Leveraging Transitions for the Upload of User-generated Mobile Video

Stefan Wilk[1], Roger Zimmermann[2], Wolfgang Effelsberg[1]
[1]TU Darmstadt, Germany
[2]National University of Singapore, Singapore
{stefan.wilk, effelsberg}@cs.tu-darmstadt.de,
rogerz@comp.nus.edu.sg

ABSTRACT

A recent trend in user-generated content production is the broadcasting of live video streams from mobile devices. A set of upload protocols have been proposed supporting the live transmission of user-generated video. Their performance depends on the environmental conditions, e.g., the mobility of users, the network conditions or the popularity of the streams. Thus, we propose a novel mobile broadcasting framework, which exchanges different uploading protocols during the runtime of the application. Our goal is to use the protocol performing best under given application requirements and environmental conditions. If the requirements or the conditions change, the system dynamically assesses whether the the protocol currently used is still the most appropriate one for streaming. In case a superior protocol is available, the system *transitions* to the new protocol. By leveraging such transitions for video upload protocols, we achieve a superior overall performance under changing network conditions in comparison to a single upload protocol.

Keywords

DASH-U, User-generated Video, Mobile Phones, Upload

CCS Concepts

•**Computer systems organization** → *Client-server architectures;*

1. INTRODUCTION

Today's Mobile Video Broadcasting Services (MBS) have in common that the media is usually streamed over unreliable, resource-capped networks, e.g., cellular networks (LTE or UMTS). Usually, those networks can only offer a limited uplink transmission speed, which impacts the bitrate of streamed videos from the mobile devices. In addition, the mobility of users and changing network conditions require a constant adaptation of the uploading scheme. Our assumption is that no single uploading protocol exists, which performs best under all network conditions and changing application requirements.

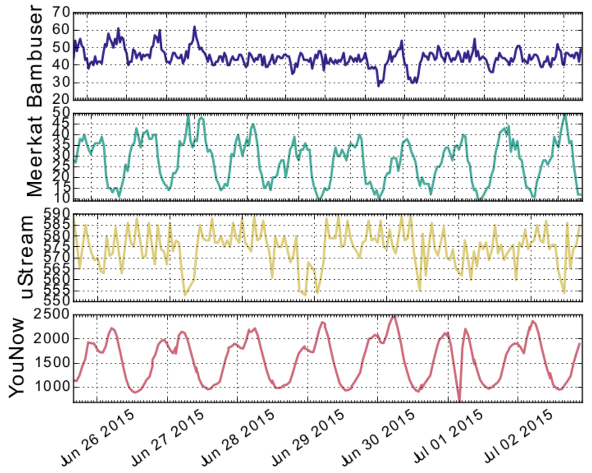
Our proposed system leverages well-known as well as latest research proposals for uploading video and allows to seamlessly switch between the protocols. A switch between protocols may be triggered, when network conditions and/or application requirements change. Such a switch is called a *transition* and allows to select the best scheme at any given time. Enabling transitions has a major advantage in comparison to the design of a hybrid streaming solution: it allows to integrate arbitrary new upload protocols in future. Thus, a transition-enabled system can adapt to even unknown environmental conditions without a complete redevelopment. The main contribution of this work is the design of a transition-capable MBS. This MBS allows researchers and engineers to retrieve the advantages (benefits) and disadvantages (costs) of using transitions in video upload applications. Furthermore, we introduce methods for decoupling the upload protocol from the application and the lower layers.

MOVID'16, May 13 2016, Klagenfurt, Austria
© 2016 ACM. ISBN 978-1-4503-4357-2/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2910018.2910658

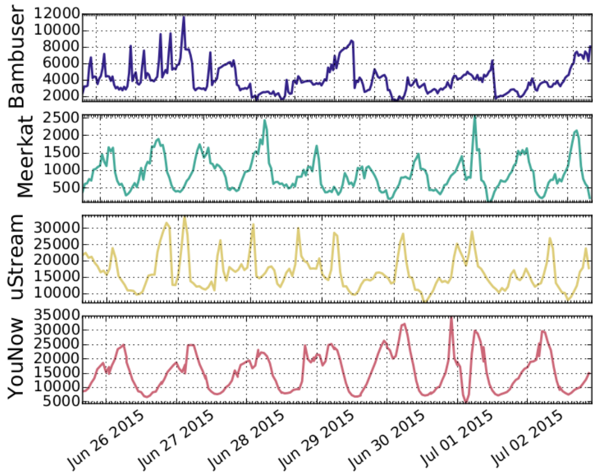
2. BACKGROUND AND RELATED WORK

We now present a brief overview on the usage of an MBS. Figure 1 shows the access patterns of users to the platforms YouNow, Bambuser, uStream and Meerkat between the 26th of June 2015 and the 2nd of July 2015. The MBSs show different popularity patterns. YouNow seems to attract most viewers and recorders. Platforms such as Meerkat seem to have the lowest success in attracting a high number of concurrent users. In general, we see that each platform has a range of 40 to 2000 concurrent live streams.

