Video Compression in the Neighborhood: An Opportunistic Approach

Dimitris Chatzopoulos*, Kathleen Sucipto*, Sokol Kosta*†, Pan Hui*
*System and Media lab, Hong Kong University of Science and Technology
†Sapienza University of Rome

***Abstract*—The proliferation of mobile devices combined with advances in the area of low-power wireless communication, such as Wi-Fi Direct and Bluetooth 4.0, gave rise to a new computation paradigm known as Device–to–Device (D2D) offloading. In this scenario, devices collaborate with each other using short wireless links to create ad-hoc P2P networks for distributed task execution. Experiments on human movement, a non-negligible factor in the D2D context, have shown that people move in group or meet frequently, which suggests that D2D is possible. In this work, we examine the case of parallel compression of smartphone recorded videos with the help of nearby devices. First, we present a mathematical formulation of the problem that optimizes the compression time on the number of nearby helping devices, and show that the problem can be mapped as a water-filling problem. Then, we present real results of the compression time and energy when the compression is performed on one device and when it is parallelized among collaborating devices. To obtain these results, we implemented an Android application that is able to detect nearby devices, connect with them using Wi-Fi Direct, send video chunks for compression, receive and merge compressed chunks into one full compressed video.**

I. Introduction

State-of-the-art mobile devices are becoming increasingly powerful, but at the same time the need for computational resources and energy of mobile applications is increasing with the same pace. To be properly functional and provide good quality of experience to the users, many applications require more resources than those provided by the device. This obstacle triggered the research in Mobile Cloud Computing [1], [2], where parts of a mobile application are offloaded for execution to remote cloud servers. However, mobile cloud computing solutions require *(i)* dedicated cloud servers that can provide pre-specified functionality, and *(ii)* Internet connectivity. Moreover, these requirements imply increased monetary costs due to the utilization of the cloud facility.

In this work, we consider the case of a specific application, video compression of mobile recorded videos, which can benefit from offloading. The continuous quality increase of the smartphones' cameras allows us to take better pictures and videos, but at the same time it sacrifices device's storage. An off-the-shelf last generation smartphone requires around 1 MB per second of captured video, since the videos are stored as raw data. Compressing the video could save up to 95% of the storage, depending on the compression settings. Unfortunately, the compression requires as much time as the length of the video, which means that the device needs to spend more resources and more energy if the compression is fully performed on the device. However, splitting a video into smaller chunks is a much faster and less expensive procedure. In the same way, merging small video chunks into a full video file is a fast and cheap process. This suggests that using a technique that compresses a video file by first splitting it and then by compressing the individual parts in parallel on multiple devices could reduce the total compression time and total energy spent for compression.

For example, in our experiments we show that it takes around 100 seconds on a Xiaomi MI3 device and around 400 seconds on a Samsung Galaxy S2 device to compress a video of 100 seconds (97.44 MB) locally on the device, while it takes only 5 seconds to split the video into 10 equal pieces on both devices. Sending the chunks to nearby devices using Wi-Fi Direct, which can reach transmission rate up to 250 Mbps [3], can speed up the compression by parallelizing the process.

In this work, we examine the case of utilizing nearby mobile devices in order to parallelize the process of video compression. We consider a PNP block structure (***p***re-processing, ***n*** tasks, ***p***ost-processing), as proposed in Serendipity [4]. Our block structure is composed by *i)* the pre-processing task, that splits the video in smaller pieces, *ii)* the offloading step, that sends the video chunks to n nearby devices, and *iii)* the post-processing task, that collects the compressed chunks from the nearby devices and merges them into one video.

Any mobile user who is utilizing our service has to define a threshold that determines whenever she is willing to help others. The threshold is of the type: "*If my battery level is more than X%, I am sharing my resources*". The thresholds can be associated with multiple resources. After compressing another user's video, a user receives an acknowledgement that she can use when asking others to compress her videos.

