Demo: Adaptive Video Streaming for Device-to-Device Mobile Platforms

Joongheon Kim, Feiyu Meng, Peiyao Chen, Hilmi E. Egilmez, Dilip Bethanabhotla, Andreas F. Molisch, Michael J. Neely, Giuseppe Caire, and Antonio Ortega
University of Southern California, Los Angeles, CA 90089, USA
{joonghek, feiyumen, peiyaoch, hegilmez, bethanab, molisch, mjneely, caire}@usc.edu, antonio.ortega@sipi.usc.edu

ABSTRACT

This demo abstract describes an initial design of a new adaptive video streaming protocol for device-to-device WiFi-based mobile platforms and its software implementation. For the demonstration, two mobile servers and two mobile users will be deployed verifying that our device-to-device adaptive video streaming implementation works with desirable user experience.

Categories and Subject Descriptors

C.2.2 [**Computer-Communications Networks**]: Network Protocols; C.2.3 [**Computer-Communications Networks**]: Network Operations; C.2.4 [**Computer-Communications Networks**]: Distributed Systems

General Terms

Algorithm, Design, Management

Keywords

Adaptive Video Steaming, Android, Device-to-Device, Mobile Platforms

1. INTRODUCTION

Recently, the USC Communication Sciences Institute has been investigating device-to-device wireless protocols and architectures [1][2][3]. Among the research contributions, the algorithm proposed in [3] introduces joint transmission scheduling and admission control for adaptive video streaming in device-to-device wireless networks or small-cell networks. As presented in [3], there are two types of devices, i.e., *helpers* and *users* where helpers are mobile video clouds, i.e., they contain video files which can be requested by users, and users are mobile smartphones which request video service. In this given network configuration, helpers are responsible for transmission scheduling, i.e., each helper decides which user can download requested video from helpers in each time slot in terms of sum rate maximization where the video contents within helpers consist of a sequence of chunks. When users receive these chunks from helpers via wireless links, they can playback the independently decodable video chunks one-by-one. While the helpers schedule the transmission of chunks, users select the quality mode for the next video chunk to be received. Eventually, this protocol can achieve desired performance in terms of network utility maximization as shown in [3]. In this demo abstract, we describe the WiFi-based software design and implementation of our previous contribution in [3] on top of Android mobile platforms. However, there is one practical limitation which is described as follows. According to the proposed algorithm in [3], multiple helpers select their users which have the maximum value of $Q_u^h(t)C_u^h(t)$ where $Q_u^h(t)$ and $C_u^h(t)$ stand for the queue backlog size from helper h to user u at time t and the channel capacity of the link from helper h to user u at time t, respectively. Now, for the system that has multiple helpers and a single user, the helpers will always select the single user on every time slot. However, it is not allowed in terms of WiFi-based wireless systems because multiple access points (APs) are not allowed in a single WiFi network (note that our software design defines helpers as APs and users as non-AP stations). We design and implement a new feature for WiFi-based mobile platforms named *greedy pull for minimum delay (GPMD) strategy* instead of the transmission scheduling. With this GPMD strategy, each user selects its helper which has the chunk that is required in terms of a *playback order* at each time slot. Then, the selected helper should stream the chunk to the user and the amount of transmitted chunk is determined by the maximum WiFi link date rate. Note that multiple chunks can be transmitted if the channel is good, otherwise, parts of a chunk can be transmitted if the channel suffers. In this demo , we design and implement the software architectures of the proposed GPMD-based adaptive streaming for device-to-device Android mobile platforms.

MobiCom'13, September 30–October 4, Miami, FL, USA.

ACM 978-1-4503-1999-7/13/09.
http://dx.doi.org/10.1145/2500423.2505292.

2. MODELS AND ALGORITHMS

2.1 A Reference Network Model

Suppose that the given network is defined by a bipartite graph $\mathcal{G} = \{\mathcal{U}, \mathcal{H}, \mathcal{E}\}$, where $\mathcal{U}$, $\mathcal{H}$, and $\mathcal{E}$ stand for the sets of users (mobile smartphone users), helpers (mobile video cloud servers), and edges for all pairs (h, u) such that there exists a potential WiFi wireless link from $h \in \mathcal{H}$ and $u \in \mathcal{U}$, respectively. The helper is defined as a device which has a set of videos. When users (i.e., mobile phones) want to watch a specific video, they search neighbor helpers and download the sequence of the chunks of the desired videos. For each chunk, users can download it from different helpers. Therefore, helpers (i) contain video files and (ii) serve user requests via WiFi wireless links. On the other hand, users (i) request videos and (ii) downstream the sequence of the chunks of the requested video.

