accordingly
    end if
    J ← J − {j_1}
    if there is any new task arrived then
        Add the new task to J;
        Sort J by descendant order of c/(d^s + d^r)
    end if
end while
```

### B. Greedy Offloading Schedule

To practically solve the problem, we propose a greedy heuristic as shown in Algorithm 1. It sorts the tasks according to the computation-to-data ratio and schedules the more computation-intensive tasks with higher priorities. As discussed earlier, our goal is to save as much energy as possible by offloading the computation to cloud. Given the network throughput, the amount of data that can be transferred in each time slot and the resulting transmission energy cost are fixed. A higher computation-to-data ratio implies that more energy could be saved if the corresponding task is offloaded. By searching the valid schedule of each task in the order of computation-to-data ratios, the proposed greedy heuristic significantly improves the search efficiency. We first sort the

task queue according to computation-to-data ratios (line 2) and further schedule each task in the sorted order. For each task, the algorithm checks every possible offloading schedule that satisfies all the constraints and computes the corresponding energy saving (line 5). If none of the examined offloading schedules can save energy, the current task is scheduled to be executed locally (lines 6 and 7); otherwise, its offloading would be scheduled to achieve the maximum energy saving (line 10). When multiple schedules are found with the same amount of highest energy saving, our heuristic algorithm schedules the transmission intervals as late as possible in order to have the minimum impact on the scheduling of later tasks. Let $M$ be the number of available time slots in the scheduling interval (e.g., $[t_i^a, t_i^d]$ for task $i$). For each of the $n$ tasks, the heuristic checks $O(M^2)$ schedules ($O(M^2)$ combinations of sending and receiving intervals given the scheduling interval), each of which needs constant time to evaluate. Our greedy algorithm can quickly find the near-optimal solution with the running time of $O(nM^2)$. For real-time video applications with tight delay constraints, the size of scheduling interval is very limited, and thus our algorithm causes negligible computation overhead on modern devices with considerable computation capabilities.

To implement our proposed heuristic, there are still several practical issues to be addressed. On the cloud side, a comprehensive mobile offloading system needs to consider various aspects: existing studies have done extensive work on user privacy [20], status monitoring [21], service scalability [22], and ubiquitous availability [23]. On the mobile side, although our algorithm introduces only very limited computation overhead, it requires information on the wireless channel, e.g., bandwidth, which inevitably causes other overhead related to network measurement and estimation. To this end, we propose a lightweight adaptive bandwidth probing and wireless channel monitoring approach in the next section.

### C. Wireless Channel Modeling and Bandwidth Probing

In online scheduling, it is necessary to monitor and adapt to the highly dynamic wireless channel conditions. There have been significant studies on modeling wireless channels [24], particularly the ones based on Markov chain [25]. Consider a basic and widely used two-state Gilbert–Elliott model [24], which has two channel states for good and bad conditions, respectively, capturing the bursty nature of a wireless channel. Let $P_G$ be the probability that the channel will stay in the good state in the next time slot given that the current state is good (similarly, we can define $P_B$). Accordingly, the state transition probability from a good state to a bad state in the next time slot is $1 - P_G$. The expectation time that the wireless channel remains in the good and bad states can then be calculated as $T_G = (1/1 - P_G)$ and $T_B = (1/1 - P_B)$.

It is known that wireless bandwidth depends only partially on the signal strength. Hence, using received signal strength indicator directly to identify the states and to infer the available bandwidth is not necessarily effective [8]. Instead, we use an adaptive bandwidth probing algorithm to identify the state of the wireless channel. Our algorithm, inspired by [26], estimates the bandwidth based on real measurement of the transmitted packets. In particular, we use the packets of the offloaded task as the probe to reduce the energy cost of transmitting irrelevant data.