A. Motivating Example

Let's consider a scenario with three users, Lencia, Mary, and Richard, starting from the left, as in Figure 1. Let's assume that Mary has captured a 10 minutes video with

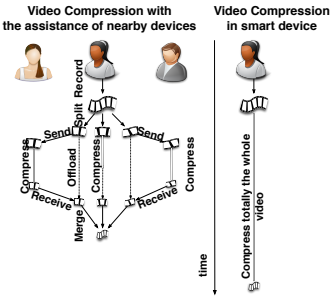


Figure 1: The user in the middle, instead of compressing the video by herself, she is asking for help from two nearby users and by doing that she decreases the required time.

her Xiaomi MI3 smartphone, which has a camera of 13 MPixel, that she wants to compress. From our experiments we found that the video file is around 600 MB. Compressing the file in her device, in the ideal case that the device was fully dedicated for compression, would take Mary's around 10 minutes. However, utilizing our solution, she can split the video into 3 chunks, offload one chunk to Lencia, one to Richard, and keep the third one for herself. Parallelizing through Device–to–Device offloading, the compression time can be reduced by almost three times, considering that video splitting and merging is much faster than compression and the network connection will be performed using Wi-Fi Direct, which promises device-to-device transfer speeds of up to 250Mbps and connection range of up to 200 meters [3]. In our system, collaborating users are unaware of the underlying video compression process, which is done transparently by the smartphone device.

One important factor that influences the chances of the Device–to–Device offloading, is the mobile nature of the users. For this system to work, devices should be connected for enough time to allow the requester device to send the video chunks to the helper devices, for the helper devices to perform the compression, and for the helper devices to send the compressed video chunks back to the requester. Studies on human mobility have shown that people move in groups and meet frequently with each other [5], [6]. As can be seen from Figure 2, which shows the Inverse Cumulative Distribution function of the contact duration between mobile users for four different datasets, the contact duration between two devices will be in the order of tens or hundreds of seconds with high probability. The plot is produced for the first day of Infocom 05 and its 41 users [6], the first day of Infocom 06 and its 78 users [6], the whole duration of Humanet, which is 1 day and has 56 users [5], and for the whole duration of MDC which is 1 month and has 64 users [7].

II. System Model

We assume a set of mobile users $\mathcal{U}$ interconnected in an ad-hoc manner and without any delay or bandwidth guarantees. Any participating user's device is able to

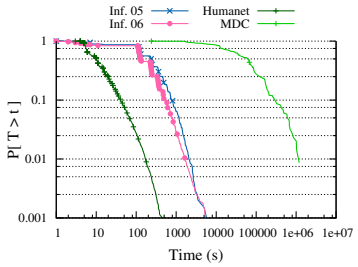


Figure 2: The inverse cumulative distribution function of the contact duration of 4 different datasets.

compress a video[1]. Function $\mathcal{C}(\cdot)$ is a compression function that takes as input any video v of length v_t seconds and size v_s MB and produces a video $\tilde{v}$ of the same duration (i.e. $\tilde{v}_t = v_t$), but with smaller size $\tilde{v}_s << v_s$:

$$\tilde{v} = \mathcal{C}(v) \tag{1}$$

It is worth mentioning that recorded videos by mobile users via their mobile devices are not processed and are stored as raw data. For example, a video of one minute in a Xiaomi MI3 device with default settings occupies 60 MB of storage. Video compression is a time and energy consuming process. According to our experiments, which are presented in Section IV, the time needed to compress a video is a linear function of the duration of the video, if the codec used is H.264 and the compression MPEG4. We use a linear function $f^{(res)}(v_t)$ to measure the time needed to compress a video v that has length v_t seconds given the available resources res.

User i has one video v^i to compress with size v_s^i. We consider the compression function as expressed in Equation 1, which is user independent and is built in any Android device. Let's assume that user i at time t has $\mathcal{N}_i(t)$ neighboring devices with whom she can connect in an ad-hoc manner. Any user j who participates in our system, following the principles of hidden market design [8], has imposed a number of conditions, in terms of thresholds, under which she is willing to share her resources in order to help others. After helping another user to compress a video, a collaborating user earns virtual credits that she can use in order to compress her videos. Virtual credits make the system work by incentivizing people to collaborate and by reducing the chances for selfish users to participate in the system. If one user does not collaborate by sharing her resources, she will soon finish her credit and will not be able to ask for help in the future. Assuming that the user k has c_k credits before helping another user, then she will have $c_k + \gamma v_t^l$ credits after compressing user's l video v^l. We use $\gamma > 0$ parameter in order to motivate users to participate in the system.

