FLEXIBLE MEDIA TRANSPORT FRAMEWORK FOR ANDROID

Sándor Molnár, Péter Megyesi, Szilárd Solymos, Zsolt Krämer, Zoltán Móczár

High Speed Networks Lab., Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics, Budapest, Hungary
{molnar, megyesi, solymos, kramer, moczar}@tmit.bme.hu

ABSTRACT

Adaptive video playing methods are mainly based on solutions involving the usage of HTTP together with TCP in the transport layer. However, TCP no longer seems to be the only solution. Other transport protocols (e.g., UDP) with different transport features (e.g., new error correction techniques) are also possible candidates. In this demo we demonstrate a flexible media delivery framework which is applicable to test both state-of-the-art media transfer methods and new possible future media transport mechanisms. We show both the joint effect of multiple buffering and adaptation algorithms as they are competing with each other and also the impact of Forward Error Correction schemes on the video quality. Measurements data can be logged in the Android application and important QoE parameters (e.g., playout state, resolution, buffer size) can be analyzed.

1. THE DEMONSTRATED FRAMEWORK

Adaptive video streaming has become the killer application in today's Internet where video playing is mainly based on the following solutions: (i) Adobe Flash Video, (ii) Microsoft Smooth Streaming, and (iii) Apple HTTP Live Streaming. Since all of these solutions are based on similar foundations, the MPEG-DASH standard was created in 2012 in order to unify the methods for adaptive video playing. In DASH the media segments are downloaded via HTTP protocol thus it inherits the usage of TCP in the transport layer. However, alternative solutions can be used in the future and we built a framework where new transport mechanisms can be investigated.

We demonstrate our newly developed *media transport framework* in which we are able to add different transport layer features and test their effect on DASH media playback over Android platform. Fig. 1 presents the architecture of our framework. In the DASH standard, a Media Presentation Description (*.mpd*) file identifies a video by containing every necessary information such as available representations (e.g., codec, resolution and corresponding bitrate) and the URL access of the media segments. After parsing the *.mpd* file, the client can download the video and audio segments via HTTP protocol. The DASH standard does not specify any client logic (e.g., buffering and adaptation algorithms), thus the players can perform significantly different from each other. We created a custom video player application on Android platform that is based on the ExoPlayer library. The media app is able to report various QoE parameters such as the bitrate of the played

This research work was supported by Traffic Lab., Ericsson Hungary.

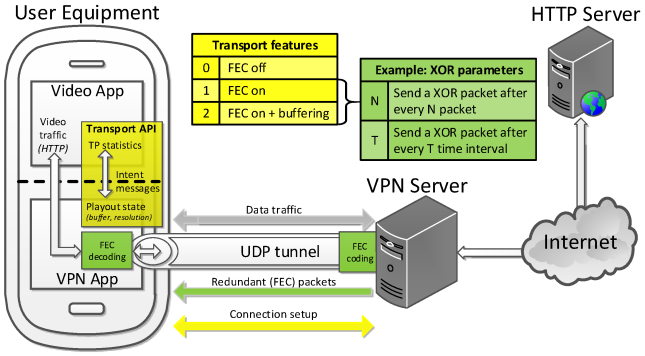

Fig. 1: The demonstrated framework.

video/audio segment and the current buffer size. Moreover, the application does not reach the network directly but via a custom VPN application. In the VPN app we are able to access to the network stream and add any given Layer-4 feature without the need of modifying the Android kernel.

We made a bidirectional API between the video and VPN apps to exchange info between the application and the transport layer. The video app sends current payout state (bitrates, buffer size, etc.) to the VPN app which now starts the change of the transport features. Also, the VPN sends metrics (available bandwidth, packet loss rate, etc.) to the video player app which initiates a resolution change based on the received data.

Fig. 2 shows the demonstrated testbed, which consists of (i) two Android tablets installed with the aforementioned apps, (ii) a laptop receiving concurrent traffic, (iii) a WiFi access point to connect the test devices and (iv) another laptop with installed VPN and HTTP servers and different DASH media content. We show scenarios of media data protected by Forward Error Correction (FEC) packets based on XOR coding resulting in an aggressive behavior and larger bandwidth share on the common link.

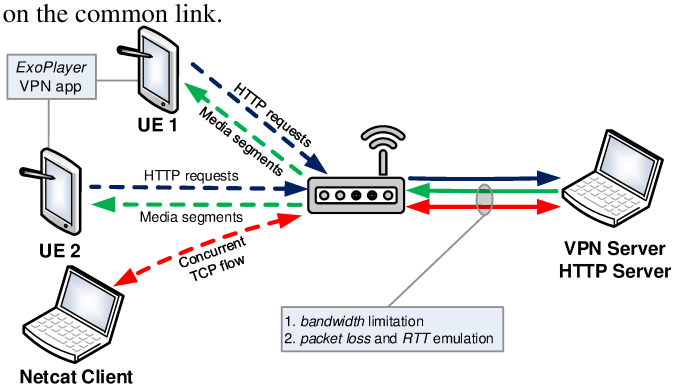

Fig. 2: The demonstrated testbed.