**Algorithm 2** Bandwidth Probing and Wireless Channel State Monitoring

1: **Initialize**: set the probing interval $T_{in} = 1$, last measurement $R_L = 0$, current measurement $R_C = 0$, counter $i = 1$, average bandwidth in good state channel $R_g = 0$, average bandwidth in bad state channel $R_b = 0$, channel state $Status = good$, threshold $R' = 3$ MB, parameter $\alpha = \beta = 0.3$
2: **while** true **do**
3: **if** $i == T_{in}$ **then**
4: $i = 0$;
5: $R_L = R_C$;
6: $R_C \leftarrow$ measured bandwidth
7: **if** $|R_L - R_C| < \beta R_L$ **then**
8: $T_{in}++$
9: **else**
10: $T_{in} = \lceil T_{in}/2 \rceil$
11: **end if**
12: **if** $Status$==good **then**
13: **if** $R_L < R'$ & $R_C < R'$ **then**
14: $Status$=bad;
15: $R_b = (1-\alpha)R_b + \alpha R_C$
16: **else**
17: $R_g = (1-\alpha)R_g + \alpha R_C$
18: **end if**
19: **else**
20: **if** $R_L > R'$ & $R_C > R'$ **then**
21: $Status$=good;
22: $R_g = (1-\alpha)R_g + \alpha R_C$
23: **else**
24: $R_b = (1-\alpha)R_b + \alpha R_C$
25: **end if**
26: **end if**
27: **end if**
28: $i++$
29: **end while**

As shown in Algorithm 2, once the bandwidth is estimated, we use a threshold value $R'$ to decide which state the wireless channel is in and update the average bandwidth in that state. If two successive measurements exceed the threshold, we consider that the state of the wireless channel has changed. By recording how long the wireless channel stays in each state, we are able to estimate the state transition probability. We can then use the average bandwidth in the current channel state as the prediction for the future (see Algorithm 1). Note that wireless bandwidth largely depends on the location of the user and user mobility is usually confined to a certain period. Our algorithm adjusts the probing interval adaptively, similar to the congestion control protocol in TCP. The parameters, such as $R'$, $\alpha$, and $\beta$, can be tuned under different practical scenarios. It is worth noting that the algorithm can also be easily extended to work with higher order channel models with more states [25] to achieve better accuracy.

## V. Trace-Driven Evaluation

In this section, we make a comparison between the optimal offloading scheduling and our proposed approach with two bandwidth probing methods using real-world traces and synthetic data.

TABLE III
POWER MODEL

| Model | $\beta_u \text{util} + \beta_{cpu}\text{CPU}_{on} + \beta_l \text{WiFi}_l + \beta_h * \text{WiFi}_h + \beta_{others} * \text{Oth}_{on}$ | | |
|---|---|---|---|
| Category | Variable | Range | Power coefficient (mW) |
| CPU | util | 1-100 | 19.9 |
| | $\text{CPU}_{on}$ | 0,1 | 139 |
| WiFi | $\text{WiFi}_l$ | 0,1 | 40 |
| | $\text{WiFi}_h$ | 0,1 | 605 |
| Others | $\text{Oth}_{on}$ | 0,1 | 740 |

TABLE IV
ENERGY EFFICIENCY OF OPTIMAL OFFLOADING SCHEDULING AND PROPOSED APPROACH WITH DIFFERENT BANDWIDTH PROBING METHODS

| | Optimal | Adaptive Probing | Regular Probing |
|---|---|---|---|
| Normalized Cost | 0.8270 | 0.8406 | 0.8599 |

The real-world bandwidth traces are collected when the test device is moved toward and away from the wireless AP periodically, experiencing a varying wireless network connection. In our later evaluation and case studies, we used IEEE 802.11n as the wireless access technology, whose power model is adopted from [18] with proper adjustments according to our measurements on the Nexus 7 tablet. As we are more concerned about the power consumption, which is directly related to computation offloading, our simplified model mainly considers the power consumption of the CPU and the Wi-Fi communication, as shown in Table III. Based on previous studies [27], the probability distribution of applications' demand can be described by Gamma distribution in certain cases. In our simulations, we set the task arrival rate to be approximately 2 tasks per second with an average delay tolerance of 0.5 s. We generate a small-scale synthetic dataset based on the collected execution traces of the test applications in Section III and using the settings described previously.

We evaluate the performance of our greedy scheduling approach and the effectiveness of our adaptive bandwidth probing method using a small-scale dataset, where the optimal offloading schedules can be obtained through exhaustive searching. We also implement a baseline reference scheme: all tasks are executed locally without offloading (LOC), which schedules task execution in a first-in first-out manner. As our algorithm is designed with the minimal performance guarantee (constrained by playback deadlines), we focus on evaluating its energy efficiency. The energy consumption of our proposed approach and the optimal offloading scheduling is normalized by the baseline result of LOC. We further compare our adaptive bandwidth probing method with the regular periodic probing method (sampling bandwidth: every 50 time slots). Table IV presents the normalized energy consumption of optimal offloading scheduling and our proposed approach with two bandwidth probing methods. It proves that, compared with the baseline LOC scheme, our proposed approach can save around 15% of the energy consumption even under a varying wireless channel. The result also shows that our greedy scheduling approach can achieve near-optimal performance, which only consumes 2% more energy than optimal offloading scheduling, and our adaptive bandwidth probing method outperforms the regular periodic bandwidth probing method.