The selection of nearby devices to offload a video is not obvious. We define the probability of user i to be connected to user j at time t as $p_{ij}(t)$. Considering this probability,

[1]This functionality can be provided by tools like FFmpeg and is easily integrable in any mobile OS: https://www.ffmpeg.org/.

the connectivity conditions, and the shared resources by each nearby device, any user who wants to compress a video with the help of her nearby devices should choose the most suitable devices for helping with this task. Then, the device should split the video into smaller chunks based on the number of chosen helper devices. In the next section we present how the video can be split optimally.

III. Problem formulation

We consider a user i that has a video v^i of size v_s^i MB and is connected at time t with $\mathcal{N}_i(t)$ nearby devices. User i can split the video into smaller chunks, offload the chunks for compression to the nearby devices, and then receive and merge the compressed chunks. We use a vector x to represent the partition of the original video into chunks. The array has $|\mathcal{N}_i(t)|+1$ elements, with each element being non negative and smaller than 1, while the sum of all elements is equal to 1. If $x_l > 0$, then $x_l \cdot v_t$ corresponds to the duration in seconds of the chunk l, while are $x_l \cdot v_s$ corresponds to the size in MB of the chunk l. We assume that the chunk l, for $1 \leq l \leq |\mathcal{N}_i(t)|$, will be sent to the nearby device l, while the $|\mathcal{N}_i(t)|+1$'th chunk will be compressed by the initiator of the compression (i.e. user i). The formal problem formulation is as follows:

$$\min_x \Big(split(x) + compress(x) + merge(x) \Big) \tag{2}$$

$$\text{s.t} \sum_{l=1}^{|\mathcal{N}_i(t)|} x_l \leq \frac{c_i}{v_t^i}, \quad \sum_{i=1}^{|\mathcal{N}_i(t)|+1} x_l = 1, \;\; 0 \leq x_l \leq 1 \;\; \forall\, l \tag{3}$$

where the split and merge functions depend linearly on the number of the video chunks that are created and merged. These can be expressed as linear functions as follows:

$$split(x) = \xi_{sp}||x||_0 + \beta_{sp} \tag{4}$$

$$merge(x) = \xi_{mr}||x||_0 + \beta_{mr} \tag{5}$$

where ξ and β are the parameters of the line.Given that the compression is parallelized, minimizing the time we have to minimize the time needed to perform the slowest compression. The offloaded compression part is composed by the video transmission, the remote compression, and the reception of the compressed video. Assuming that the bandwidth between user i and the nearby use l is B_l and the resources she is providing are R_l, the compression is:

$$\begin{aligned} compress(x) &= \max_{x_l} \Big(send(x_l) + process(x_l) + receive(x_l) \Big) \\ &= \max_{x_l} \Big(x_l \cdot \frac{v_s^i}{B_l} + x_l \cdot \frac{v_t^i}{R_l} + x_l \cdot \frac{\tilde{v}_s^i}{B_l} \Big) \\ &= \max_{x_l} \left(\Big(\frac{v_s^i}{B_l} + \frac{v_t^i}{R_l} + \frac{\tilde{v}_s^i}{B_l} \Big) x_l \right) \\ &= \max_{x_l} (\gamma_l x_l) \end{aligned}$$

where γ_l is defined as the delay in seconds on every nearby device. If we define $\Gamma = diag(\gamma_l)$ then, by using the fact that the infinity norm equals to the biggest element of one vector, the compress function can be formulated as:

