On-demand Video Processing in Wireless Networks

Zongqing Lu[†], Kevin S. Chan[‡], Rahul Urgaonkar[§] and Thomas La Porta[†]
[†]Pennsylvania State University, [‡]Army Research Laboratory, [§]Amazon
[†]{zongqing, tlp}@cse.psu.edu, [‡]kevin.s.chan.civ@mail.mil, [§]urgaonka@amazon.com

***Abstract*—The vast adoption of mobile devices with cameras has greatly assisted in the proliferation of the creation and distribution of videos. For a variety of purposes, valuable information may be extracted from these videos. While the computational capability of mobile devices has greatly improved recently, video processing is still a demanding task for mobile devices. Given a network consisting of mobile devices and video-clouds, mobile devices may be able to upload videos to video-clouds, which are more computationally capable for these processing tasks. However, due to networking constraints, when a video processing task is initiated through a query, most videos will not likely have been uploaded to the video-clouds, especially when the query is about a recent event. We investigate the problem of minimal query response time for processing videos stored across a network; however, this problem is a strongly NP-hard problem. To deal with this, we first propose a greedy algorithm with bounded performance. To further deal with the dynamics of the transmission rate between mobile devices and video-clouds, we propose an adaptive algorithm. To evaluate these algorithms, we built an on-demand video processing system. Based on the measurements of the system, we perform simulations to extensively evaluate the proposed algorithms. We also perform experiments on a small testbed to examine the realized system performance. Results show the performance of the greedy algorithm is close to the optimal and much better than other approaches, and the adaptive algorithm performs better with more dynamic transmission rates.**

I. Introduction

The proliferation of handheld mobile devices and wireless networks has facilitated the generation and rapid dissemination of vast numbers of videos. Videos taken for various purposes may contain valuable information about past events that can be exploited for on-demand information retrieval. For example, a distributed video processing problem may involve a query of a set of mobile devices to find a specific vehicle in a region of a city. Various stored videos within mobile devices, not necessarily for the intention of capturing the object of interest, may provide valuable information for such queries. However, the processing requirements for such applications approach the computational limits of the mobile devices. Although the computational capacity of mobile devices has greatly improved in the past few years, processing (multiple) videos is still overwhelming for mobile devices.

In this paper, we consider wireless networks consisting of mobile devices and video-clouds. Instead of storing and processing videos locally, mobile devices can choose to upload videos to more capable devices (*e.g.*, computers with a much powerful GPU), which can significantly accelerate video processing. We call these devices video-clouds. However, due to the availability gap (the time between when the video is taken and when it is uploaded) [1] and when a query is issued, video-clouds will not likely have the pertinent video pre-stored, especially when the query is about recent events. Therefore, to reduce the delay of the on-demand information retrieval from videos related to a query, the related videos can be processed either locally on the mobile devices or transmitted and processed on the video-clouds.

Based on this use of wireless networks for video processing, there are clear scenarios to which this can be applied. Example scenarios are emergency response and video forensics, in which authorities attempt to identify objects or people of interest in videos captured by surveillance systems or other mobile devices. These devices may have been either present or deployed in the time and area of interest. In these situations, video-clouds can be deployed in this area to support the storage and processing of videos to address on-demand information queries about past events. Without video-clouds, this process is significantly delayed, resulting in serious consequences in the event that the query is not addressed satisfactorily.

As an example, an information query may be the following "*did a red truck drive through downtown today?*" Then, all related videos stored on either mobile devices or video-clouds taken in proximity of the "downtown" area need to be processed to detect the presence of a "red truck". The query will reach all devices in the network and finds all of the related videos based on video metadata (*e.g.*, GPS, timestamp). The network needs to determine where to process each video (locally or offloaded to video-clouds), and to which video-cloud to upload each video. This approach should minimize the time required to process all of the related videos, which is referred to as the *query response time*.

Unfortunately, the problem of processing pertinent videos distributed throughout a network with minimal query response time, which is referred to as the *processing scheduling* problem, turns out to be a strongly NP-hard problem. To deal with this, we design a greedy algorithm with bounded performance, which determines whether or not to offload each video, and schedules a transmission sequence to offload videos from a set of mobile devices before processing the videos. To cope with the dynamics of the transmission rate between mobile devices and video-clouds during this process, we further propose an adaptive algorithm, which makes such decisions in runtime. We have built an on-demand video processing system. Based on the measurements of the system, we perform simulations to extensively evaluate the proposed algorithms. We also perform experiments on a small testbed to examine

This work was supported in part by Network Science CTA under grant W911NF-09-2-0053.

the system performance. The major contributions of this paper are summarized as follows.

- We formulate the processing scheduling problem for on-demand video processing to determine the optimal video offloading and transmission sequence in terms of minimizing the query response time.
- We design a greedy algorithm with bounded performance, which exploits average completion time of nodes as a criterion to consecutively determine each video offloading. The performance of the greedy algorithm is close to the optimal and much better than other approaches.
- We propose an adaptive algorithm with very low message overhead to collect information from nodes and then determine video offloading during runtime. The adaptive algorithm performs better when the transmission rate between mobile devices and video-clouds is more dynamic.
- We build an on-demand video processing system for a network of mobile devices and a video-cloud. Experimental results verify the performance of the designed algorithms on the testbed.

The rest of this paper is organized as follows. Section II reviews related work. Section III gives the overview. The greedy algorithm is presented in Section IV, followed by the adaptive algorithm in Section V. Section VI evaluates the performance. Section VII concludes the paper.

II. Related Work

The proliferation of mobile devices with cameras, such as smartphones and tablets, has substantially increased the prevalance of images and videos. Images and videos taken by mobile devices create opportunities for many applications and have attracted considerable attention from research communities. Much of the research focuses on images. Yan *et al.* [2] studied real-time image search on smartphones. Qin *et al.* [3] investigated tagging images, integrating information of people, activity and context in a picture. Wang *et al.* [4] optimized the selection of crowdsourced photos based on the metadata of images including GPS location, phone orientation, etc. Hua *et al.* [5] designed a real-time image sharing system for disaster environments. Likamwa *et al.* [6] investigated the energy optimization of image sensing on smartphones.

Some work focuses on videos. Ra *et al.* [7] designed a system for real-time video processing with computation offload. Simoens *et al.* [8] designed a system for continuous collection of crowdsourced videos from mobiles devices using cloudlets. Jain *et al.* [9] proposed video-analytics for clustering crowdsourced videos by line-of-sight. Chen *et al.* [10] designed a response system for uploading crowdsourced videos. However, none of these works consider on-demand information retrieval from videos of networked mobile devices.