TABLE V
COMPARISON OF VIDEO PLATFORMS AND OUR GENERIC MODEL (EA-MRDA)

| Feature | VNC [12] | RDP [28] | GAW [13] | EA-MRDA |
|---|---|---|---|---|
| Primitive-support | Yes | Yes | No | Yes |
| Game Streaming | No | No | Yes | Yes |
| Energy-aware | No | No | No | Yes |

## VI. REAL-WORLD CASE STUDIES

So far we have considered the generic model that tries to cover different application demands. Based on this model, we propose a practical framework called energy-aware MRDA (EA-MRDA), the basic idea of which is to adaptively switch among the more energy-efficient offloading schemes while meeting quality and deadline constraints. A comparison between the proposed framework and other representative remote desktop access platforms is presented in Table V. As the energy consumption characteristics are strongly hardware and platform-dependent, to understand the performance of our solution in practice and corroborate its superiority, we provide two case studies on advanced real-world mobile platforms for cloud gaming and MRDA, respectively. We further evaluate our proposed solution based on the collected real-world experiment traces.

### A. Case 1: Video Benchmark on Cloud Gaming

In the first case study, we focus on cloud gaming and evaluate our model using the real-world execution traces of the selected video benchmarks.

*1) Data Trace Collection:* We collected the data traces on a 7-inch EVGA Tegra NOTE 7 tablet, which is one of the most powerful and highly optimized tablets for video applications. To measure the system's overall power consumption, we measure the dc intensity in the tablet's circuit and compute the instantaneous power as the product of current and voltage (almost constantly at 3.7 V).

We used a digital clamp meter (Mastech MS2115B) to measure the dc amperage (precision of $\pm 2.5\%$) and chose an advanced 3D benchmark for mainstream tablets, namely, `3DMark Ice Storm` Benchmark. We installed and utilized the Rhizome cloud gaming platform [14] to deliver and display the gaming scenes. For the local rendering tests, we ran the benchmark and began to record readings of the clamp meter using the data logger PCLink with a sampling rate of two times per second. Likewise, for the remote rendering tests, we selected the network video stream and started recording the measurements. For all the experiments, the screen brightness, sound volume, and other settings remained unchanged.

*2) Energy Improvement:* With the collected execution traces (e.g., CPU load, network traffic, and instant current), we compare the power consumption of our approach with those of local rendering and mobile cloud offloading.

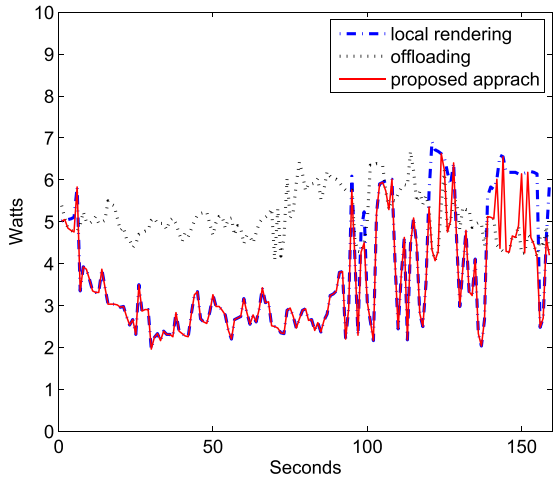



Fig. 3. Comparison of power consumption.

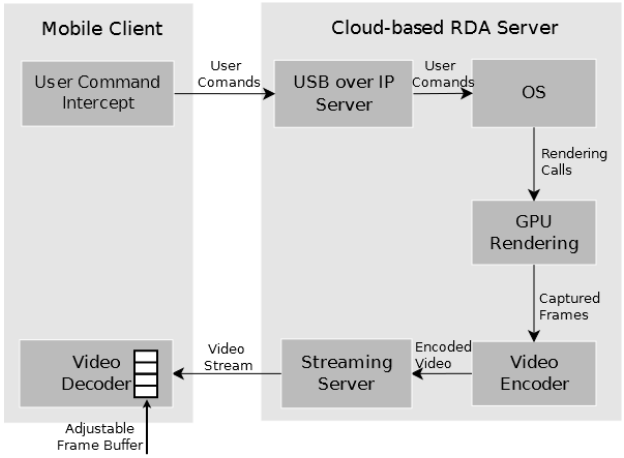



Fig. 4. Workflow of video-based MRDA.

We let the available bandwidth follow a normal distribution with the mean equivalent to the video encoding bit rate and the standard deviation of 100 kB. Fig. 3 shows that our solution can outperform any of the baselines by smartly making offloading decisions. Normally, ours selects the more energy-saving approach between local rendering and offloading to cloud. Yet there are still some cases where we choose the more power-hungry local rendering, when the network condition is too poor to maintain a continuous video playback. It is worth mentioning that, as shown in Fig. 3, there are quite some time periods in which local rendering consumes less power than offloading. A possible explanation is that video decoding itself at these periods is a highly computation-intensive task, which thus can be more energy consuming than directly rendering the corresponding video scenes. As the power consumption of offloading is affected by many factors, such as encoding codec, network condition, and even the objects in video, it reaffirms that the energy model is better kept generic in our previous modeling.

