CAME: Cloud-Assisted Motion Estimation for Mobile Video Compression and Transmission

Yuan Zhao
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
yza173@sfu.ca

Lei Zhang
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
lza70@sfu.ca

Xiaoqiang Ma
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
xma10@sfu.ca

Jiangchuan Liu
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
jcliu@sfu.ca

Hongbo Jiang
Department of EIE
Huazhong University of
Science and Technology
Wuhan, Hubei, China
hongbojiang@hust.edu.cn

ABSTRACT

Video streaming has become one of the most popular networked applications and, with the increased bandwidth and computation power of mobile devices, anywhere and anytime streaming has become a reality. Unfortunately, it remains a challenging task to compress high-quality video in real-time in such devices given the excessive computation and energy demands of compression. On the other hand, transmitting the raw video is simply unaffordable from both energy and bandwidth perspective.

In this paper, we propose CAME, a novel cloud-assisted video compression method for mobile devices. CAME leverages the abundant cloud server resources for motion estimation, which is known to be the most computation-intensive step in video compression, accounting for over 90% of the computation time. With CAME, a mobile device selects and uploads only the key information of each picture frame to cloud servers for mesh-based motion estimation, eliminating most of the local computation operations. We develop smart algorithms to identify the key mesh nodes, resulting in minimum distortion and data volume for uploading. Our simulation results demonstrate that CAME saves almost 30% energy for video compression and transmission.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information Systems**]: Video; C.2.1 [**Network Architecture and Design**]: Wireless communication

General Terms

Experimentation

NOSSDAV'12, June 7–8, 2012, Toronto, Ontario, Canada.

Keywords

Mesh-based Motion Estimation, Mobile Video Compression, Cloud-assisted Motion Estimation

1. INTRODUCTION

Video streaming has become one of the most popular networked applications, and it contributes a dominant fraction of internet traffic. Along with the advances in 3G and wireless network, mobile devices become an important end device for Internet video applications. Online statistics [14] shows that YouTube mobile gets over 400M views a day, representing 13% of the overall YouTube daily views. Unfortunately, video compression on mobile devices remains a challenging task due to limited energy. The user has to copy the video to a personal computer, then compress and upload the video. This however is not convenient and discourages people to share mobile videos on Internet.

Another trend of Internet is that Cloud Computing is booming recent years. Cloud Computing provides an illusion of infinite computing resources which include bandwidth, computation and storage. Major cloud providers also provide High Performance Computing (HPC), which suits multimedia processing well.

If mobile devices can leverage Cloud Computing resources to perform video compression, the computation cost on the mobile device itself can be reduced dramatically. However, transferring large video file to cloud server introduces huge energy consumption, which contradicts the benefit. The question then becomes how to leverage cloud server computation resources without transferring the whole video file.

A typical video compression consists of: motion estimation, transformation, quantization, and entropy coding. Among all these steps, the motion estimation is the most computation intensive and time consuming, which accounts for 90% computation time of the whole compression process [4]. Hence it is worthwhile to transfer even part of the motion estimation computation to cloud servers.

This paper proposes Cloud-assisted Motion Estimation (CAME), a novel method to smartly leverage cloud's computation resources for motion estimation. We focus on mesh-based motion estimation, which is known to be highly effec-