The average quota of viewers per broadcaster is 11 for YouNow, 33 for Meerkat, around 49 for uStream and 116 for Bambuser. Usually, broadcasters only compete for viewers with live streams and not with pre-recorded video clips. Our data indicates that the reduced competition allows broadcasters to more easily attract viewers for their live streams. As Stohr et al. [13] show for YouNow a cumulative distribution function on the popularity of videos is usually far less skewed compared to a video sharing site like YouTube. This finding is backed in other publications for live video streaming platforms such as Vine [15], YouTube Live [5] or Twitch [14]. In addition, Stohr et al. [13] investigate the encoding parameters of video and show that the com-



(a) Concurrent Broadcasters



(b) Concurrent Viewers

Figure 1: Overview of available stream and viewer numbers on Bambuser, Meerkat, uStream and YouNow. (a) The y-axis depicts the number of concurrent broadcasters. (b) The y-axis represents the number of concurrent viewers.

mon bitrates, frame rates and resolutions of video streams, generated on mobile devices, are rather low. For encoding YouNow uses the currently widely deployed video encoding H.264/AVC. The findings of Stohr et al. indicate that encoding parameters are set based on the network interface and device capabilities. In general, the bitrates are rather low, reaching up to 1 Mbit/s and frame rates are below 15 frames per second.

Ito et al. [3] propose the Spatial and Temporal Omni-Directional Video Distribution and Collective System (SODiCS), which is able to collect video in a pull-based manner from mobile devices. SODiCS is an exotic example for a MBS which is designed for the break-down of the infrastructure in disaster cases. El Essaili et al. [2] investigate the process of uploading video when the scheduling of an LTE network can be controlled. They show a centralized optimal decision for the uplink transmission when a client uploads video in a quality-aware manner. In contrast to El Essaili, our focus lies on the development of an efficient video uploading scheme in the application layer. Seo et al. [10] discuss how the MPEG DASH standard can be used for the upload of media. For this purpose, they leverage the HTTP POST method to continuously upload video segments. The proposed system achieves transcoding and transmission of a 480p video with a start-up delay of about the duration of one segment under good WiFi conditions. Seo et al.'s HTTP-based uploading protocol is similar to the streaming protocol Meerkat uses. We use the implementation of Seo et. al. as one uploading protocol candidate. The DAVVI system [4] is designed to generate video segments and upload them immediately afterwards in order to generate a low-delay video streaming experience. Johansen et al. report on dynamically adapting the bitrate of a video during the streaming session.

Recently, Richerzhagen et al. [7] propose a collaborative uploading scheme for MBS. To ensure that video streams are received in-time and at a high quality, mobile recorders cooperate by sharing their upload capacities.

We propose to leverage *transitions* between uploading protocols in a MBS to cope with varying environmental conditions and application needs. We understand a transition as a complete replacement of a protocol during the runtime of an application.We assume that a set of protocols exist, offering similar functionality, e.g., the uploading of video streams, but showing different performance characteristics under different environmental conditions.

3. SYSTEM DESIGN

To allow transitions between upload mechanisms on the application layer without influencing the lower layers our MBS needs to run in a decoupled way (see Figure 2). A media recording application on a mobile device accesses a media recorder API provided by the operating system to record video from the device's camera and microphone. The media streams are stored in a recording buffer representing an abstraction of the transmission layer on which transitions can take place. This transmission layer is decoupled both from the recording application and the network. A transition-capable runtime in the transmission layer allows to switch between protocols during upload. The proposed MBS aims at efficiently uploading live video streams to a central server. We focus on leveraging existing research proposals on the application layer for uploading protocols to show that the system can be easily extended with new protocols. Protocols investigated in this work are the Real-time Messaging Protocol (RTMP) [1], a modified version of Dynamic Adaptive Streaming over HTTP (DASH) [12] allowing media upload, Seo et al.'s HTTP POST-based DASH [10] and our custom protocol for Live Video Upload (LiViU). These protocols are explained in the following subsection.

Please note that the transitions are planned and executed on the local device.