Mobile cloud computing bridges the gap between the limitations of mobile devices and increasing mobile multimedia applications. Mobile devices can potentially perform offloading of computational workloads to either improve resource usage or augment performance. MAUI [11] and ThinkAir [12] are the system frameworks to support method-level computation offloading by code migration. Dynamic execution patterns and context migration is investigated for code offloading in [13]. Virtual Machine synthesis is exploited for offloading in [14], [15]. A few works focus on the latency of mobile offloading. Wang *et al.* [16] considered reducing task completion by adaptive local restart. Kao *et al.* [17] optimized the latency with energy constraints by task assignment of mobile offloading. Unlike existing work that focuses on workload offloading from individual mobile devices, the major focus of this paper is to optimize the latency of video processing across multiple mobile devices and video-clouds through video offloading.

III. Overview

A. The Big Picture

We consider a wireless network that consists of mobile devices and video-clouds, where mobile devices can directly communicate with video-clouds via wireless links. When an information retrieval query is initiated, videos on the nodes related to the query need to be processed to answer the query. Note that when we say node or network node, it refers to either a mobile device or a video-cloud. In such networks, queries can be easily disseminated in the network and then parsed at each node to find the related videos, *e.g.*, based on metadata of videos, such as GPS location and timestamp. The dissemination and parsing of the queries is important to this process but is not the focus of this paper.

Since mobile devices have limited computational capability, processing videos on mobile nodes may result in long processing times, especially when there are many videos to be processed. Therefore, besides processing videos locally, mobile devices can also offload videos to video-clouds and process videos remotely. However, the offload process incurs other delays, *e.g.*, the processing delay at the video-cloud and communication delay. Moreover, we consider deep learning for video processing. Although deep learning (*e.g.* convolutional neural networks) can be greatly accelerated by GPU using parallel computing, processing even a single video will fully occupy a GPU and thus videos have to be processed sequentially. Therefore, when a video-cloud is busy processing a video, it has to put other videos into a queue. Moreover, we do not consider mobile to mobile offloading since mobile devices have similar computational capacity and such offloading rarely benefits when considering these delays together.

Moreover, due to the constraints of video processing techniques (*e.g.*, the feature extraction for action recognition requires all frames be available beforehand [18]), nodes can process videos only when the video has been fully received [1]. Considering this together with the limitation of wireless link capacity, when more than one mobile device needs to offload videos to the same video-cloud, it is desirable to transmit the videos sequentially rather than in parallel such

[1]Very large videos can be easily segmented into smaller videos by preprocessing based on the change of scene or context for storing and transmission. We assume that the videos of mobile devices have already been preprocessed. More sophisticated optimization incorporating with pre-processing is to be considered in future work.

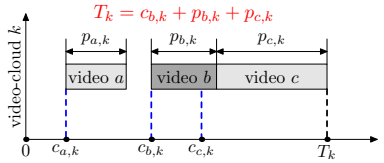


Fig. 1: An example of calculating the completion time of video-clouds

that the video-cloud can process videos early. Similarly, each mobile device should offload videos sequentially as well. For example, assuming a node needs to transmit two videos with the same size to another node and transmitting one video costs time t, if the two videos are transmitted one by one, the receiver can start processing the first video at t and the second video at $2t$. However, if the two videos are sent out simultaneously, the receiver can only start processing at time $2t$. In addition, it is possible that different video-clouds are receiving videos from different mobile devices simultaneously. This can be accomplished by assigning different wireless channels at video-clouds so as to avoid potential interference. These constraints on video processing and communications extremely complex the problem of processing videos throughout a wireless network, specifically, when we aim to take advantage of video-clouds to optimize the query response time.

B. Problem Definition

To minimize the query response time, which is the time required to process all the videos related to the query, we need jointly consider several factors: which nodes should process which videos, and what transmission sequence to perform the video offloading, as each node can only transmit (or receive) one video at a time. The *processing scheduling* problem is to find such a video offloading and transmission sequence that minimizes the query response time. The processing scheduling problem is NP-hard, which can be proved by reduction to *machine scheduling* [19]. Considering the special case where the communication delay of videos is zero, processing scheduling can be seen as a generalization of machine scheduling with the constraint that certain jobs can be only scheduled on some machines (*i.e.*, videos stored at a mobile device can only be processed at this mobile device or remotely at video-clouds). Thus, processing scheduling is NP-hard in the strong sense. We do note that there is past work on machine scheduling, considering different constraints. However, to the best of our knowledge, they do not consider the cost of scheduling a job (*i.e.*, the communication delay of an offloaded video) as a part of the completion time at the scheduled machine. We will show the performance of the scheme that does not consider the communication delay in Section VI.

Let V represent the set of videos stored in the network and related to the query, and let U denote the set of nodes in the network. U_c denotes the set of video-clouds and U_d denotes the set of mobile devices, where $U = U_c \cup U_d$. The query response time $\mathbf{T}_{\max}$ is the maximum time to complete processing of the assigned videos among all of the nodes. *For video-clouds, the assigned videos are the videos stored locally and the videos scheduled to be offloaded from mobile devices. For mobile devices, the assigned videos are the locally stored videos excluding offloaded videos.* Let T_k, $k \in U$ denote the completion time of node k and then $\mathbf{T}_{\max} = \max_{k \in U} T_k$. The processing scheduling problem is to minimize $\mathbf{T}_{\max}$.

C. Completion Time

First, we investigate how to calculate the completion time of nodes with the assignment of videos. Each video i assigned at node k, has processing delay $p_{i,k}$ and communication delay $c_{i,k}$. Note that $p_{i,k}$ may vary across different types of queries that require different video processing solutions; and $c_{i,k}$ is the time from the initiation of the query to when node k fully receives video i. Note $c_{i,k} = 0$ represents video i is locally stored at node k.

Since videos may be scheduled to be processed by video-clouds instead of locally by mobile devices, we need to account for the communication delay incurred by the offload. As a result, the completion time of video-clouds is not simply equal to the sum of processing delay of assigned videos. Further, a video-cloud, say k, may also spend time waiting for assigned videos. Therefore, for each video assigned to k, we check if the video offload to k is completed before k finishes processing existing videos or previously received videos. If the offload is complete, the video-cloud does not incur any waiting time, otherwise, the waiting time of k for the video needs to be included in T_k. Therefore, T_k is calculated as the sum of the processing delay of videos assigned at node k and the waiting time for each video to be offloaded.