$$compress(x) = ||\Gamma x||_\infty \tag{6}$$

A. Mobility

Due to human mobility, contact duration between pairs of devices is variable. In our video compression application, it can happen that the requester device sends the video chunk to a helper device but then they disconnect. If they meet again after a long time, the helper device sends the compressed video chunk to the requester. This way, the respective γ_l of the helper device will be very high. We include a multiplicative penalization factor in our system for each participating device, so to avoid selecting helper devices with high values of γ_l. The penalty vector has one entry for each device, so it is of size $|\mathcal{N}_i(t)| + 1$:

$$p = \left(\frac{1}{p_{i1}}, \frac{1}{p_{i2}}, \ldots, \frac{1}{p_{i\mathcal{N}_i(t)}}, 1 \right) \tag{7}$$

with the requesting device having penalty factor equal to 1, meaning that it should not be penalized. The compression function now becomes:

$$compress(x) = ||(\Gamma + diag(p))x||_\infty = ||Ax||_\infty \tag{8}$$

and by defining $\alpha_l = \gamma_l + p(l)$, the optimization problem can be written as:

$$\min_x \Big(||Ax||_\infty + (\xi_{sp} + \xi_{mr})||x||_0 \Big) \tag{9}$$

$$\text{s.t} \sum_{l=1}^{|\mathcal{N}_i(t)|} x_l \leq \frac{c_i}{v_t^i}, \quad \sum_{l=1}^{|\mathcal{N}_i(t)|+1} x_l = 1, \;\; 0 \leq x_l \leq 1 \;\; \forall\, l \tag{10}$$

However, as it is shown in our experiments in the next section both, ξ_{sp} and ξ_{mr} are very close to 0 and for the rest of the paper we will assume that both of them functions are constants and not related to the number of the videos. It is also well know that l_0 norm minimization is an NP-complete problem and usually approximation algorithms are used [9]. In order to solve the problem we are rewriting it in an epigraph form:

$$\min_{x,\tau} \quad \tau \tag{11}$$

$$\text{s.t:} \quad 0 \leq Ax \leq \tau \tag{12}$$

$$\sum_{l=1}^{|\mathcal{N}_i(t)|} x_l \leq \frac{c_i}{v_t^i}, \quad \sum_{l=1}^{|\mathcal{N}_i(t)|+1} x_l = 1, \;\; 0 \leq x_l \leq 1 \;\; \forall\, l \tag{13}$$

The lagrangian function is then:

$$\begin{aligned} J(x, \tau, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \mu) = \tau \;\; &+ \sum_{l=1}^{|\mathcal{N}_i(t)|+1} \lambda_1^l(\alpha_l x_l - \tau) \\ &+ \sum_{l=1}^{|\mathcal{N}_i(t)|} \lambda_2^l(x_l - \frac{c_i}{v_t^i}) \;\; + \;\; \mu\Big(\sum_{l=1}^{|\mathcal{N}_i(t)|+1} x_l - 1 \Big) \\ &+ \sum_{l=1}^{|\mathcal{N}_i(t)|+1} \lambda_3^l(x_l - 1) \;\; - \sum_{l=1}^{|\mathcal{N}_i(t)|+1} \lambda_4^l(x_l) \end{aligned} \tag{14}$$

The KKT conditions, the primal and dual feasibility constraints and the complementaty slackness constraints are:

$$\frac{\partial J}{\partial \tau} = 0 \Leftrightarrow 1 - \sum_{l=1}^{|N_i(t)|+1} \lambda_1^l = 0;$$

$$\frac{\partial J}{\partial x_l} = 0 \Leftrightarrow \lambda_1^l \alpha_l + \lambda_2^l + \lambda_3^l - \lambda_4^l + \mu = 0;$$

$$\sum_{l=1}^{|N_i(t)|+1} x_l = 1; \qquad Ax \leq \tau;$$

$$\lambda_1^l(\alpha_l x_l - \tau) = 0; \qquad \lambda_2^l(x_l - \frac{c_i}{v_t^i}) = 0;$$

$$\lambda_3^l(x_l - 1) = 0; \qquad \lambda_4^l x_l = 0;$$

$$\lambda_1^l, \lambda_2^l, \lambda_3^l, \lambda_4^l, \mu \geq 0; \qquad 0 \leq x_l \leq 1.$$