### *B. Case 2: Mobile Remote Desktop Access*

In this case study, we examine our offloading algorithm for MRDA, a computing paradigm that allows mobile device users to enjoy full desktop experience as with traditional PCs. In MRDA, user's inputs are casted to a remote server hosting the actual desktop environment, which in turn handles the resulting state changes and streams back display updates to the client [12]. State-of-the-art MRDA implementations can be categorized into two types: *video based* and *primitive based*. The video-based approach is illustrated in Fig. 4, in which screen updates are rendered at the server side and sent back to the client in the form of a compressed live video stream. In the primitive-based approach, updates are expressed as drawing primitives sent back to the client, where those visual objects are rendered locally and composited with other updates. There have been pioneer studies on energy efficiency for MRDA implementations [29], [30]. Different from the existing work, our offloading model offers more design space. For video-based MRDA, it facilitates adaptive switching between different decoding complexities. For instance, in `x264` [31], the coding tools are encapsulated in the form of option *presets* and *profiles*, where each preset contains predefined encoding settings to achieve different speed-to-compression ratios, and a profile specifies the techniques used for encoding. Depending on the profile used, the client side can spend different amounts of computing resources to decode, leading to distinct energy cost. Our model can then be tuned to choose the encoding profile in an energy-efficient manner. Beyond this, we can even explore the best of both approaches, i.e., adaptive switching between primitive- and video-based MRDA.

It is worth noting that offloading scheduling is not only hardware- and platform-dependent, but also highly application-dependent. Different applications may have different computation-to-data ratios, and thus some of them are more suitable for offloading execution. To comprehensively study the two different decisions (offloading and switching decisions) concurrently, the target application must have a wide range of computation-to-data ratios. However, in practice, we can hardly find such an application. Therefore, to demonstrate the applicability of our algorithm and make the selection choice more clear, we next select two different target applications that are suitable for the two studying cases, respectively, and investigate adaptive switching and offloading separately.

*1) Adaptive Switching Between Profiles:* We first focus on the video-based MRDA with adaptive switching between profiles. We employ two h.264 encoding profiles with different compression complexities, namely, the *baseline* and the *high* profiles, under different network conditions. The baseline profile is compatible with most low-end mobile devices, though it can lead to low compression ratio and excessive bandwidth utilization. On the contrary, the high profile enables the encoder to achieve a given quality level with reduced bit rate and bandwidth requirement at the cost of greater encoding and decoding complexity. Besides, we leverage the `x264 presets` to prevent the encoder from generating intolerable latency, where each preset specifies the parameters used by various encoding techniques, such as the number of *B* frames used per group of pictures. Particularly, we use the `veryfast`

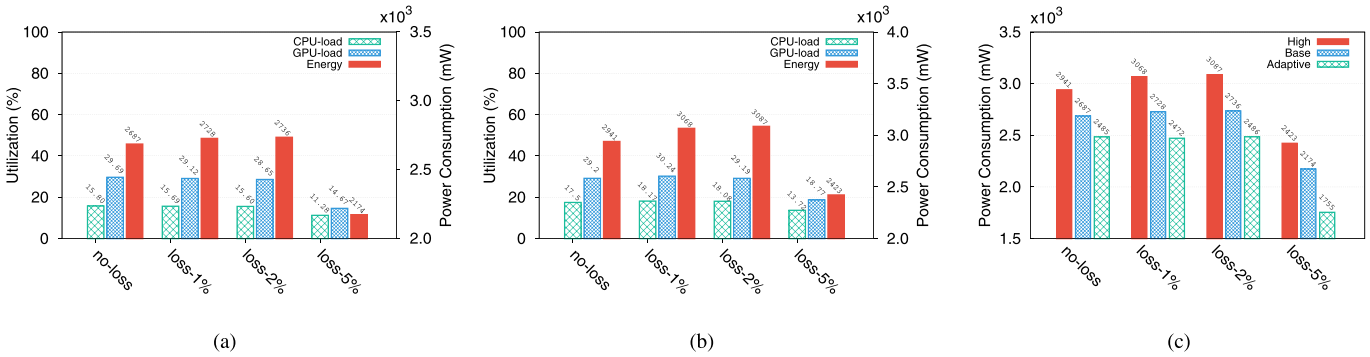



Fig. 5. Video-based MRDA with baseline and high profiles and adaptive profile switching. (a) Baseline profile. (b) High profile. (c) Energy comparison.

`preset` in the high profile experiment, resulting in an average increased streaming latency of ≈30 ms compared with the baseline profile counterpart. Finally, we enforce constant bit rate encoding in base profile and high profile experiments with the maximum bit rate at 3 and 1.8 Mb/s, respectively, achieving close video quality on our samples in terms of peak signal-to-noise ratio (within ±2 db).