Fig. 1 is an example of calculating the completion time of video-cloud k involving the offloading of videos with various cases of processing and communication delays. In this example, with knowledge of the processing and communication delay of each video shown in the figure, the completion time for k can be calculated by $T_k = c_{b,k} + p_{b,k} + p_{c,k}$, which can be interpreted that if node k spends time waiting for a video, then the time before processing the video can be denoted by the communication delay of the video. Thus, the calculation of the completion time of nodes can be generalized as

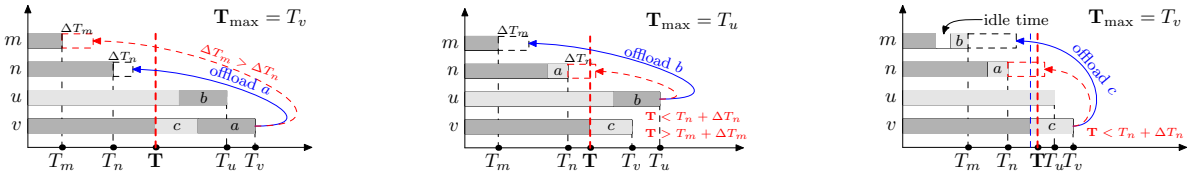
$$T_k = \max_{i \in V_k} (c_{i,k} + \sum_{j \in V_k} \alpha_{i,j,k} p_{j,k}), \tag{1}$$

where V_k denotes the set of videos assigned to node k, and $\alpha_{i,j,k} = 1$, if $c_{j,k} \geq c_{i,k}$, otherwise 0. Note that (1) can also be used to calculate the completion time of mobile devices. Since, for mobile device k, $c_{i,k} = 0$ and $\alpha_{i,j,k} = 1$ in (1), $T_k = \sum_{i \in V_k} p_{i,k}$.

D. Mathematical Formulation

Suppose $\mathbf{x}$ is a solution from the solution space for processing scheduling, where $\mathbf{x}$ determines which videos each mobile device should offload, to which video-clouds these videos should be sent, and the transmission sequence of all the offloaded videos. The problem then can be formulated as

$$\begin{aligned} \min \quad & \max_{k \in U} \max_{i \in V_k(\mathbf{x})} (c_{i,k}(\mathbf{x}) + \sum_{j \in V_k(\mathbf{x})} \alpha_{i,j,k} p_{j,k}) \\ \textit{s.t.} \quad & \alpha_{i,j,k} = 1, \text{ if } c_{j,k}(\mathbf{x}) \geq c_{i,k}(\mathbf{x}), \text{ otherwise } 0, \\ & \forall k \in U, \ \forall i, j \in V_k(\mathbf{x}), \end{aligned} \tag{2}$$



(a) offloading video a (b) offloading video b (c) offloading video c

Fig. 2: Illustration of the greedy algorithm, where m and n are video-clouds, and u and v are mobile devices.

where $V_k(\mathbf{x})$ denotes the set of videos to be processed at node k of solution $\mathbf{x}$, $c_{i,k}(\mathbf{x})$ denotes the communication delay of video i under solution $\mathbf{x}$ and $c_{i,k}(\mathbf{x})$ is subject to the constraint that each node can only send or receive one video at any time.

The processing delay of each video can be easily obtained based on the size of video, the node profile, and the execution profile of the processing method, as in [11] and [14]. Therefore, for a specific node and processing method, the processing delay is proportional to the size of the videos (discussed in Section VI). However, the communication delay not only depends on the size of videos and the transmission rate, but also the transmission sequence of the mobile devices and the receiving sequence for each of the video-clouds. For example, according to solution $\mathbf{x}$, mobile device k needs to first offload video a to a video-cloud and then transmit video b to another video-cloud m. To calculate the communication delay of $c_{b,m}(\mathbf{x})$, we need to determine when node k can start to transmit video b to m, which is actually the time when node k finishes offloading video a or the time when the video scheduled before b in the receiving sequence at m is received. Therefore, we see the calculation of communication delay is nonlinear and thus (2) cannot be further formulated by integer linear programming, which can be solved by the CPLEX optimizer. To deal with this, we propose a greedy algorithm with bounded performance to solve the processing scheduling problem.

IV. Greedy Algorithm

In this section, we describe the design of the greedy algorithm, give the performance analysis and discuss how the greedy algorithm can be easily and efficiently implemented.

A. The Algorithm

The processing scheduling problem addresses how to offload videos from mobile devices to video-clouds to minimize the maximum completion time for the entire process, which equivalently can be seen as averaging the completion time of all the nodes.

Intuitively, it is desirable for video-clouds not to be idle since they are able to process the videos faster than the mobile devices. Even more preferable is that they are processing and receiving videos simultaneously. We consider two situations in which this may occur. Initially, the video-cloud may have locally stored videos to process; therefore, it is desirable to have the mobile devices upload larger videos first. This is also true when the disparity of the completion time among the nodes is the greatest. After several offloading steps, there may be some convergence in terms of the average completion time. When the completion time among the nodes has less variability, it is better to offload videos with small size. Based on these intuitions, we design the greedy algorithm, which offloads a video from the mobile device with the maximum completion time to a video-cloud each step and improves $\mathbf{T}_{\max}$ step by step. The algorithm works as follows.

1. Calculate the completion time for each node according to (1), and then calculate the average completion time of nodes, denoted as
$$\mathbf{T} = \frac{\sum_{i \in U} T_i s_i}{\sum_{i \in U} s_i}, \tag{3}$$
where s_i denotes the processing rate of node i. Note that the completion time of video-clouds may include idle time for waiting for an incoming video.
2. For the mobile device that has maximum completion time, say i, find the videos that have sizes less than or equal to $(\mathbf{T}_{\max} - \mathbf{T})s_i$. Note that $\mathbf{T}_{\max} = T_i$.
3. Then, this video set is iterated from large to small to find the first pairing of video and video-cloud such that, if the video is offloaded to the video-cloud, it has the minimal increase in completion time among all video-clouds and its completion time is still less than or equal to $\mathbf{T}$.
4. If there is no valid pair, select the smallest video on mobile device i and offload it to the video-cloud, say m. Video-cloud m is chosen such that the completion time of m is minimal among all video-clouds and $T_m < \mathbf{T}_{\max}$ after the offloading of the video.
5. If $T_m \geq \mathbf{T}_{\max}$ (*i.e.*, $\mathbf{T}_{\max}$ cannot be reduced by offloading videos from mobile devices to video-clouds), the process stops; otherwise iterate the process from step one.

Let us use Fig. 2 as an example to illustrate the algorithm. There are four nodes in the network, where m and n are video-clouds and u and v are mobile devices. First, each node calculates its own completion time. In Fig. 2a, since no videos have been offloaded, the completion time is simply the sum of the processing delay of videos. Then, we calculate $\mathbf{T}$ according to (3). In (3), $\sum_{i \in U} T_i s_i$ can be seen as the sum of workload at each node and $\sum_{i \in U} s_i$ is the processing power of all nodes. Thus, $\mathbf{T}$ is the weighted average completion time, assuming that the future offloading of videos does not incur any idle time on any video-clouds and videos can be fragmented to any sizes. Therefore, $\mathbf{T}$ can be seen as a criterion to determine video offloading at each step, which avoids overloading the video-cloud.