3.1 Video Upload Protocols

RTMP is a TCP-based and thus stateful media streaming protocol, which assumes that the media content is pushed from a source to its receivers. As an upload protocol RTMP establishes a reliable and, ideally, low delay end-to-end con-

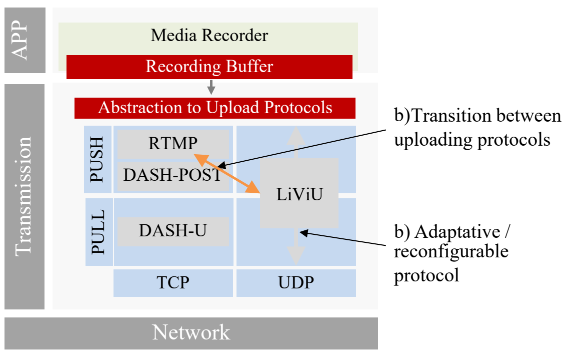

APP
Media Recorder
Recording Buffer
Transmission
Abstraction to Upload Protocols
PUSH
RTMP
DASH-POST
LiViU
PULL
DASH-U
TCP
UDP
Network
b)Transition between uploading protocols
b) Adaptative / reconfigurable protocol

Figure 2: Overview on our transition-enabled MBS.

nection between a mobile device recording a digital video and a server. A session is established using a three-way handshake procedure, mainly consisting of the exchange of authentication information. Thus, RTMP is capable to not only transfer one media stream, but deliver multiple synchronized audio and video tracks in parallel. The initialized connection allows the transmission of variable-sized media segments with a compact header structure. A drawback of the initiating procedure is the rather long delay until a connection is active. Especially, an MBS requires that the join procedure does not delay the initial video segments to be transferred. An advantage of the protocol is the message structure, when a connection is established. Header overhead is minimized as long as a connection is established.

Our second upload protocol is DASH-U. In recent years, HTTP-based video streaming approaches gained significant interest. The MPEG DASH [12] standard defines the network communication using HTTP (TCP) as well as the description of the video in a manifest – the Media Presentation Description (MPD). This description is required as DASH supports adaptive video streaming, which allows to dynamically select an appropriate bitrate for each network connection. This pull-based video streaming scheme is used for the design of **DASH-U**. Here, the video server regularly requests video segments of a client. Each client transmits segments of a video only if they are requested by the server. Such a request/response communication pattern for a continuous stream is very costly in respect of the overhead of the communication. On the other hand, the method is well suited for scenarios, where the streaming server requires only a few video segments from each client.

The **DASH-POST** protocol is our third upload protocol. It is similar to the one introduced by Seo et al. [10]. As the name implies, communication in DASH is based on the transmission of video segments via HTTP-POST requests. Whereas in *DASH-U* the server requests individual segments using HTTP GET requests, DASH-POST solely pushes the video segments to the server. The overall delay until a segment is available on the streaming server should thus be lower. An advantage of using HTTP instead of a dedicated streaming protocol such as RTMP is that no session or state has to be established before a new segment is transmitted.

We have also designed our custom **hybrid upload protocol** called Live Video Upload (LiViU). LiViU is based on an quality-adaptive video streaming relying on the unreliable UDP transport protocol. LiViU is designed for rapid connection establishment and immediate streaming. LiViU differs from the other protocols by a more complex scheduling component for distributing video segments. Whereas, the push-based streaming transmits video to a server as soon as a segment is available, the pull-based method requires more sophisticated control operations, e.g., to allow a server to retrieve the latest video segment before the next one is sent. It is furthermore possible, that requests are made for multiple video segments, e.g., for the upcoming five seconds of video. LiViU can easily adapt between these scheduling modes. In addition, LiViU requires a lightweight session management in order to inform the network nodes, when a mobile device joins or leaves the network. The advantage of this approach is the comparably low overhead as well as the reduced latency due to the usage of UDP. It is a hybrid video uploading approach, which can reconfigure itself automatically in order to push video segments or allow them to be pulled by the server. This reconfiguration or adaptation allows to handle some environmental changes, but not all. In contrast, transitions allow the complete replacement of a protocol by another one.

The implementation of the protocols does not include security features, e.g., for encrypted video streaming.

3.2 Upload Protocol Transitions

The rationale behind the transitioning between two upload protocols is to ensure a consistently high streaming quality measured in the average uploaded video bitrate. The possibly transmittable video bitrate can vary between the protocols as the overhead of protocols is different, the underlying transport protocols (TCP or UDP) work differently well under given network conditions, and the scheduling type (push/pull-based) influences the latency. The dynamic replacement of an upload protocol requires a runtime environment that supports transitions.

We build an abstraction of our transmission layer towards the recording application and the lower layer network protocols, i.e., using TCP and UDP sockets. The recording buffer is the interface to the recording application. This allows a unified access for all uploading protocols to the video data. The buffer works as follows: When a recorder application generates a new video segment, e.g., by recording a video frame and the associated audio samples, this segment is transferred to the buffer, which informs the uploading protocol to transfer the segment. In most cases this segment can be directly passed without being cached in the buffer. Only when the uploading bandwidth is below the bitrate of the recorded video data is cached in the recording buffer. The buffer is then read sequentially in FIFO order. The adaptive environment additionally offers the functionalities for monitoring metrics that determine when to switch from one upload protocol to another. The *monitoring* gathers performance data on the current upload protocol.