We consider that there are $n$ different video encoding/compression settings. Let the decision variable be $d_i \in [0, n]$ ($d = 0$ indicates local rendering), and the corresponding computation and transmission energy cost functions be $E^{d_i}$ and $E_t^{d_i}$, respectively. The previously mentioned implicit energy saving constraint in the generic model can be instantiated as

$$\forall i \in [1, n], \quad d_i = \text{argmin}\left(E_t^{d_i}\left(d_i^s + d_i^r\right) + E_c^{d_i}(c_i)\right). \quad (6)$$

The remote server is configured to run on an Amazon EC2 GPU instance (G2). For the control module, user commands are intercepted by the mobile client and forwarded directly to the remote OS through USB over IP client module and its server counterpart, triggering GPU rendering calls on the updated display (see Fig. 4). Our implementation copies the rendered frames from the GPU display buffer to the main memory by leveraging hardware utilities and completing the screen capture efficiently. Next, the captured frames are encoded by the video encoder and delivered to the streaming server, which supplies the live streams to the client device using the *real-time messaging protocol*.

Our profiling tool is the latest `Trepn 6.1` profiler (published by Qualcomm on August 3, 2015) with the ability to read on-chip sensors and report accurate power consumption and CPU and GPU loads, with a sampling interval of 100 ms, which provides a higher accuracy in power measurements than our previous hardware profiling approach. However, it can only run on the devices that have Qualcomm processors, such as Nexus 6, the test device in this experiment. As both of the previously used devices, Nexus 7 and EVGA Tegra NOTE 7, are equipped with NVIDIA processors, they cannot be profiled by `Trepn 6.1`.

Fig. 5 plots the results when running the `Unigine Valley 3D` GPU Benchmark, a pure video application on the server with customized high quality and full-screen settings, which contains intensive computations and suits the purpose of studying adaptive switching between multiple profiles. We vary the packet loss rate in different tests. In particular, we show the execution traces (e.g., CPU load, GPU load, and power consumption) of the fixed high/baseline profile approach in Fig. 5(a) and (b), respectively, to examine the influences of the varying network condition. We further compare our proposed adaptive switching approach with the fixed profile approach and present the performance gain in Fig. 5(c). We observe that video compression with high profile consumes more power than that consumed with baseline profile, which implies that the increasing decoding complexity causes a significant power rise. Another interesting observation is shown in Fig. 5(a) and (b). When the packet loss rate increases from 0 to 2%, the average power consumption of the two profile settings goes up to 1.82% and 4.97%, respectively. When the packet loss rate reaches 5%, the average power consumption for the two profile settings, however, suddenly drops 19.1% and 17.6%. As the packet loss rate increases from 0 to 2%, as shown in Fig. 5(a) and (b), the CPU and GPU utilization remain almost stable (with only slight drop in some cases). On the other hand, the average power consumption

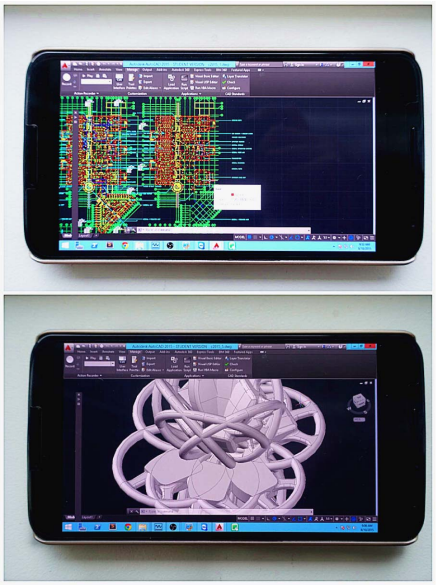


Fig. 6. Remote accessing CAD 2D and 3D tasks in a video-based approach.

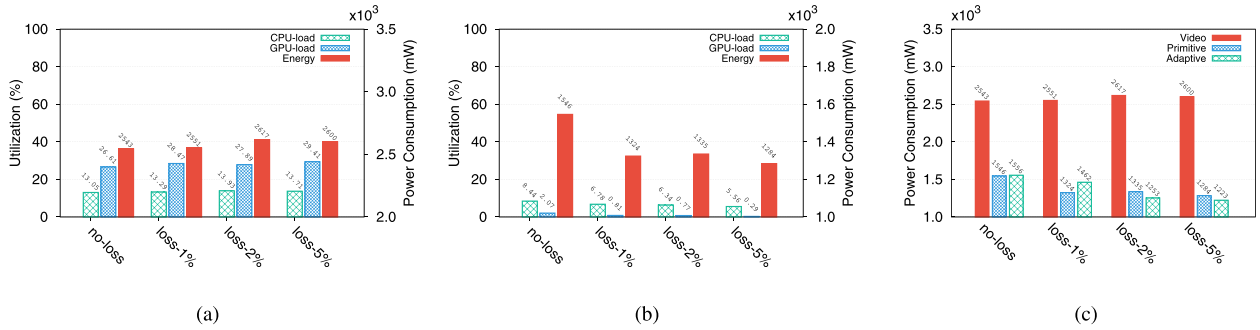



Fig. 7. Video-based versus primitive-based approaches, and adaptive switching between them. (a) Video-based approach. (b) Primitive-based (RDP) approach. (c) Energy comparison.