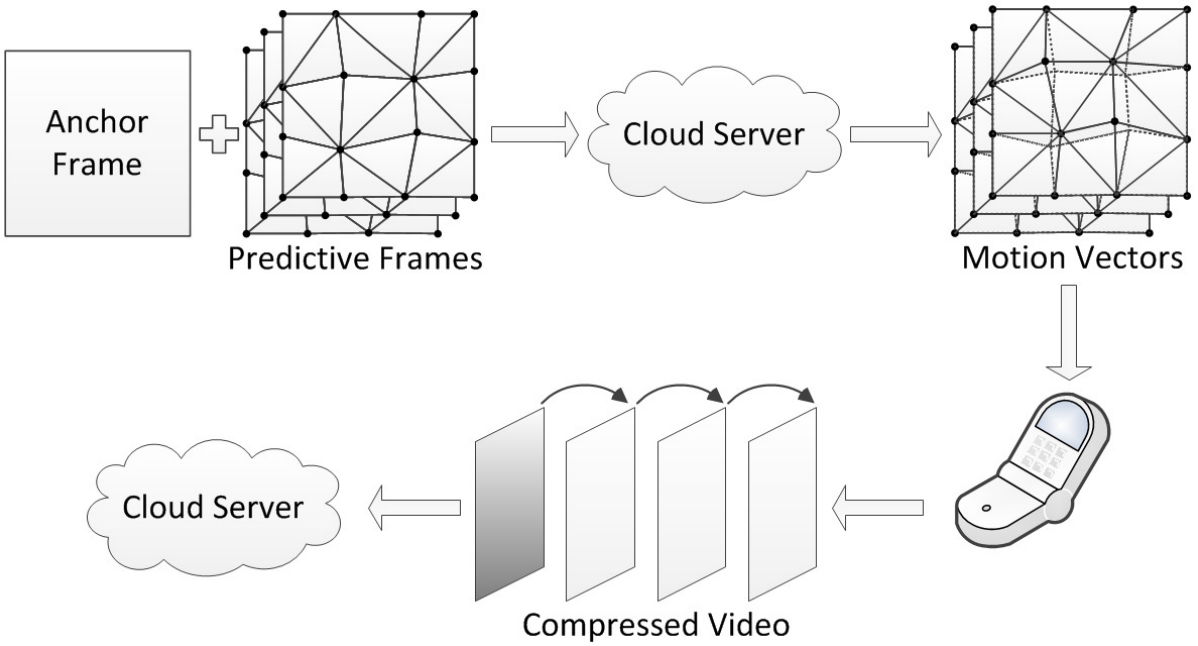

Figure 1: CAME illustration

tive. Our method uploads anchor frames and mesh nodes to cloud server for calculating mesh nodes Motion Vectors (MV). The motion vectors are pushed back to mobile devices for motion estimation of sub blocks and the rest video compression steps. By carefully choosing mesh structure, video compression energy consumption can be largely reduced on mobile devices.

Our simulation result suggests, the proposed method can save up to 30% energy for video compression and transmission compared to All-on-Mobile method which performs the complete video compression on mobile devices.

The rest of this paper is organized as following: We first review the related work in section 2. Then in Section 3, we present CAME system architecture. In Section 4, we introduce the implementation details of CAME. Section 5 evaluates the proposed method using simulated testing system. Finally in Section 6, we draw conclusions and propose the future research work.

2. RELATED WORK

2.1 Cloud-based Video Streaming

With the elastic and on-demand nature of resource provisioning, cloud computing has become a promising platform for diverse applications, many of which are video related [7, 9, 12].

To reduce bandwidth reservation cost and to guarantee the streaming performance, a predictive cloud bandwidth auto-scaling system is proposed in [10] for VoD providers. The predictable anti-correlation between the demands of video channels is exploited in the system for statistical multiplexing and for hedging the risk of under-provision. Built on a peer-to-peer storage cloud, *Novasky* [8] provides on-demanding streaming of cinematic-quality videos over a high-bandwidth network with two novel mechanisms: a coding-aware peer storage and replacement strategy and an adaptive server push strategy with video popularity-redundancy awareness. A cloud-based video proxy system is presented in [5] transcoding the original video in real rime using a scalable codec based on H.264/SVC (Scalable Video Coding), which is aimed at streaming videos of various qualities.

Motivated by these previous studies, our CAME seeks to exploring the resources from cloud as well. Our focus in this paper, however, is to smartly utilize such resource for motion estimation, the most computation-intensive task in video compression, so as to realize realtime highly quality video encoding and streaming from mobile devices.