As aforementioned, videos that are offloaded from mobile device v should be smaller than $(T_v - \mathbf{T})s_v$. In Fig. 2a, these

videos are a and c. For video offloading, we first consider the increase of the completion time of video-clouds (*i.e.*, the joint consideration of the workload at the video-cloud and the communication delay of the video). Moreover, we consider the completion time itself. If it is longer than $\mathbf{T}$ after the assignment of the video, we should choose a smaller video. In Fig. 2a, since $D_a > D_c$, where D denotes the size of the video, video a is considered for offloading first. Although video-cloud n has more workload than m, the offloading of video a results in less increase in the completion time for n than for m (*i.e.*, $\Delta T_n < \Delta T_m$ and $T_n + \Delta T_n < \mathbf{T}$), so video a is offloaded to video-cloud n.

After that, we recalculate $\mathbf{T}$. Since the offloading of video a does not incur any idle time at video-cloud n, $\mathbf{T}$ is the same as before. As in Fig. 2b, currently, $\mathbf{T}_{\max} = T_u$ and thus the video offloading will be from mobile device u. Although n has more workload than m (which means n may not have to be idle in waiting for b), the offloading of video b to n incurs more communication delay than m; *i.e.*, $c_{b,n} = c_{a,n} + D_b/r_{u,n}$, where $r_{u,n}$ denotes the transmission rate between u and n, while $c_{b,m} = D_b/r_{u,m}$, assuming $r_{u,m} = r_{u,n}$. Since $\mathbf{T} < T_n + \Delta T_n$ and $\mathbf{T} > T_m + \Delta T_m$, video b will be offloaded to video-cloud m.

Due to the idle time of m incurred by the offloading of video b, $\mathbf{T}$ increases as shown in Fig. 2c. To determine the assignment of videos, the processing delay at video-clouds can be easily calculated, but the communication delay is more complicated to compute as discussed before. For example, in Fig. 2c, $c_{c,m} = \max\{c_{a,n}, c_{b,m}\} + D_c/r_{v,m}$. So, video c is assigned at m rather than n since $T_n + \Delta T_n > \mathbf{T}$. The algorithm terminates with this offloading as the remaining two videos on mobile devices are large and the offloading of these videos can no longer reduce $\mathbf{T}_{\max}$.

The processing scheduling problem can be seen as balancing the completion time at each node. Thus, $\mathbf{T}$ is employed as a criterion for video offloading at each iteration, since $\mathbf{T}$ can be treated as the optimal average completion time. Moreover, at each step, we consider the increase of the completion time at video-clouds, which is a joint consideration of the incurred communication delay and idle time at video-clouds. Therefore, by regulating video offloading by $\mathbf{T}$ and minimizing the increase of completion time at video-clouds, the greedy algorithm can reduce $\mathbf{T}_{\max}$ step-by-step towards the optimal.

B. Performance Analysis

For each offloading step, the greedy algorithm attempts to minimally increase the completion time for the video-cloud. However, when the completion time with the minimal increase is more than $\mathbf{T}$, the greedy algorithm chooses to balance the completion time among video-clouds to avoid overload. Moreover, due to the heterogeneity of processing rates and transmission rates, it is hard to give a tight bound on the performance of the greedy algorithm. However, we try to give some insights on the algorithm performance with the variability of these rates.

Let t be the last time when all video-clouds are busy (idle time does count as busy), $x = \frac{\sum_{i \in V} D_i}{\sum_{j \in U} s_j}$ and let y denote the processing delay of the video with the largest size at the video-cloud, which is scheduled to process the last offloaded video (assuming that $\mathbf{T}_{\max}$ is determined by a video-cloud). Since $\mathbf{T}$ is explored as a criterion to determine video offloading, together with (1), we have

$$t \leq x + \sum_{i \in U_c} \frac{x s_i}{r_i}, \tag{4}$$

where, to simplify the analysis, we assume that videos offloaded at a video-cloud i are transmitted at a constant rate r_i from mobile devices. In (4), $\sum_{i \in U_c} \frac{x s_i}{r_i}$ gives the worse case of communication delay. Moreover, the last video offloading of the greedy algorithm minimizes the completion time at the assigned video-cloud among all video-clouds. Therefore,

$$\mathbf{T}_{\max} \leq x + \sum_{i \in U_c} \frac{x s_i}{r_i} + y + \sum_{j \in U_c} \frac{y s_j}{r_j}. \tag{5}$$

Let $\mathbf{T}^*$ denote the optimal maximum completion time and clearly we have

$$x \leq \mathbf{T}^*$$
$$y \leq \mathbf{T}^*.$$

Together with (5), we have

$$\mathbf{T}_{\max} \leq 2\mathbf{T}^*(1 + \sum_{i \in U_c} \frac{s_i}{r_i}). \tag{6}$$

From (6), when the processing rate of video-clouds is high, the communication delay has a great impact on the completion time of video-clouds. Thus, the approximation ratio goes up. When the transmission rate is high, the processing delay dominates the completion time and then the approximation ratio approaches 2. Although the bound on the performance is not tight, as shown in Section VI, the greedy algorithm performs much better than this bound.

For the computational complexity, as one video is offloaded during each iteration, there are at most $|V|$ iterations for the greedy algorithm. For each iteration, the videos stored at the mobile device with the maximum completion time are iterated over video-clouds to minimize the increase in completion time. Therefore, the computational complexity of the greedy algorithm is $O(|U||V|^2)$.

C. Discussion

The greedy algorithm is a centralized approach and requires the information of all the videos *a priori*. When a query is initiated, the information (*e.g.*, data size) about videos stored in the network and related to the query needs to be collected at one node, *e.g.* a video-cloud, to run the greedy algorithm. The solution is then sent to the other nodes. Alternatively, the information can be collected at each node and each node may run the greedy algorithm. This is feasible, since the information collected is small and the computational complexity of the algorithm is low.

The solution of the processing scheduling problem determines which videos are offloaded between mobiles and video-clouds. It also determines the transmission sequence, but this sequence is shown not to be trivial. For example, in Fig. 2, for mobile device v, the sending sequence is a and then c.

However, v may not transmit c immediately after a; it must be transmitted after m receives video b from u. Therefore, when there is a video for which the mobile device cannot locally determine the transmission start time, the receiving video-cloud will inform the mobile device when it is ready to receive. Although such coordination incurs additional communication overhead (and idle time), the overhead is low since there is at most one message for each offloaded video.

The greedy algorithm is designed for the scenario where mobile devices and video-clouds are stationary (e.g., surveillance systems) and the transmission rate between them is steady (or varies slightly). To cope with the scenario with high dynamics of transmission rate, we further propose an adaptive algorithm.

V. Adaptive Algorithm

In this section, we consider the case where the transmission rate between mobile devices and video-clouds dynamically changes during the on-demand querying of videos process (but assume that all nodes stay connected to the network during the process). Due to the dynamics of the transmission rate, the communication delay of offloaded videos also varies. This makes the processing scheduling problem more difficult, because we do not know how the transmission rate changes *a priori*. Since the communication delay of an offloaded video is only known after the transmission of the video is completed, it is better to determine video offloading in realtime in such scenarios. Therefore, we propose an adaptive algorithm that makes video offloading decisions during runtime, through consideration of the transmission rate, the communication delay and the completion time.