2.2 An Operational Framework

In terms of video data structure, a video consists of a sequence of chunks. In [3], the unit of service processes of transmission queue is in bits. However, bit-level transmission control is not allowed in current Android API (SDK Version 4.1). To cope with this problem, we define the concept of *sub-chunk* where a single chunk consists of a sequence of sub-chunks. With this concept, we can control the amount of transmitted bits in terms of the number of transmitted sub-chunks depending on the channel conditions. The dynamics of the transmission queue at each helper for each user is modeled as follows where $t \in \{0, 1, \cdots\}$:

$$Q_u^h(t+1) = \max\left\{Q_u^h(t) - \mu_u^h(m,t), 0\right\} + c_u^h(m,t) \quad (1)$$

where $\forall(h,u) \in \mathcal{E}$ and $Q_u^h(0) = 0, \forall h \in \mathcal{H}, \forall u \in \mathcal{U}$. In addition, $c_u^h(m,t)$ and $\mu_u^h(m,t)$ stand for the number of bits of incoming chunk with quality mode m at time t and the number of bits of processed one or more sub-chunk(s) with quality mode m at time t, respectively. Note that if the channel is good, a helper can transmit more sub-chunks in a given time interval. The users can employ three different algorithms: (i) selecting helpers for each chunk placement (i.e., Sec. 2.2.1), (ii) selecting helper which should transmit one or more sub-chunk(s) to the user based on the GPMD strategy (i.e., Sec. 2.2.2), and (iii) selecting quality mode for each chunk (i.e., Sec. 2.2.3). Therefore, users can handle entire protocol behaviors and helpers just transmit chunks to requested users with desired quality mode.

2.2.1 Chunk Placement

For each time slot, each user determines which helper should place/stream the next chunk. When each time slot starts, each user sends packets requesting the next video chunk to all neighbor helpers, and each helper replies back the current queue backlog size to the user. Then, the user select the helper which has the smallest queue backlog size and the user sends one packet to the helper for letting it know that the helper should send the next chunk in its queue. If there are multiple helpers which has the smallest queue backlog sizes, the user performs a random selection.

2.2.2 Greedy Pull for Minimum Delay

For each time slot, each user selects its helper which has to transmit sub-chunks to the user. In our previous work [3], transmission scheduling is defined and implemented at helpers and the objective of the function is sum rate maximization. Then, one user can be selected by multiple helpers due to the fact that helpers are operating in a fully-distributed manner. Moreover, if only one user exists in the system, then all helpers select the single user on every time slot. Unfortunately, this kind of system cannot be supported by WiFi-based mobile platforms, because having multiple APs in a single WiFi network is not allowed. Thus, we design an alternative algorithm at user side, named *greedy pull for minimum delay (GPMD)*. Suppose that a user is receiving a sequence of chunks from helpers, and the sub-chunks of the desired video exist within the local storage of the user. Then, the user checks the playback order of its own list of sub-chunks. When the user figures out that a sub-chunk is not yet received from helpers and urgent in terms of playback order, then user requests that sub-chunk from the helper. If the channel condition is good enough, the user is able to download not only this sub-chunk but also the sub-chunks within the FIFO queue of the currently serving helper. The amount of sub-chunks to be transmitted from helpers to users is determined by the channel state of the WiFi link.

2.2.3 Quality Mode Selection

For each user, the quality mode $m_u(t)$ is computed by

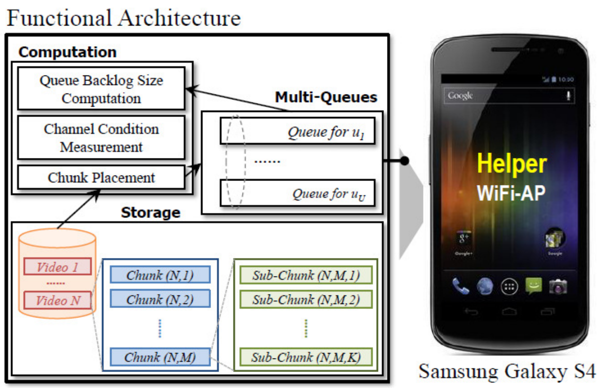
$$m_u(t) = \arg \min_{m \in \{H,L\}} \left\{Q_u^h(t) \cdot kB_v(m,t) - VD_v(m,t)\right\} \quad (2)$$

as explained in [3] where H and L stand for high-quality and low-quality, respectively. In addition, $D_v(m,t)$ and $kB_v(m,t)$ stand for the video quality measurement index and the number of bits for video v with quality mode m at time t. Last, $V > 0$ means a control parameter which affects an utility-delay tradeoff [3].