2.2 Mobile Video Compression

Since mobile devices typically depend on a limited energy supply and video compression is computation-intensive, many existing works have focused on reducing the computational cost for mobile devices [1, 3, 11]. The low complexity video compression system suggested in [6] abandons the ME/MC paradigm and codes the difference between successive frames, making the process significantly less time consuming. A two-step algorithm is introduced in [1], which is further improved in [3] to reduce the computation and memory accesses with variable block size motion estimation. A mobile video communication system is developed [11], in which the transmitter uses a Wyner-Ziv (WZ) encoder while the receiver uses a traditional decoder. An efficient transcoder should be inserted in the network to convert the video stream.

Our work differs from them in that we explores mesh-based motion estimation [13], which is known to be cost-effective and yet has to be examined in the mobile communication context. We demonstrate that, given the small data volume of meshes, they work well with cloud-assisted compression to best balance computation cost, transmission overhead, and compression quality.

3. CAME: CLOUD-ASSISTED MOTION ESTIMATION FOR MOBILE DEVICES

We consider a cloud-assisted mobile video streaming system that consists of mobile devices and cloud servers. A mobile device user captures video in realtime and expects to compress the video and then streaming it to others in realtime as well. Given the high computation overhead of compression and the limited computation power and battery of the device, it is preferable that the compression operation or part of the operation is shifted to the cloud servers. Yet, simply uploading the raw video to the cloud server for compression will consume significant bandwidth and therefore energy for transmission, and is thus not applicable. To this end, our CAME seeks to shift the motion-estimation, the most computation-intensive module in compression to the cloud. Specifically, CAME employs a mesh-based motion estimation, which consists of two parts: mesh node motion estimation and the sub-block motion estimation. As we will show later, the mobile device can upload reference frames and mesh data to the cloud for estimation, which are of much smaller data volume. It then downloads the estimated Motion Vectors (MVs) from the cloud server and completes the remaining video compression steps.

The CAME architecture is illustrated in Figure 1, which includes following four key steps:

1. In the mobile device, the raw video is divided into macro blocks (MBs). For each MB, a reference frame is extracted, together with a mesh for each successive P-frame. The device then uploads the reference frame and meshes to the cloud;

2. The cloud server conducts the mesh motion estimation for the uploaded reference frame and meshes, and pushes the generated mesh MVs back to the CAME client on the mobile device;

3. The mobile devices, upon receiving the MVs for mesh nodes of each P-frame, continues to calculate sub block MVs using block-based motion estimation as well as entry coding;

4. The compressed video is then stored in the device or stream to other devices or servers through the wireless channel.

4. CAME IMPLEMENTATION

While the CAME architecture is intuitive, there are a number of implementation issues to be addressed toward a complete system. In this section, we explain the implementation details of CAME, particulary on mesh node selection strategy for cost minimization.

4.1 Mesh Node Selection

The mesh-based motion estimation is performed by estimating one motion vector (MV) for each mesh node. As shown in Figure 2 and Figure 3, mesh nodes are sampled on the reference frame and MVs are calculated from predictive frames (P-frame).

Generally, two different mesh structures are used: regular triangular or rectangular mesh structures and object-based mesh structure.

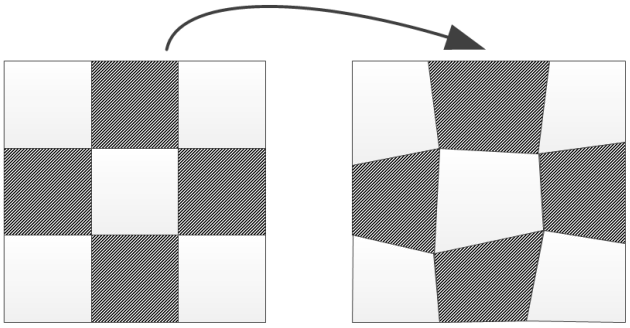

Figure 2: Mesh-based Motion Estimation

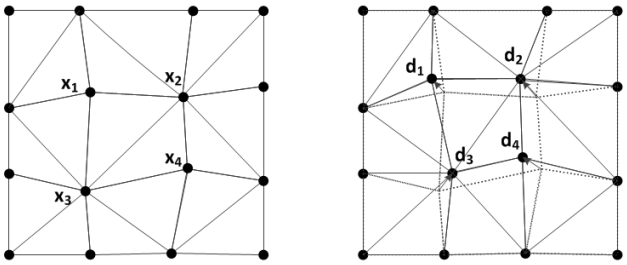

Figure 3: Mesh Nodes in Motion Estimation

Regular Mesh Structures
Triangular and rectangular mesh structures are two commonly used structures in all the mesh structures. Mesh topology is simple and predefined in regular mesh.