increases, which implies that more unnecessary packets are transferred (as described in Section III) and consume more energy.

Given each video frame is carried by multiple network packets, when the packet loss rate reaches a threshold (between 2% and 5% in our case), the amount of time taken to receive a complete frame would easily exceed the playback deadline. Those incomplete frames will be dropped by the client without being decoded, resulting in reduced decoding workload. Unfortunately, these frame drops can cause frequent screen freeze, which is undesirable. Fig. 5(c) shows that our adaptive approach outperforms the two fixed compression settings in all the network conditions. It saves 7.52% of energy in the no loss case and 19.3% of energy in the 5% loss case compared with the baseline profile. Our approach is able to select the compression setting with lower transmission cost. In high loss rate cases, our approach avoids unnecessary transmission cost and lowers the computation cost by properly adjusting the compression setting.

*2) Switching Between Video- and Primitive-Based MRDA:* We now examine a hybrid solution that switches between video- and primitive-based MRDA adaptively. We use cloud-based computer-aided design (CAD) as the test application, which includes both desktop objects (2D) and 3D object rendering. We install AutoCAD 2015 on the server and configure the latest version of `Cadalyst` benchmark to run CAD tasks with both 2D and 3D graphics, as shown in Fig. 6. Since our focus is on the decision between video- and primitive-based approaches, we use the high profile at 1.5 Mb/s as the default and set the maximum allowed bandwidth as 2 MB/s for the video-based approach. The primitive-based approach is implemented by the remote desktop protocol (RDP) in Microsoft Windows.

In Fig. 7, we depict the results of the experiments with the hybrid approach as well as the video-based and the primitive-based approaches. From Fig. 7(a) and (b), we can see that the primitive-based approach normally consumes less power than the video-based approach. Although requiring a certain amount of transmission, the primitive-based approach locally renders the video and thus introduces no additional computation cost on video decoding. It is worth noting that the bandwidth required by the RDP in the primitive-based approach is directly affected by the video content, which in turn influences its power consumption. For dynamic contents such as the 3D part of the CAD test, RDP needs to send a large amount of data to instruct local rendering. On the contrary, for static contents such as the 2D part of the CAD test, RDP sends only a small amount of data to compensate for the difference between the successive images and the previously rendered image. Fig. 7(b) also shows that the average power consumption of the RDP approach decreases by 16.9% when the packet loss rate grows up to 5%. Similar to the OnLive test in Section III, the transmission rate of RDP decreases as the packet loss rate increases, which lowers the transmission cost of RDP. Note that our hybrid approach does not always perform the best in terms of the average energy consumption, as shown in Fig. 7(c). In certain low loss rate cases, the average power consumption of our approach is slightly higher than that of RDP. The reason is that RDP sometimes needs higher bandwidth than the maximum bandwidth (set as 2 MB/s) allowed by us in the hybrid approach. Our approach instead chooses to compress the video, which incurs higher cost but controls the bandwidth within the limit.

## VII. Conclusion

In this paper, we investigated performance and energy efficiency of migrating representative video applications to the cloud under dynamic wireless network channels on state-of-the-art mobile platforms. Based on the identified challenges and opportunities for offloading real-time video applications, we formulated a generic energy-efficient offloading scheduling problem and proposed an adaptive scheduling algorithm that makes fine-grained offloading decisions according to the dynamic wireless network conditions. We further evaluated the effectiveness of our solution through trace-driven simulations and extensive experiments. Finally, we presented two case studies on video cloud gaming and MRDA to evaluate the performance of our solution in real-world video applications.

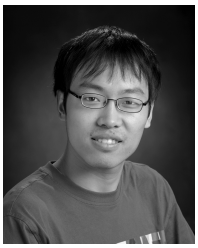
## References

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2014-2019 white paper," Cisco, San Jose, CA, USA, Tech. Rep., 2015.

[2] S. Robinson, "Cellphone energy gap: Desperately seeking solutions," Strategy Analytics, Boston, MA, USA, Tech. Rep., 2009.

[3] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: Architecture and performance," *IEEE Netw.*, vol. 27, no. 4, pp. 16–21, Jul./Aug. 2013.

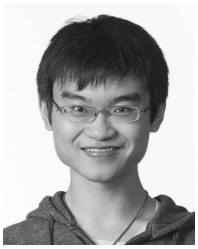
[4] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. 6th ACM EuroSys*, 2011, pp. 301–314.