A. The Algorithm

We assume the same query is issued to the network of mobile devices and video-clouds. Unlike the greedy algorithm which determines video offloading before processing any videos, the adaptive algorithm offloads videos from mobile devices to video-clouds in realtime.

Intuitively, to offload videos in runtime, the designed algorithm should gradually reallocate videos from mobile devices, balance the workload among video-clouds, and prevent video-clouds from being overloaded. Moreover, the adaptive algorithm should not incur too much communication overhead, which would delay the video transmission. Based on these considerations, the adaptive algorithm is designed to adapt to the dynamics of transmission rate and reduce $\mathbf{T}_{\max}$ dynamically as videos arrive and others are being processed.

To describe the adaptive algorithm, we first give the overall workflow and then detail how the video-cloud decides whether to accept offload requests from mobile devices and how the mobile device decides to which video-cloud to offload the video based on replies from video-clouds.

Upon receiving the query, each node identifies locally stored videos related to the query. Then, it broadcasts the information about these videos to other nodes and starts to process videos. For processing, each mobile device continuously processes videos from small to large in size. Each video-cloud can process any video it currently has in any order as the order will not impact the completion time on the video-cloud.

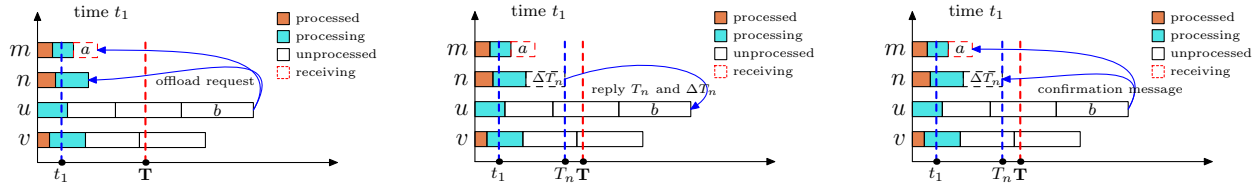
For video offloading, each time a mobile device offloads the largest video, for which it has not completed processing (*i.e.*, it is possible to offload the video that is being processed). When a mobile device is ready to offload videos (*i.e.*, it is not transmitting any video), it will broadcast an offload request to inform all the video-clouds. When video-clouds receive an offload request, they will add the request into a set of unhandled requests. If the mobile device just finished offloading another video before sending out the request, video-clouds will acknowledge the actual communication delay of the previously offloaded video and update the information of the video-cloud that received that video.

When a video-cloud is ready to receive videos, (*i.e.*, it is not receiving any video), it will determine whether to accept the requests it has received and reply the accepted request. Based on the replies from video-clouds, the mobile device will eventually determine to which video-cloud the video should be offloaded. After making the decision, the mobile device will broadcast a confirmation message to video-clouds to inform them of the selected video-cloud and the estimated communication delay of the video, and then start transmitting the video. When other video-clouds receive the message, they will mark the offload request from the mobile device as handled and then update the locally stored information of the mobile device and the chosen video-cloud, *i.e.*, change the location of the video from the mobile device to the video-cloud and add the estimated communication delay for the video.

This process continues until all videos are processed.

A video-cloud needs to decide whether to accept received requests when it is ready to receive videos, and a mobile device needs to decide to which video-cloud to offload the video based on the replies from video-clouds. The algorithm works as follows.

1. A video-cloud, say m, which is not currently receiving a video, calculates the completion time of each node based on the collected information at that time, and then calculates $\mathbf{T}$ according to (3).
2. From the set of unhandled requests, it selects the request from the mobile device, u, that has the maximum completion time among the set. Then, using the current transmission rate between the mobile device and itself, which can be estimated based on signal strength, signal-to-noise ratio, etc., video-cloud m calculates the completion time T_m and the increase ΔT_m if the video is offloaded to m.
3. If $T_u = \mathbf{T}_{\max}$, video-cloud m will accept the offload request when $T_m < \mathbf{T}_{\max}$ and then send T_m and ΔT_m to u. If $T_u < \mathbf{T}_{\max}$, video-cloud m will accept the request of u only if $T_m \leq \mathbf{T}$, otherwise, m will skip the request.
4. Mobile device u may receive multiple replies at the same time. It will choose the video-cloud that has the minimal completion time if the received completion times are more than $\mathbf{T}$. Otherwise, it will select the video-cloud whose completion time is less than $\mathbf{T}$ such that the



(a) u sends out an offload request (b) n replies u with T_n and ΔT_n (c) u confirms b will be sent to n

Fig. 3: Illustration of the adaptive algorithm, where m and n are video-clouds, and u and v are mobile devices.

increase in the completion time of the chosen video-cloud is minimal.

5. After mobile device u selects the video-cloud, it will broadcast a confirmation message. When video-clouds receive the message, they will update locally stored information accordingly as discussed above. The unselected video-clouds that are ready to receive videos will continuously process the unhandled requests if the request set is not empty.

See Fig. 3 as a simple example to illustrate the adaptive algorithm. As in Fig. 3a, at time t_1, mobile device u is ready to offload videos and thus it sends out an offload request of video b to video-clouds m and n. Since m is currently receiving video a, it will add the request into the set of unhandled requests. Since n is not currently receiving a video, it will calculate T_n and ΔT_n if video b is offloaded to itself based on the current transmission rate between n and u, and then send them to u as shown in Fig 3b. When u receives the reply, it will decide to offload b to n, because it only gets one reply. Before offloading b to n, it will first send out a confirmation message as in Fig. 3c. When n receives the confirmation message, it will setup the connection to receive b. Meanwhile when m receives the message, it will mark the offload request from u as handled.

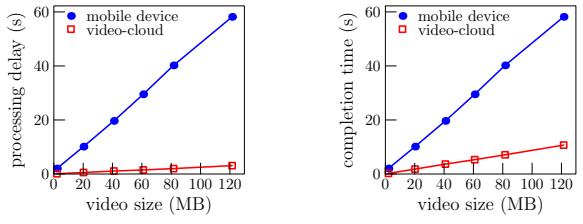
B. Discussion

Since, typically, there are more mobile devices than video-clouds in the network, a video-cloud is most likely to decide whether to accept a request when it finishes receiving a video rather than when it receives an offload request. As the video-cloud selects the request of the mobile device that has the maximum completion time among the set of unhandled request, the adaptive algorithm will gradually decrease $\mathbf{T}_{\max}$ by handling each offload request until it cannot be reduced.