Object-based Mesh Structure
The object-based mesh structure is also known as adaptive mesh structure. As in real world videos, usually there are discontinuities at object boundaries, so it would be more accurate to sample mesh nodes along object boundaries. In another word, the mesh selection and sampling should be adapted to the video objects, so that the motion within mesh nodes is smooth. Someone has also proposed algorithms to sample mesh nodes along motion or luminance discontinuities in order to fit the objects boundaries.

Hierarchy Mesh Structures
Hierarchy mesh structure is between regular mesh structure and adaptive mesh structure. It's based on regular mesh, but is adaptive to image objects.

Regular mesh is commonly used since it is simple and predefined. Using regular mesh, both encoder and decoder know the mesh structure, thus there is no overhead to store and send the mesh topology to the decoder. The object-based structure is more accurate because it samples mesh nodes along object boundaries, however it requires extra overhead to analyze objects and mesh topology transmission. Therefore, regular mesh is preferred in CAME method.

Unlike standard mesh-based motion estimation, CAME exploits a smarter algorithm for mesh node selection: CAME applies a reversed mesh node selection and motion estimation algorithm, in which mesh nodes are sampled on P-frames and MVs are calculated from the mesh and the reference frame. Compared to standard mesh-based motion esti-

mation, CAME loses the advantage of tracking the same set of mesh nodes over successive P-frames. However, CAME gains much more benefits by uploading only the reference frame and mesh data of P-frames instead of uploading the whole video frames.

For a single MB, we denote one reference frame as R, one P-frame as P, and the mesh node fraction M_f from P as f. The total transmission cost is

$$C_m = \sum_{i=1}^{m} (C_R + \sum_{i=1}^{n} C_i \cdot f) \quad (1)$$

where m is the number of MB, and n is number of P-frame in a single MB.

4.2 Cloud Server Motion Estimation for Mesh Nodes

The energy of the cloud server is much more economy than mobile devices, therefore we do not take the cloud side energy consumption into consideration. However, total delay is crucial to the CAME system as CAME is a real-time video compression and uploading system. To reduce total delay of CAME system, we leverage cloud parallel computation resources to compute mesh MVs. The natural separation of frames to MBs provided a great isolation for application level parallelism. CAME server utilize a master-slave paradigm to coordinate the mesh motion estimation for a single video. The CAME server delay can be largely reduced thanks to cloud parallelled computation. The cloud server motion estimation is depicted in Algorithm 1.

Algorithm 1 CloudServer-Motion-Estimation

INPUT: $\{R, M_{fi}\}$, anchor frame R and mesh nodes on P
  $MV_s = \phi$
  for each M_{fj} in $\{R, M_{fi}\}$ **do**
    Full search to calculate MV from R and M_{fj}
    $MV_s = MV_s \cup MV$
  end for
OUTPUT: Push motion vector MV_s back to the mobile device

4.3 Sub-Block Motion Estimation

The mobile device downloads estimated MVs for each P-frame mesh, and we denote the downloading transmission cost as C_d. Then the mobile device uses these MVs to calculate the sub block MVs and finally compose a complete MV for P-frame. The cost to compute all MVs for all sub blocks inside a single P-frame is

$$C_f = \sum_{j=1}^{n} \sum_{i=1}^{m} C_i \cdot f \quad (2)$$

where n is number of P-frame in a single MB, m is sub block number inside a single P-frame. This step is a local search which is constrained by the mesh node MV, so the computation is also restricted and relatively small energy consumption is consumed. Next, the mobile device regenerates the block pixels based on the motion estimation result. We denote the motion compensation cost as

$$C_c = \sum_{i=1}^{n} C_{comp-i} \quad (3)$$

where n is the number of P-frames in a single MB. All the other steps are the same as normal video compression process, and we denote their costs as C_o.