There are two general cases in the solution of the optimization problem. In the one case exists a λ_2^l or a λ_3^l that is not zero. In this case there exists a $x_l : x_l = \min(1, \frac{c_i}{v_t^i})$. If $l \neq |\mathcal{N}_i(t)| + 1$ then the video is split in two parts and $x_l\%$ of it is offloaded to device l while the rest is compressed locally. Else, in the case of $l = |\mathcal{N}_i(t)| + 1$ the video is compressed locally. However, in the general case $\lambda_2^l = \lambda_3^l = 0$ the solution is a classic water filling case that depends on α_l's of x_l that are not zero [10].

IV. Experiments

We used the FFmpeg [11] compression package as the underlying software in our Android app to implement the compression functionality. To conduct our experiments, we used one Xiaomi MI3 deviceand a Samsung Galaxy S2 device.We measured the compression time, the time it takes to split a video into small chunks, the time it takes to transmit video chunks of different size from one device to another, and the time it takes to merge several chunks into one full video. We also measured the energy consumed by the Samsung phone for each of the above tasks. To measure the energy, we used the widely adopted Monsoon Power Monitor[2], which samples the power of the device at a frequency of 5 KHz. However, due to physical limitations—the Xiaomi MI3 does not have a removable battery, which is a requirement for the Monsoon Power Monitor—we only measured the energy on the Samsung Galaxy S2 device. We selected MPEG4 compression, which is based on H.264 code [12], since it is a standard and can achieve high compression rate. In our experiments, we used a framerate equal to 24, a video screen size of 320×240, an aspect ratio of 4:3 and, a constant rate factor of 20. We recorded a list of videos and we present their duration, initial size and size after the compression in Table I.

Figure 3a shows that the size of the video decreases by around 89% after the compression. We should notice that this rate does not depend on the device, since it is only related to the settings of the encoder. On the other hand, the time needed to compress the file is related to the processing power of the device, as can be seen in Figure 3b. Figure 3c shows the energy consumed by the Samsung Galaxy S2 device for compressing each of four videos with length 10s, 50s, 100s, and 500s.

[2]https://www.msoon.com/LabEquipment/PowerMonitor/

Duration	Initial (MB)	Compressed (MB)
00:00:01	1.79	0.21
00:00:02	2.61	0.43
00:00:05	5.54	0.96
00:00:10	10.58	0.86
00:00:20	20.16	2.23
00:00:30	29.94	2.24
00:00:50	49.35	4.89
00:01:40	97.44	13.42
00:08:20	483.26	53.42

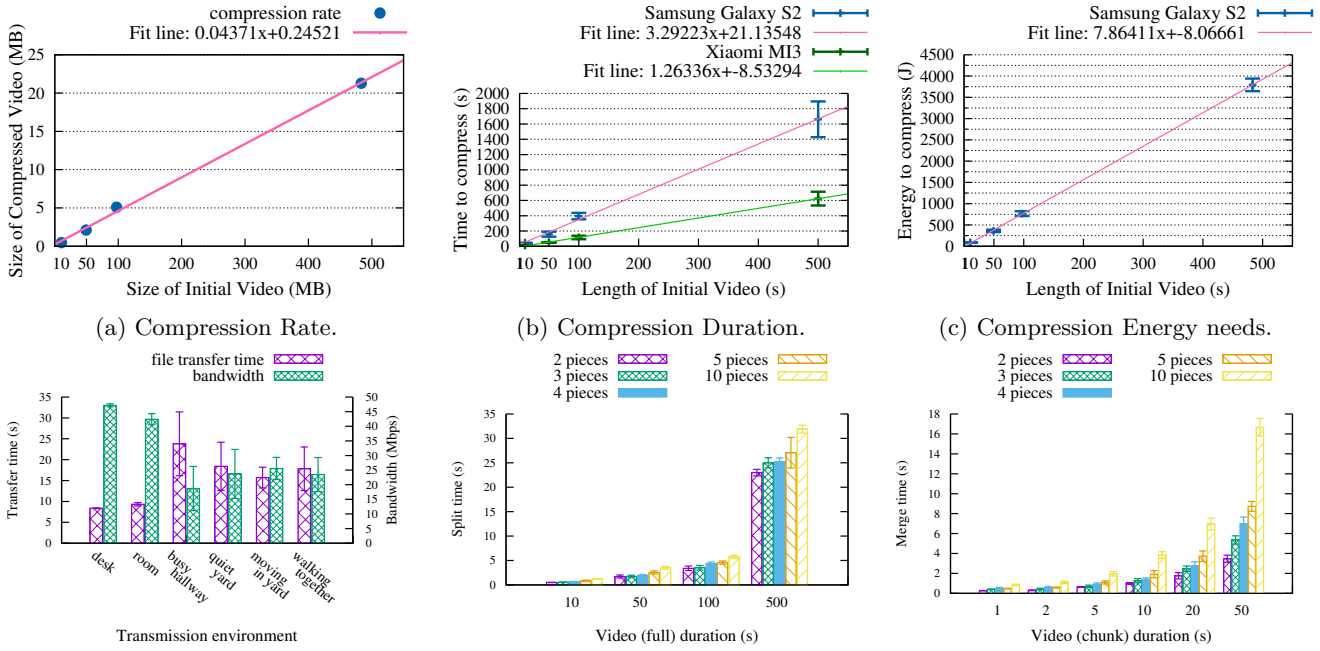
Table I: Video files used in the experiments.