The confirmation message from a mobile device is designed to inform video-clouds that the offload request has been handled and the estimated communication delay of the video to be offloaded, which will be used to calculate $\mathbf{T}$ at each video-cloud when it handles other offload requests. The communication delay is estimated based on the transmission rate at the beginning of offloading each video. Since the transmission rate may vary during offloading, the actual communication delay will be different than what is estimated. However, each video-cloud will be notified of the completion of each video offload (by the offloading mobile device) and then the other video-clouds can update their previously received estimation by the actual communication delay. Therefore, the difference between the actual completion time at each video-cloud and the estimated will only vary by the actual communication delay of one video. Thus, it only slightly impacts the criterion $\mathbf{T}$ and the performance of the adaptive algorithm.

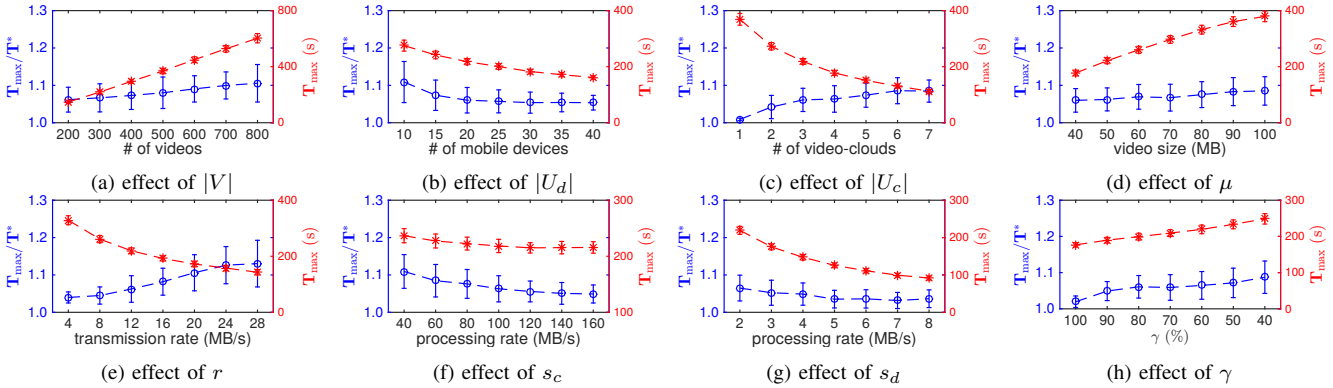
As message overhead can delay video offloading, the adaptive algorithm is designed to produce messages with as little overhead as necessary. At the beginning of video processing, each node will broadcast a message including the information of locally stored videos and thus there will be $|U|$ messages. As discussed before, the video-cloud will most likely handle the request after receiving a video, and thus there is most likely one reply for each request. Therefore, for each offloaded video, there will be three messages, *i.e.*, request, reply and confirmation. In the worse case that all videos are offloaded to video-clouds, the overall message overhead of the adaptive algorithm is $3|V| + |U|$. The small number of messages is sufficient to obtain all the information to determine video offloading. Moreover, a node needs to compute the completion time of all nodes when it (for video-clouds) decides to accept the offload request or when it (for mobile devices) selects the video-cloud. However, the computation overhead is low, i.e. $|V|$. For the worst case that all videos are offloaded to video-clouds, the sum of computation overhead of all nodes is $2|V|^2$.

As video-clouds can also communicate with each other, we could consider transfer of videos among video-clouds to balance the workload. However, we decided against this because video offloading among video-clouds incurs additional communication delay. That means a video might be transferred multiple times before being processed and thus increase the communication delay. As a result, it might also increase the communication delay of other videos due to the constraint that each node can only send or receive one video at a time. However, the adaptive algorithm requires only one transfer for each offloaded video, and instead of balancing workload by transferring videos among video-clouds, it balances the



(a) processing delay vs video size (b) completion time vs video size

Fig. 4: Processing delay and completion time of videos with different sizes for mobile device and video-cloud, where videos have the resolution 1920×1080, bit rate 16Mbps, frame rate 30fps, and the transmission rate between mobile device and video-cloud is 16MBps.



(a) effect of $|V|$ (b) effect of $|U_d|$ (c) effect of $|U_c|$ (d) effect of μ

(e) effect of r (f) effect of s_c (g) effect of s_d (h) effect of γ

Fig. 5: Comparison between greedy algorithm and optimal solution in terms of $\mathbf{T}_{\max}/\mathbf{T}^*$ and the value of $\mathbf{T}_{\max}$, where the default setting is $|V| = 300$, $|U_d| = 20$, $|U_c| = 3$, $\mu = 50$MB, $\sigma = 20$MB, r =12MB/s, $s_d = 2$MB/s, $s_c = 100$MB/s and $\gamma = 0.6$.

workload when offloading videos from mobile devices to video-clouds.

The adaptive algorithm estimates the communication delay of each offloaded video based on the transmission rate just before offloading and makes video offloading decision in realtime. Therefore, it is more suitable for the scenarios where the transmission rate is dynamic during video processing.

VI. Performance Evaluation

In this section, we first evaluate the proposed algorithms by extensive simulations based on the measurements of an on-demand video processing system, and then we investigate the system performance on a small testbed.

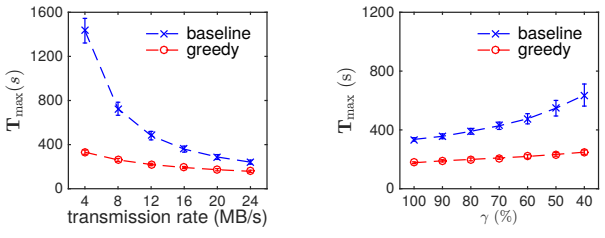
A. Processing Delay

First, we evaluate the processing delay of videos in terms of data size on mobile devices and video-clouds. We implemented our video processing approach for object detection and recognition based on Caffe [20], a deep learning framework using convolutional neural networks, on both tablets (Nexus 9) and a video-cloud implementation (Dell Precision T7500 with GeForce GTX TITAN X 12 GB GPU) for processing acceleration. We took several videos with different sizes using the tablet and processed them on both the tablet and video-cloud. Fig. 4a gives the comparison of the processing delay between the tablet and video-cloud. From Fig. 4a, we can see that GPU can greatly accelerate video processing. The processing rate on the GPU is about 100MB/s, while the processing rate of the smartphone is only about 2MB/s. Both linearly increase with the data size of videos. When taking the communication delay of videos into consideration, as shown in Fig. 4b, the completion time of processing each video (offloaded from the tablet) on the video-cloud is still much less than that of the tablet. Note that the specifications of videos, such as resolution, frame rate and bit rate, may affect the processing delay. However, mobile devices have similar camera sensors and can be easily adapted to take videos with the same specifications.