4.4 Mobile Device Algorithm

After the whole video compression process is finished, the video is ready for uploading, and only P-frames are uploaded. We denote cost in this step as C_u. The complete algorithm for the mobile device video compression is shown in Algorithm 2.

Algorithm 2 Mobile-Video-Compression

INPUT: Complete raw video V
  Divide V to MacroBlocks(MBs) denoted by $\{MB_i\}$
  for each MB_i in $\{MB_i\}$ **do**
    Calculate $\{R, M_{fi}\}$, anchor frame R and fraction mesh nodes on P
    Upload $\{R, M_{fi}\}$ to server for mesh node motion estimation
  end for
  $MV_{video} = \phi$
  On receiving $\{MV_{mesh}\}$ from server:
  for each MV in $\{MV_{mesh}\}$ **do**
    Local search to calculate sub-block MV_{sub} for current frame
    Interpolate MV_{sub} into MV to get MV_f for frame
    $MV_{video} = MV_{video} \cup MV_f$
    Calculate video compensation for current frame
  end for
  if MV_{video} is complete **then**
    Finish video compression process to get V_{comp}
    Anchor frames are already deducted from V_{comp}
    Upload V_{comp}
  end if

4.5 Total Cost Equation

Finally, the total cost equation is formulated as:

$$C = C_m + C_d + C_f + C_c + C_o + C_u \quad (4)$$

where C_m is the total cost for uploading all MBs, C_d is MV downloading cost, C_f is the total cost for sub block motion estimation, C_c is the motion compensation cost, C_o is all other costs for video compression on the mobile device after motion estimation, C_u is the final step mobile video uploading cost.

5. EVALUATION

In this section, we evaluate implemented CAME method on a simulated system. Mobile video compression and transmission both require energy consumption. We evaluate CAME method using a two-computer simulation system: one acts as cloud server and the other acts as the mobile device. We use simulated system because it is more generic and the result is not specific to a mobile device.

The transmission energy consumption is measured and calculated from compressed video file size and such CAME system intermediate output as mesh motion vectors. We consider a generic energy model in terms of CPU cycle consumption. Video compression CPU cycles can be directly monitored and measured. However, transmission measurement in CPU cycles is not straightforward, as transmission

Table 1: YUV Video Files Used in Testing

Video Name	Size($W \times H$)	Frame Num	YUV	File Size
Foreman	352×288	300	420	43MB
Mother	352×288	300	420	43MB
Flower	352×288	250	420	36.2MB

consumes energy not only on CPU, but also on wireless. Based on the research work done in [2, 15], we assume the energy conversion between CPU cycle and transmission is: one byte of WiFi transmission consumes roughly the same energy as 132.3251 CPU cycles.

As mentioned previously, mesh node density impacts both transmission and computation. We choose different mesh node fractions f in simulation. CAME are evaluated when mesh node is sampled from 16×16, 8×8, and 4×4 blocks respectively. Our simulation shows that 8×8 mode achieves best energy efficiency. Further, to evaluate how CAME performs over existing mobile video compression and transmission methods, we compare CAME performance with All-on-Mobile and Raw-Upload methods. In the All-on-Mobile method, standard H.264 encoding is performed completed on the mobile device and H.264 encoded video file is uploaded. In the Raw-Upload method, uncompressed video is uploaded in raw format.

In the following sub sections, we first describe the video compression testing data. Then we analyze CAME transmission and computation energy respectively. Finally total energy consumption is evaluated by adding up transmission and computation energy consumption.

5.1 Testing Data

We take three YUV video files in the evaluation which are commonly used in video compression testing. The details of the three videos are illustrated in Table 1. The frame size and frame number are not that important, because the total energy consumption and the total energy saving increase proportionally to them. On the other hand, the video content really matters since the estimated MVs depend on specific video content. Mother and Daughter has low spatial detail, while Foreman has medium spatial detail and Flower has high spatial details.