[5] F. Wang, J. Liu, and M. Chen, "CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 199–207.
[6] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th ACM MobiSys*, 2010, pp. 49–62.
[7] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 945–953.
[8] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proc. 8th ACM MobiSys*, 2010, pp. 255–270.
[9] F. R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices," in *Proc. 8th ACM MobiSys*, 2010, pp. 107–122.
[10] A. Schulman *et al.*, "Bartendr: A practical approach to energy-aware cellular data scheduling," in *Proc. 16th ACM MobiCom*, 2010, pp. 85–96.
[11] J. He, D. Wu, Y. Zeng, X. Hei, and Y. Wen, "Toward optimal deployment of cloud-assisted video distribution services," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 10, pp. 1717–1728, Oct. 2013.
[12] C.-L. Tsao, S. Kakumanu, and R. Sivakumar, "SmartVNC: An effective remote computing solution for smartphones," in *Proc. 17th ACM MobiCom*, 2011, pp. 13–24.
[13] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu, "GamingAnywhere: The first open source cloud gaming system," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 10, no. 1, pp. 10:1–10:25, Jan. 2014.
[14] R. Shea, D. Fu, and J. Liu, "Rhizome: Utilizing the public cloud to provide 3D gaming infrastructure," in *Proc. 6th ACM MMSys*, 2015, pp. 97–100.
[15] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proc. 19th ACM Int. Conf. Multimedia*, 2011, pp. 1269–1272.
[16] S. Gurun, C. Krintz, and R. Wolski, "NWSLite: A light-weight prediction utility for mobile devices," in *Proc. 2nd ACM MobiSys*, 2004, pp. 2–11.
[17] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu, "NAPman: Network-assisted power management for WiFi devices," in *Proc. 8th ACM MobiSys*, 2010, pp. 91–106.
[18] L. Zhang *et al.*, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2010, pp. 105–114.
[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009, pp. 951–953.
[20] A. Gember, C. Dragga, and A. Akella, "ECOS: Practical mobile application offloading for enterprises," in *Proc. 2nd USENIX Conf. Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services*, 2012, p. 4.
[21] J. Park, H. Yu, K. Chung, and E. Lee, "Markov chain based monitoring service for fault tolerance in mobile cloud computing," in *Proc. IEEE Workshops Int. Conf. Adv. Inf. Netw. Appl. (WAINA)*, Mar. 2011, pp. 520–525.
[22] M. Dong, H. Li, K. Ota, L. T. Yang, and H. Zhu, "Multicloud-based evacuation services for emergency management," *IEEE Cloud Comput.*, vol. 1, no. 4, pp. 50–59, Nov. 2014.
[23] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," in *Proc. IEEE 16th Int. Enterprise Distrib. Object Comput. Conf. (EDOC)*, Sep. 2012, pp. 123–132.
[24] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell Syst. Tech. J.*, vol. 39, no. 5, pp. 1253–1265, 1960.
[25] H. Yang and M.-S. Alouini, "A hierarchical Markov model for wireless shadowed fading channels," in *Proc. IEEE 55th Veh. Technol. Conf.*, May 2002, pp. 640–644.
[26] F. Liu, P. Shu, and J. C. S. Lui, "AppATP: An energy conserving adaptive mobile-cloud transmission protocol," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3051–3063, Nov. 2015.
[27] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," in *Proc. ACM SIGMETRICS*, 2001, pp. 50–61.
[28] (2015). *Windows Remote Desktop Protocol*, accessed on Jun. 2015. [Online]. Available: https://goo.gl/Y9LHyF
[29] S.-P. Chuah, Z. Chen, and Y.-P. Tan, "Energy minimization for wireless video transmissions with deadline and reliability constraints," *IEEE Trans. Circuits Syst. for Video Technol.*, vol. 23, no. 3, pp. 467–481, Mar. 2013.
[30] C. Li, D. Wu, and H. Xiong, "Delay—Power-rate-distortion model for wireless video communication under delay and energy constraints," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 7, pp. 1170–1183, Jul. 2014.
[31] (2015). *x264 Video Encoding Library*. [Online]. Available: http://www.videolan.org/developers/x264.html



**Lei Zhang** (S'12) received the B.Eng. degree from the Advanced Class of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, China, in 2011, and the M.S. degree from Simon Fraser University, Burnaby, BC, Canada, in 2013, where he is currently pursuing the Ph.D. degree with the School of Computing Science.

His current research interests include mobile cloud computing, social media, wireless networks, and multimedia systems and networks.

Mr. Zhang was a recipient of the C. D. Nelson Memorial Graduate Scholarship in 2013.