We also measured the time it takes to transmit videos of different size between devices using Wi-Fi Direct in a peer–to–peer mode under several environmental conditions and mobility patterns. In more details, we examined six different cases with two devices: *(i)* standing close to each other in the same desk, *(ii)* standing in the same room in a 3 meters distance, *(iii)* standing in 10 meters distance in a busy hallway with many obstacles and people, *(iv)* standing in an open yard area in a 25 meters distance, *(v)* moving in an open yard area with a distance between 2 and 25 meters, and *(vi)* walking nearby, in a 1 meter distance in a busy area. The experiments were repeated multiple times and the results were averaged. Figure 3d shows the average time required to transfer a video of 50 seconds, 49.35 MB, (left y-axis) and the bandwidth of the Wi-Fi Direct connection (right y-axis) in each scenario. The errorbars represent the standard deviation.

We also measured the energy spent by the Wi-Fi Direct interface when transmitting and receiving, which is: $4.07674772 \cdot 10^{-8} Joules/bit$ for transmission and $5.963156535 \cdot 10^{-8} Joules/bit$ for reception. To obtain these results, we exchanged a video of 483.26 MB between a Samsung Galaxy S2 and a Xiamo MI3 device in the setting of the first case. The experiments were performed ten times and the results were averaged.

Then, we measured the time needed to split the aforementioned videos into 2, 3, 4, 5, and 10 chunks, presented in Figure 3e. Due to the lack of space and to the fact that the results were similar in both devices, we only show the results obtained with the Xiaomi MI3. We also performed the inverse experiment, measuring the time needed to merge small video chunks of 1, 2, 5, 10, 20, and 50 seconds into a full video, shown in Figure 3f. The merging process is performed by merging the video chunk with a copy of itself 2, 3, 4, 5, and 10 times.

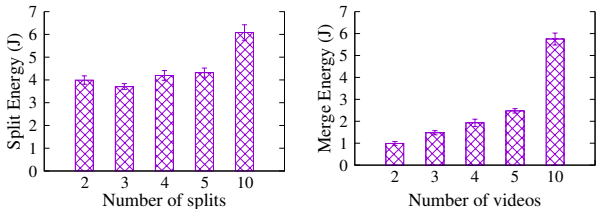
In Figure 4a we show the energy spent on the Samsung Galaxy S2 to split a 50 seconds long video (49.35 MB) into 2, 3, 4, 5, and 10 smaller chunks, while in Figure 4b we show the energy spent on the device to merge multiple copies of 5 seconds long video chunks. Using the results



(a) Compression Rate. (b) Compression Duration. (c) Compression Energy needs.