5.2 Transmission Energy Consumption

We measure and calculate the total transmission consumption in total transmission data size, and convert it to equivalence CPU cycle consumption. There are three transmission phases in CAME: Initial reference frame and mesh node data uploading, mesh node motion vectors downloading, and compressed video data uploading.

The detail measure and calculation result in total transmission data size is illustrated in Figure 4. The total transmission energy consumption in CPU cycles is illustrated in Table 2. Mother and Daughter has the similar result with Foreman. Though Flower's original video size is smallest, the AoM and CAME transmission size is largest, because Flower has higher spatial details. It is obvious that AoM method's data transmission cost is the lowest among all three. The mesh-based motion estimation interpolated to H.264/AVC encoding achieves almost 10:1 compression ratio in our testing. This result is just as expected. Compared to

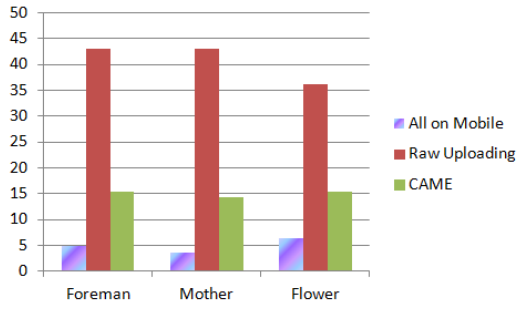

Figure 4: Total Transmission Size (MB)

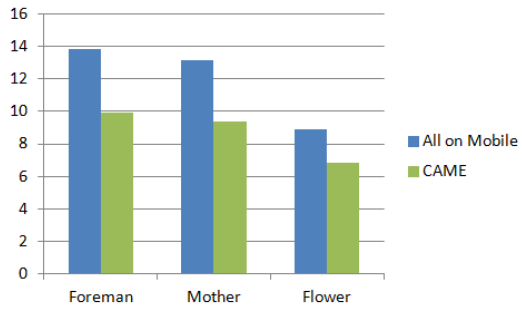

Figure 5: Total Energy Consumption in CPU Cycles (billion cycles)

AoM method, the proposed method introduces more transmission because of the extra data transmission overhead for mesh node uploading and mesh motion vectors downloading. On the other hand, compared to Raw-Upload, the CAME method still saves approximately 60% on total data transmission. This verifies our idea that CAME can leverage cloud server resources for motion estimation and video compression without transferring the whole video data.

5.3 Video Compression Energy Consumption

The measurement of video compression energy consumption in CPU cycles is straightforward. For CAME, the CPU cycles are measured for part of the motion estimation and all the other steps of video compression that performed on the mobile device. For the AoM method, complete video compression is performed solely on the mobile device, the total CPU cycle consumption is measured when video is encoded by the Java H.264 encoder. The measurement result is depicted in detail in Table 3.

In this measurement, we notice that CAME method can save nearly 40% compared to the AoM method, that is exactly what we are expecting. By transferring part of the motion estimation to cloud server, CAME method do save a lot on computation, approximately 50% of the motion esti-

Table 2: Transmission Energy Consumption in CPU Cycles

Video Name	All on Mobile	CAME
Foreman	6.2987×10^8	2.0429×10^9
Mother	4.6843×10^8	1.8814×10^9
Flower	8.4291×10^8	2.0201×10^9

Table 3: Video Compression Energy Consumption in CPU Cycles

Video Name	All on Mobile	CAME
Foreman	1.3225×10^{10}	7.8689×10^{9}
Mother	1.2650×10^{10}	7.5268×10^{9}
Flower	8.0500×10^{9}	4.7898×10^{9}

mation computation can be reduced on mobile devices. The test result also verifies that motion estimation accounts for 90% of total video compression time.

5.4 Total Energy Consumption

To evaluate the total energy consumption and calculate how much energy we can save using the CAME method, we simply add up energy consumption in transmission and video compression in Table 2 and Table 3, and the results are compared in Figure 5.