**Di (Silvery) Fu** (S'15) received the B.Sc. (First Class with Distinction) degree in computer science from Simon Fraser University, Burnaby, Canada and B.Eng. degree in computer science from Zhejiang University, Hangzhou, China, both in 2016. He is currently pursuing the M.Sc. degree with the School of Computing Science at Simon Fraser University.

His research interests include cloud computing topics surrounding virtualization, networking, and online gaming.

**Jiangchuan Liu** (S'01–M'03–SM'08) received the B.Eng. (*cum laude*) degree in computer science from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, in 2003.

He was an Assistant Professor with The Chinese University of Hong Kong, Hong Kong, from 2003 to 2004. He is currently a University Professor with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, and an NSERC E. W. R. Steacie Memorial Fellow. He is an EMC-Endowed Visiting Chair Professor with Tsinghua University from 2013 to 2016. His current research interests include multimedia systems and networks, cloud computing, social networking, online gaming, big data computing, wireless sensor networks, and peer-to-peer networks.

Dr. Liu was a co-recipient of the Inaugural Test of Time Paper Award of the IEEE INFOCOM in 2015, the ACM TOMCCAP Nicolas D. Georganas Best Paper Award in 2013, the ACM Multimedia Best Paper Award in 2012, the IEEE Globecom Best Paper Award in 2011, and the IEEE Communications Society Best Paper Award on Multimedia Communications in 2009. His students received the Best Student Paper Award of the IEEE/ACM International Symposium on Quality of Service (IWQoS) twice in 2008 and 2012. He has served on the Editorial Boards of the IEEE TRANSACTIONS ON BIG DATA, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, the IEEE ACCESS, the IEEE INTERNET OF THINGS JOURNAL, *Computer Communications*, and *Wireless Communications and Mobile Computing* (Wiley). He is the Steering Committee Chair of the IEEE/ACM IWQoS from 2015 to 2017.

**Edith Cheuk-Han Ngai** (S'01–M'08–SM'15) received the Ph.D. degree from The Chinese University of Hong Kong, Hong Kong, in 2007.

She held a post-doctoral position with Imperial College London, London, U.K., from 2007 to 2008. She has conducted research with the University of California at Los Angeles, Los Angeles, CA, USA, Simon Fraser University, Burnaby, BC, Canada, and Tsinghua University, Beijing, China. She is currently an Associate Professor with the Department of Information Technology, Uppsala University, Uppsala, Sweden. She is a Visiting Researcher with Ericsson Research, Stockholm, Sweden, from 2015 to 2016. Her current research interests include Internet-of-Things, mobile cloud computing, network security and privacy, smart city, and e-health applications.

Dr. Ngai was a VINNMER Fellow of VINNOVA, Sweden, in 2009. She is a member of the Association for Computing Machinery (ACM). She has served as a TPC Member in networking conferences, including the IEEE International Conference on Distributed Computing Systems, the IEEE INFOCOM, the IEEE International Conference on Communications, the IEEE Globecom, the IEEE/ACM IWQoS, and the IEEE CloudCom. Her co-authored papers have received best paper runner-up awards in the IEEE IWQoS 2010 and ACM/IEEE International Conference on Information Processing in Sensor Networks 2013. She was a TPC Co-Chair of the Swedish National Computer Networking Workshop in 2012 and QShine in 2014. She was a Program Chair of ACM womENcourage 2015, and the TPC Co-Chair of the IEEE SmartCity 2015 and the IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing 2015. She has served as a Guest Editor on the special issue of the IEEE INTERNET-OF-THINGS JOURNAL, the IEEE TRANSACTIONS OF INDUSTRIAL INFORMATICS, and *Mobile Networks and Applications*.

**Wenwu Zhu** (M'97–SM'01–F'10) received the Ph.D. degree in electrical and computer engineering from the New York University Polytechnic School of Engineering, New York, NY, USA, in 1996.

He was with Bell Laboratories, Murray Hill, NJ, USA, as a member of the Technical Staff from 1996 to 1999. He was the Chief Scientist and Director of Intel Research, Beijing, China, from 2004 to 2008. He was a Senior Researcher and Research Manager with Microsoft Research Asia, Beijing. He is currently a Professor and the Deputy Head of the Computer Science Department with Tsinghua University, Beijing. He has authored over 200 refereed papers in multimedia computing, communications, and networking. He holds over 50 patents. His current research interests include multimedia big data computing, cyber-physical-human big data computing, and multimedia communications and networking.

Dr. Zhu is an International Society for Optics and Photonics Fellow and ACM Distinguished Scientist. He received six best paper awards, including ACM Multimedia in 2012 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 2001. He served/serves on various editorial boards, such as a Guest Editor of the PROCEEDINGS OF THE IEEE, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, and an Associate Editor of the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, and the IEEE TRANSACTIONS ON BIG DATA.