(d) Transmit using Wi-Fi Direct. (e) Split. (f) Merge.

Figure 3: Basic characteristics of video compression in terms of process duration, energy needs and data transmission.



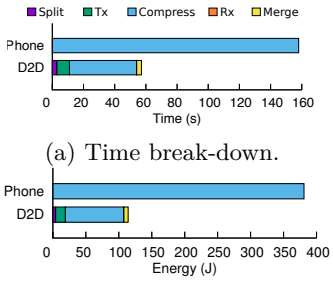
(a) Split. (b) Merge.

Figure 4: Energy required on the Samsung Galaxy S2 to: a) split a video of 50 seconds into smaller videos, and b) merge videos of 5 seconds into a longer video.

of the experiments, one can easily see that by distributing the compression process on more than one device, the processing time and the energy consumption will be reduced.

As an example, let's consider a 50 seconds (49.35 MB) video that need to be compressed by a Samsung Galaxy S2 device. According to our experiments, it takes around 158 seconds and around 362 Joules of energy on average if the compression is fully performed on the device (see Figure 3b and Figure 3c). Let's now assume that the device splits the video into 5 equal chunks of 10 seconds, each chunk is around 10 MB. This process takes 3 seconds (see Figure 3e) and consumes around 4 Joules (see Figure 4a).

Considering the scenario where devices are on the same desk, the data rate is around 45 Mbps (see Figure 3d). Sending four of the five chunks to the nearby devices—the last chunk will be compressed by the device itself—it takes less than 8 seconds (4 chunks × 10 MB sent at 45 Mbps), and consumes less than 14 Joules of energy (sending 40 MB with $\sim 4.07 \cdot 10^{-8} J/b$). Then, the parallel compression time is guided by the slowest device, which in our case is the Samsung Galaxy S2, and it takes around 43 seconds to compress the 10 seconds video chunk (see Figure 3b). As for the energy consumed on the considered device, from Figure 3c we can see that the energy spent to compress the 10 seconds chunk is around 84 Joules. The time and energy needed to receive the four compressed 10 seconds video chunks (each of size 0.86 MB now, see Table I) from the remote devices are 0.15 seconds and 0.43 Joules, respectively (4 chunks × 0.86 MB received at 45 Mbps consuming $\sim 5.96 \cdot 10^{-8} J/b$). Finally, the time and energy spent to merge the five compressed chunks in a unique video are around 3 seconds (see Figure 3f) and 6 Joules, (see Figure 4b) respectively. To summarize, the total time and energy spent during the parallel compression process are around 57.15 seconds and 108.43 Joules, respectively. Compared to the case when the compression was performed fully on one device (158 seconds and 362 Joules), we have an improvement of almost 3 times in terms of compression time and of more than 3 times in terms in energy, as shown in Figure 5.



(a) Time break-down.

(b) Energy break-down.

Figure 5: Time and energy of compression when performed fully on the phone and when distributed using D2D.

V. Related Work

Authors of [13], motivated by the same factors presented in this work, i.e. high computation resources and energy needs for video compression on mobile devices, propose a cloud assisted video compression solution. Their approach offloads the video to a cloud server and executes remotely the most intensive part of the video compression algorithms, that is the motion estimation component. Furthermore, authors of [14] deal with the applicability of H.264 video encoder on mobile devices and propose a modularization of the encoder. They also propose three different offloading schemes for their proposal, but they don't implement any real system and evaluate their performance only using simulations. Further, authors of [15] analyze the energy costs of transferring files between devices using 3G, Wi-Fi, and Bluetooth. Moreover, they also analyze the energy consumption of file compression and decompression and examine the benefits that these techniques can bring to the file transfer in terms of latency and energy. However, their study is only limited to file transfer and doesn't consider computation offloading. Authors of [16] use a face detection algorithm in video streaming to prove that a device with limited resources is not able to deal with such intensive operation, and argue that a possible solution could be to offload the heavy computation to the cloud.