As shown in Figure 5, CAME achieves up to 30% total energy saving on video compression and transmission compared to the AoM method. This is a significant improvement towards energy saving on mobile video compression and transmission. Though the total transmission size in CAME is larger than the AoM method, CAME saves considerable energy on video compression, particularly for the motion estimation.

Our experimental result suggests that 8×8 mesh selection mode achieve the best performance compared to 4×4 and 16×16 mesh selection modes. This is because in the 4×4 mesh mode, the mesh nodes uploading and mesh motion vectors downloading data size is too large. In the 16×16 mesh mode, as the mesh node density is not enough, though the mesh uploading and mesh motion vectors downloading data size is small, the mobile device still need to perform a large fraction of the motion estimation.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented Cloud-assisted Motion Estimation (CAME), a novel video compression scheme for mesh-based motion estimation. By taking advantage of computational resource of the cloud server, our proposed method significantly reduces the complexity of video compression on mobile devices, which leads to considerable energy saving. Experimental results showed that CAME is highly energy-efficient. One drawback of this work may be the encoding delay introduced by closed loop design of our scheme, which is mainly due to transmission delay between client and server.

As part of our future work, we will consider some optimization problems in our system, especially the coordination of wireless transmission between the cloud and mobile devices, to achieve even lower transmission overhead and higher energy-efficiency.

7. ACKNOWLEDGMENTS

This research is supported by a Canada NSERC Discovery Grant, an NSERC DAS grant, a Nokia University Relation Fund, and an NSFC Major Program of International Cooperation.

8. REFERENCES

[1] A. Bahari, T. Arslan, and A. Erdogan. Low-power h. 264 video compression architectures for mobile communication. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(9):1251–1261, 2009.

[2] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM, 2009.

[3] S. Chatterjee and I. Chakrabarti. Power efficient motion estimation algorithm and architecture based on pixel truncation. *Consumer Electronics, IEEE Transactions on*, 57(4):1782–1790, 2011.

[4] M. Dudon, O. Avaro, and C. Roux. Triangular active mesh for motion estimation. *Signal Processing: Image Communication*, 10(1):21–41, 1997.

[5] Z. Huang, C. Mei, L. Li, and T. Woo. Cloudstream: delivering high-quality streaming videos through a cloud-based svc proxy. In *INFOCOM, 2011 Proceedings IEEE*, pages 201–205. IEEE, 2011.

[6] E. Jackson and R. Peplow. Video compression system for mobile devices. *RN*, 2:2, 2003.

[7] Y. Lai, C. Lai, C. Hu, H. Chao, and Y. Huang. A personalized mobile iptv system with seamless video reconstruction algorithm in cloud networks. *International Journal of Communication Systems*, 2011.

[8] F. Liu, S. Shen, B. Li, B. Li, H. Yin, and S. Li. Novasky: Cinematic-quality vod in a p2p storage cloud. In *INFOCOM, 2011 Proceedings IEEE*, pages 936–944. IEEE, 2011.

[9] D. Miao, W. Zhu, C. Luo, and C. Chen. Resource allocation for cloud-based free viewpoint video rendering for mobile phones. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 1237–1240. ACM, 2011.

[10] D. Niu, H. Xu, B. Li, and S. Zhao. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In *Proc. of IEEE INFOCOM*, volume 12, 2012.

[11] E. Peixoto, R. de Queiroz, and D. Mukherjee. Mobile video communications using a wyner-ziv transcoder. In *Symposium on Electronic Imaging, Visual Communications and Image Processing (SPIE), San Jose, CA, USA*, 2008.

[12] K. Singh and C. Davids. Flash-based audio and video communication in the cloud. *Arxiv preprint arXiv:1107.0011*, 2011.

[13] Y. Wang, J. Ostermann, and Y. Zhang. *Video processing and communications*, volume 1. Prentice Hall, 2002.

[14] YouTube. Youtube statistics. "http://parsec.cs.princeton.edu/". [Online; accessed 05-Jan-2012].

[15] W. Yuan and K. Nahrstedt. Energy-efficient cpu scheduling for multimedia applications. *ACM Transactions on Computer Systems (TOCS)*, 24(3):292–331, 2006.