B. Greedy Algorithm vs. Optimum

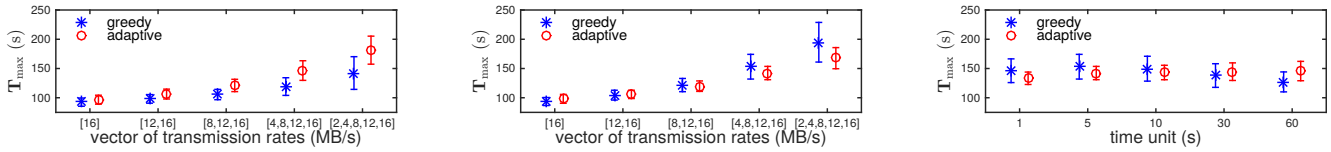
In order to evaluate the performance of the proposed algorithms, we setup a simulation environment. The videos are generated with different data sizes following normal distributions with different μ and σ. To capture the heterogeneity of the processing rate, the processing rates of mobile devices and video-clouds are set uniformly and randomly to between $[\gamma s_d, s_d]$ and between $[\gamma s_c, s_c]$, respectively, where s_d denotes the maximum processing ratio for mobile devices and s_c denotes the maximum processing rate of video-clouds. Also, the transmission rate between a mobile device and a video-cloud is set uniformly and randomly to $[\gamma r, r]$. The number of videos $|V|$, the number of mobile devices $|U_d|$, the number of video-clouds $|U_c|$, r, μ, σ, γ, s_d and s_c are system parameters for simulations. The default settings of these parameters are $|V| = 300$, $|U_d| = 20$, $|U_c| = 3$, $r = 12$MB/s, $\mu = 50$MB, $\sigma = 20$MB, $\gamma = 0.6$, $s_d = 2$MB/s and $s_c = 100$MB/s, where the settings of s_d and s_c correspond to the implementation measurement in the previous section.

We evaluate the greedy algorithm and compare it with the optimum achieved by an exhaustive search in various settings. For each setting, we generate one hundred instances according to the randomness of simulation setup. The two solutions run on these instances. The performance is compared in terms of $\mathbf{T}_{\max}/\mathbf{T}^*$ to demonstrate how the greedy algorithm approximates the optimum, and the value of $\mathbf{T}_{\max}$ is also illustrated. Fig. 5 demonstrates the effects of system parameters on the performance of the greedy algorithm. For each evaluated parameter, all other parameters use the default settings. From Fig. 5a, we can see $\mathbf{T}_{\max}/\mathbf{T}^*$ slightly increases with the increased number of videos. When using 200 videos, the greedy algorithm is less than 10% worse than the optimum, and it is less than 20% when using 800 videos. The increase



(a) effect of r (b) effect of γ

Fig. 6: Comparison between greedy algorithm and baseline in terms of $\mathbf{T}_{\max}$, where the default setting is $|V| = 300$, $|U_d| = 20$, $|U_c| = 3$, $\mu = 50$MB, $\sigma = 20$MB, r =12MB/s, $s_d = 2$MB/s, $s_c = 100$MB/s and $\gamma = 0.6$.



(a) effect of static transmission rates (b) effect of dynamic transmission rates (c) effect of time unit

Fig. 7: Comparison between adaptive algorithm and greedy algorithm in terms of $\mathbf{T}_{\max}$, where the default setting is $|V| = 100$, $|U_d| = 10$, $|U_c| = 2$, $\mu = 50$MB, $\sigma = 20$MB, $s_d = 2$MB/s, $s_c = 100$MB/s, $\gamma = 0.6$, $t = 5$s, and $R = [4, 8, 12, 16]$.

is caused by increased video offloading when there are more videos to be processed. Correspondingly, when there are more mobile devices in the network, each mobile device has fewer videos to process and thus less video offloading. Therefore, the greedy algorithm performs better as the number of mobile device increases in Fig. 5b. When there is only one video-cloud in the network, the greedy algorithm achieves the optimum shown in Fig. 5c. The difference rises when the number of video-clouds goes up, but it tends to flatten out when the number of video-cloud increases further.

In Fig. 5d, the greedy algorithm performs close to the optimum in the settings with different average video sizes. Fig. 5e demonstrates the effect of transmission rates. When the transmission rate increases, mobile devices tend to offload more videos to video-clouds as offloading videos costs less than before. This leads to an increased deviation between the greedy algorithm and the optimum, although both $\mathbf{T}_{\max}$ and $\mathbf{T}^*$ decrease. The completion time of video-clouds is determined based on the processing delay and communication delay of videos. When the processing rate of video-clouds increases, the processing delay decreases and thus the greedy algorithm performs better as shown in Fig. 5f. Moreover, $\mathbf{T}_{\max}/\mathbf{T}^*$ also declines when mobile devices are more computationally powerful as indicated in Fig. 5g, because fewer videos are offloaded when mobile devices have higher processing rates. The effect of the diversity of processing rates and transmission rates is captured in Fig. 5h; *i.e.*, such diversity leads to slightly increased $\mathbf{T}_{\max}$ and deviation from the optimum.

In summary, through extensive simulations, we can see that the performance of the greedy algorithm is close to the optimum in various settings (no more than 20% worse than the optimum) and it is much less than the theoretical upper bound as in (6).

C. Greedy Algorithm vs. Baseline

We also compare the greedy algorithm with a *baseline* scheme that does not consider communication delay and iteratively offloads a video from the mobile device that has the maximum completion time to the video-cloud that has the minimum. As illustrated in Fig. 6, the greedy algorithm performs much better than the baseline. When the transmission rate increases, the impact of the communication delay on the completion time decreases and thus the difference between these two algorithms narrows, as shown in Fig. 6a. Moreover, the baseline is more sensitive to the increased diversity of processing rates and transmission rates as indicated in Fig. 6b. Therefore, we can conclude that the greedy algorithm that considers both processing delay and communication delay is much faster than the baseline that considers only processing delay.

D. Adaptive Algorithm vs. Greedy Algorithm

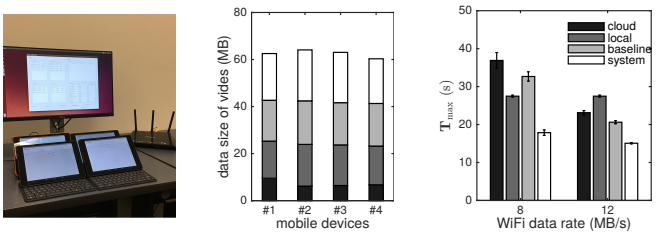
The adaptive algorithm is designed for the scenarios where the transmission rate varies during video processing. To model the dynamics of the transmission rate, we also adopt a Markov chain [21]. Let R denote a vector of transmission rates $R = [r_0, r_1, \ldots, r_l]$, where $r_i < r_{i+1}$. The Markov chain moves at each time unit. If the chain is currently in rate r_i, then it can change to adjacent rate r_{i-1} or r_{i+1}, or remain in the current rate with the same probability. Therefore, for a given vector, *e.g.*, of four rates, the transition matrix can be defined as

$$P = \begin{array}{c} \\ r_0 \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cccc} r_0 & r_1 & r_2 & r_3 \end{array} \\ \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix} \end{array}.$$

In the simulations, the transmission rate between mobile device and video-cloud is initially set to a randomly selected rate from R and it dynamically changes according to the transition matrix each time unit t. The greedy algorithm determines video offloading and transmission sequence based on the initially assigned transmission rates before processing videos. Then, the simulation runs and produces the runtime $\mathbf{T}_{\max}$ for the greedy algorithm. The adaptive algorithm runs during video processing and determines video offloading during runtime of simulations.