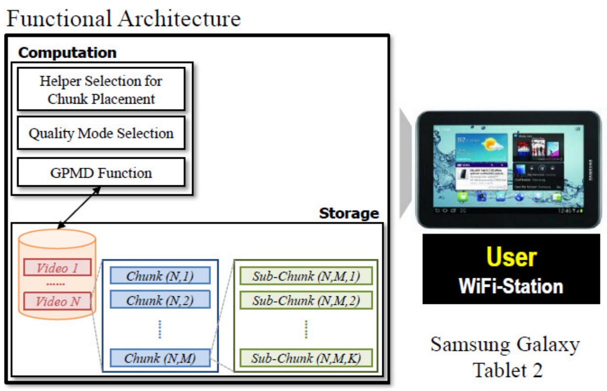
3. IMPLEMENTATION

3.1 A Protocol Operation Overview

For the first time slot, the implemented protocol starts when a user broadcasts a request of desired video to neighbor helpers. Then, the neighbor helpers reply back packets to the requested user with the information of their queue backlog sizes. In this initial stage, there are no sub-chunks in the queues yet, i.e., all neighbor helpers will reply back to the user with zero. Then, the user selects one helper which has the shortest queue backlog size. According to the fact that the queue backlog sizes of all neighbor helpers are zero at this moment, the user performs a random selection where the selected helper is denoted as h'. Now, the user transmits one packet to inform h' about: (i) h' is selected as the serving helper, (ii) desired chunk of the requested video (note that the user needs the first chunk at this initial stage), (iii) maximum link rate between current user and h', and (iv) desired quality mode for the requested chunk computed by (2). Then, h' sends the sub-chunks of the first chunk into the queue with desired quality. Note that there is no video sub-chunk transmission from helper to user in this initial time slot, i.e., just the sub-chunks are pushed into the queue. For the next time slot, the user requests the next chunk (i.e., the second chunk)from neighbor helpers. Then, the neighbor helpers reply back packets to the user with their queue backlog sizes. After that, the user selects the helper which has the shortest queue backlog size and follows the same procedure discussed before (chunk placement procedure). In terms of video chunk transmission, if the user have not received any chunks , i.e., the user requests the first sub-chunk of the first chunk along with computed quality mode located at h'. On the helper side, h' transmits the sub-chunks with desired quality mode (H or L)within the given time slot interval. If the channel condition is good, the h' may transmit more sub-chunks, but in the worst case, it can transmit only one sub-chunk. If the *helper*, h', cannot



(a) Helper Software Architecture



(b) User Software Architecture

Figure 1: Device Software Architectures

send the all sub-chunks of the first chunk, then it can be transmitted later when the helper is scheduled. In the following time slots, the previous operations are repeated until helpers transmit the last sub-chunk of the last chunk.

3.2 Helper Implementation

The software for helper side is implemented on top of Samsung Galaxy smartphones with Android platforms. The helpers in this system are set to WiFi-APs, i.e., they work as WiFi APs. Therefore, the users can also periodically measure achievable link rates or RSSI measures from the helpers, because the corresponding functions are allowed in WiFi AP-centric systems, such as `getRssi()` defined by the classes in the Android API's `android.net.wifi` package. For multi-user support, the helpers can create multiple TCP sockets for the multiple users which want to download sub-chunks from the helpers. The roles of helpers are as follows: (i) returning the queue backlog sizes when they receive chunk requests from users, (ii) placing sub-chunks when they are selected by users for specific chunk service, and (iii) transmitting a number of sub-chunks depending on the channel conditions. The corresponding software architecture is illustrated in Fig. 1(a).

3.3 User Implementation

The software for user side is implemented on top of Samsung Galaxy tablet with Android platforms. The roles of users are (i) determining which helpers should be selected for chunk placements, (ii) selecting helpers for the GPMD strategy in terms of sub-chunks downloading, and (iii) selecting quality modes for each chunk. The corresponding software architecture is illustrated in Fig. 1(b).

4. DEMONSTRATION

4.1 Settings

For the demonstration, two helpers and two users will be used (Samsung Galaxy S4, ID: GI9250). To do this setting, four AC-power sources are required (110V) and one table for deploying devices is also required. In addition, one easel for presenting the poster which describes our system and algorithm is also needed. Additional wireless resources such as AP are not required. In terms of space, one easel for a poster and one table for four phones are required.

4.2 Video

Throughout this demo, we use standard test sequences ”Bridge” and ”Highway” both of which have 2000 frames with CIF (352×288) resolution. Each video sequence is divided into 10 chunks and then encoded using ffmpeg encoder [4] at 25 fps so that each chunk corresponds to 8 seconds. In order to generate two different quality levels per video, we encode each video at rates 250 kbps and 50 kbps corresponding to high quality and low quality streams, respectively. Our demonstration video clip is available in [5].

5. CONCLUDING REMARKS

This demo abstract presents a practical implementation of an adaptive video streaming protocol for device-to-device mobile software platforms. In the given network, users can dynamically select helpers and video quality modes (two quality levels) for adaptive video streaming. In addition, we proposed the GPMD strategy to deal with the implementation issues or limitations in our previous work [3].

Acknowledgments

This work is supported in part by the Intel VAWN projects.

6. REFERENCES

[1] N. Golrezaei, A.F. Molisch, A.G. Dimakis, and G. Caire. Device-to-device collaboration: a new architecture for wireless video distribution. *IEEE Communications Magazine*, 2012.

[2] N. Golrezaei, K. Shanmugam, A.G. Dimakis, A.F. Molisch, and G. Caire. FemtoCaching: wireless video content delivery through distributed caching helpers, In *INFOCOM'12: Proceedings of IEEE International Conference on Computer Communications*, 2012.

[3] D. Bethanabhotla, G. Caire, and M.J. Neely. Joint transmission scheduling and congestion control for adaptive streaming in wireless device-to-device networks. In *Asilomar'12: Proceedings of Asilomar Conf. Signals, Systems, and Computers*, 2012.

[4] `http://ffmpeg.org/`

[5] `https://www.youtube.com/watch?v=uRqwnCWDCsw`