Serendipity [4] is one the most recent and prominent works related to Device–to-Device computation offloading. The authors propose and implement a framework that uses nearby devices for distributed task computation. They show, through real experiments, that the P2P network of collaborative devices reduces the execution time of the considered tasks. In this work, we advance on the model presented in Serendipity and show that the distributed video compression on a Device–to–Device scenario not only is possible but is also beneficent to the interested mobile device. More recently, authors of [17] propose a more generic architecture for mobile D2D offloading and build a first prototype for Android devices.

VI. Conclusion and Future Work

In this paper we proposed the problem of distributed video compression for mobile devices in a D2D offloading scenario. First, we formulated the problem mathematically and solved it as an optimization problem. Then, we presented a real Android system that we built based on the proposed architecture. We showed that the high bandwidth offered by Wi-Fi Direct makes it possible for long videos to be split in smaller chunks, while the chunks were efficiently sent to nearby devices for compression.

As future work, we plan to create a more stable version of the Android application in order to perform large-scale experiments, involving volunteering students and researchers. We also plan on extending the energetic measurements to a broader range of mobile devices, so to cover a more heterogeneous network.

VII. Acknowledgements

This research has been supported, in part, by General Research Fund 26211515 from the Research Grants Council of Hong Kong, Innovation and Technology Fund ITS/369/14FP from the Hong Kong Innovation and Technology Commission, and the European Commission under the Horizon 2020 Program through the RAPID project (H2020-ICT-644312).

References

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[2] 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 *Proceedings of IEEE INFOCOM*, 2012.

[3] "Wifi alliance: The worldwide network of companies that brings you wi-fi," http://www.wi-fi.org/, accessed: 2015-10-23.

[4] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of MobiHoc '12*, 2012, pp. 145–154.

[5] J. M. Cabero, V. Molina, I. Urteaga, F. Liberal, and J. L. Martin, "CRAWDAD data set tecnalia/humanet (v. 2012-06-12)," http://crawdad.org/tecnalia/humanet/, Jun. 2012.

[6] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, "CRAWDAD data set cambridge/haggle (v. 2006-01-31)," Jan. 2006.

[7] J. K. Laurila, D. Gatica-Perez, I. Aad, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, M. Miettinen *et al.*, "The mobile data challenge: Big data for mobile computing research," in *Pervasive Computing*, no. EPFL-CONF-192489, 2012.

[8] S. Seuken, D. C. Parkes, E. Horvitz, K. Jain, M. Czerwinski, and D. Tan, "Market user interface design," in *Proceedings of the 13th ACM Conference on Electronic Commerce*, ser. EC '12. New York, NY, USA: ACM, 2012, pp. 898–915.

[9] M. Hyder and K. Mahata, "An approximate l0 norm minimization algorithm for compressed sensing," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, April 2009, pp. 3365–3368.

[10] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[11] "Ffmpeg: A complete, cross-platform solution to record, convert and stream audio and video." https://www.ffmpeg.org/, accessed: 2015-10-16.

[12] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, July 2003.

[13] Y. Zhao, L. Zhang, X. Ma, J. Liu, and H. Jiang, "Came: Cloud-assisted motion estimation for mobile video compression and transmission," in *Proc. of the 22Nd International Workshop on NOSSDAV*. New York, NY, USA: ACM, 2012.

[14] X. Zhao, P. Tao, S. Yang, and F. Kong, "Computation offloading for h.264 video encoder on mobile devices," in *Computational Engineering in Systems Applications, IMACS Multiconference on*, Oct 2006, pp. 1426–1430.

[15] R. Palit, A. Singh, and K. Naik, "Enhancing the capability and energy efficiency of smartphones using wpan," in *Personal Indoor and Mobile Radio Communications (PIMRC), 2011 IEEE 22nd International Symposium on*, Sept 2011, pp. 1020–1025.

[16] D. Kovachev, "Framework for computation offloading in mobile cloud computing," *IJIMAI*, vol. 1, no. 7, pp. 6–15, 2012.

[17] A. Mtibaa, K. Harras, K. Habak, M. Ammar, and E. Zegura, "Towards mobile opportunistic computing," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, June 2015, pp. 1111–1114.