First, we compare the adaptive algorithm with the greedy algorithm under static transmission rates. As shown in Fig. 7a, the greedy algorithm outperforms the adaptive algorithm in various vectors of transmission rates. Moreover, the difference between the greedy algorithm and adaptive algorithm expands with the increased diversity of transmission rates. In the adaptive algorithm, video-clouds can only accept the offload request after receiving previously offloaded video to adapt to the variation of transmission rate. Therefore, when a mobile device selects a video-cloud for offloading, the video-clouds that are currently receiving videos are not considered. However, the greedy algorithm makes offloading decisions beforehand and considers every video-cloud at each step. Therefore, the greedy algorithm performs better under static transmission rates.

When transmission rates change dynamically, the performance of the greedy algorithm and the adaptive algorithm is shown in Fig. 7b, where the time unit $t = 5$s. When transmission rates are more stable, *e.g.*, $R = [16]$ or $[12, 16]$,



(a) testbed (b) sizes and distribution of videos (c) completion time of different algorithms

Fig. 8: Performance of different algorithms on a small testbed.

the greedy algorithm performs better than the adaptive algorithm. When transmission rates are more dynamic, *e.g.*, $R = [8, 12, 16]$, $[4, 8, 12, 16]$ or $[2, 4, 8, 12, 16]$, the adaptive algorithm outperforms the greedy algorithm. Fig. 7c gives the performance comparison in terms of time unit of the Markov chain. As short time intervals produce a dynamic transmission rate during video processing, the adaptive algorithm performs better when time interval is short, and vice versa.

In summary, as expected, the greedy algorithm is preferred for the scenarios where the transmission rate is steady, while the adaptive algorithm is more suitable for the scenarios where the transmission rate is dynamic.

E. System Performance

We implemented an on-demand video processing system on a small testbed that includes four Nexus 9 tablets and the video-cloud implementation which are connected through a WiFi router, as shown in Fig. 8a. Both the tablets and video-cloud are running a same deep learning model using Caffe for object detection on videos. The video-cloud can issue queries with a targeted object to tablets. For video processing, frames are extracted from a videos and then object detection are performed on the frames.

The performance is measured under two different WiFi data rates (i.e., 8MB/s and 12MB/s). Since the data rate is stable in our test environment, our *system* performs the greedy algorithm rather than the adaptive algorithm to achieve the best performance. We compare it to *local* (videos are processed locally), *cloud* (all videos are offloaded to the video-cloud for processing) and *baseline*. Experiments are performed on a small set of videos (16 clips with bit rate about 16Mbps and frame rate 30fps, average size 15 MB). In order not to introduce any bias on these approaches, we distribute the same number of videos, each with similar size, on each tablet. The sizes and the distribution of the videos are shown in Fig. 8b. As illustrated in Fig. 8c, our *system* outperforms all other approaches for both WiFi data rates. Note that the greedy algorithm is optimal when there is one video-cloud in the network as discussed in Section VI-B.

VII. Conclusion

In this paper, we investigated on-demand video processing in wireless networks. We formulated the processing scheduling problem, *i.e.*, to process videos with the minimal query response time in the network consisting of mobile devices and video-clouds. However, the processing scheduling problem is a strongly NP-hard problem. To deal with this, we designed a greedy algorithm and proved the approximation ratio. To handle the dynamics of the transmission rate between mobile devices and video-clouds, we further proposed an adaptive algorithm. Extensive simulations and experiments on a small testbed show that, as expected, the performance of the greedy algorithm is close to the optimum and much better than other approaches, and the adaptive algorithm performs better when the transmission rate is more dynamic.

References

[1] Y. Jiang, X. Xu, P. Terlecky, T. Abdelzaher, A. Bar-Noy, and R. Govindan, "Mediascope: selective on-demand media retrieval from mobile devices," in *IPSN*, 2013.

[2] T. Yan, V. Kumar, and D. Ganesan, "Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones," in *MobiSys*, 2010.

[3] C. Qin, X. Bao, R. Roy Choudhury, and S. Nelakuditi, "Tagsense: a smartphone-based approach to automatic image tagging," in *MobiSys*, 2011.

[4] Y. Wang, W. Hu, Y. Wu, and G. Cao, "Smartphoto: a resource-aware crowdsourcing approach for image sensing with smartphones," in *MobiHoc*, 2014.

[5] Y. Hua, H. Jiang, and D. Feng, "Real-time semantic search using approximate methodology for large-scale storage systems," in *INFOCOM*, 2015.

[6] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, "Energy characterization and optimization of image sensing toward continuous mobile vision," in *MobiSys*, 2013.

[7] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *MobiSys*, 2011.

[8] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *MobiSys*, 2013.

[9] P. Jain, J. Manweiler, A. Acharya, and K. Beaty, "Focus: clustering crowdsourced videos by line-of-sight," in *SenSys*, 2013.

[10] Z. Chen, W. Hu, K. Ha, J. Harkes, B. Gilbert, J. Hong, A. Smailagic, D. Siewiorek, and M. Satyanarayanan, "Quiltview: a crowd-sourced video response system," in *HotMobile*, 2014.

[11] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *MobiSys*, 2010.

[12] 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 *INFOCOM*, 2012.

[13] W. Gao, Y. Li, H. Lu, T. Wang, and C. Liu, "On exploiting dynamic execution patterns for workload offloading in mobile cloud applications," in *ICNP*, 2014.

[14] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *EuroSys*, 2011.

[15] F. Hao, M. Kodialam, T. Lakshman, and S. Mukherjee, "Online allocation of virtual machines in a distributed cloud," in *INFOCOM*, 2014.

[16] Q. Wang and K. Wolter, "Reducing task completion time in mobile offloading systems through online adaptive local restart," in *ICPE*, 2015.

[17] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," in *INFOCOM*, 2015.

[18] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *ICCV*, 2015.

[19] J. K. Lenstra, A. H. G. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[20] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *MM*, 2014.

[21] A. Fu, P. Sadeghi, and M. Médard, "Dynamic rate adaptation for improved throughput and delay in wireless network coded broadcast," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 1715–1728, 2014.