We focus on the following metrics: a) the overhead of a protocol, b) the goodput of the protocol, c) the initial time until a stream is established and d) the current latency for each recorded video segment.

The *overhead* (OH) of an upload protocol is calculated in $[bits]$ and is affected by the number and size of the control messages of a protocol as well as headers of video messages. The average overhead is used in our system which uses the unit $[\frac{bits}{second}]$ as is calculated by: $\bar{OH} = \frac{OH}{t_{session}}$

The *goodput* (GP) is measured in terms of the effective throughput of video data in $[\frac{bits}{second}]$. This excludes protocol overhead or coordination messages by other system modules.

The *join time* (T_J) measures the time from the first recorded video frame takes until it reaches the server. It is measured in milliseconds. The rationale behind introducing the join time is that MBS users want to quickly share their videos to an audience. The join time is the lower bound for the delay of a video being recorded until it is accessible by viewers. Note, that the transitions, which implies the initialization of an upload protocol, are planned so that the join time occurs only once in a streaming session.

Thus, besides the initial join time the different protocols may cope differently with changing upload bandwidths. This is addressed by the *current latency* (T_L) which measures the latency between the timestamps of recording a video segment on the mobile device and the timestamp at the receiving server: $T_L = t_{receive} - t_{record}$.

The central idea of switching between uploading protocols is to achieve a superior performance of the system by optimizing of either *overhead*, *the goodput* of the upload protocol, *join time* or *latency*. Transition rules either minimize the recording latency ($min(T_L)$), the join time ($min(T_J)$), the overhead of the protocol ($min(OH)$) or maximize the goodput ($max(GP)$). The current implementation of our framework optimizes – either minimizing or maximizing – only one metric at a given point in time. During runtime the application decides which metric shall be optimized (see Section 4.1). In future work we plan to optimize a combination of these parameters, depending on the type of video application.

A switch from one uploading protocol to another is achieved by leveraging the recording buffer. It is made instantly available to the new upload protocol by setting according listeners. For the previous protocol, it seems that no new video segment has been recorded and it can be turned off. To determine when to switch from one protocol to another, we regularly check, e.g., every $t_T = 5s$ which protocols performs best for a given metric. After each check the currently best uploading protocol is chosen. Thus, a transition may occur every t_T seconds.

4. EVALUATION

Our evaluation is driven by assessing the potential of transitions between different video uploading protocols. The research question is, if a higher overall efficiency can be achieved by transitioning between different video upload protocols in comparison to a single protocol.

The evaluation investigates this question based on a simulative evaluation with traces from the MBS YouNow. We decided for a simulation setup to assess the system under realistic conditions with hundreds of streaming users, which is not possible in a real-world experiment. For evaluating transitions in our uploading module and between the respective uploading protocols, we conducted simulative runs using the Simonstrator platform [6] and NS-3 communication models [8]. We simulated the proposed MBS for 24 hours on the basis of a real traces from the platforms YouNow recorded on the 06/27/2015 (see Figure 1). The trace consists of up to 1000 concurrent broadcasters, which we assign to different regions with a simulated space of 200mx200m. We assume that one region consists of up to 200 concurrent recorders. Joining users are randomly assigned to a region. The mobility of recorders is modeled according to movement traces taken from the videos from the events 'NAF160312', 'NAF230312' and 'RAF100812'. The videos are accessible from the JIKU Mobile Video dataset [9]. For each video, recorded by a single user, we have extracted the corresponding movement model.

Table 1: Parameters used in the simulative evaluation of the MBS.

Parallel recorders:	up to 1000 (trace-based)
Event space:	200m x 200m
Networks:	UMTS & LTE based on ns-3 models
Latency:	100-300 ms
Bandwidth:	LTE: 50 Mbit/s UL
Movement model:	JIKU Mobile Video DS [9] NAF160312, NAF230312, RAF100812
Video representation:	500, 750 and 1000 kbit/s segment length: 1s

Table 1 shows the simulation setup. We assume that a parallel transcoding of up to three video representations in the bitrates 500, 750 and 1000 kbit/s is possible. We focus on the mobile video broadcaster side and neglect to model the video viewer side.

The minimum length of a video segment which can be played independently is 1 second. This segment length has also been used in the DASH-based video upload protocols [10].

All simulative experiments are repeated ten times with varying simulation seeds for the used random number generator. Depicted figures, which include confidence intervals, indicate a confidence of 95%.

4.1 Scenarios

We introduce a concurrent streaming scenario in which all recorders stream their video according to the trace described. In this case the upload capacity is scarce, all recorders at the same event share the upload capacity.

In a second scenario, the MBS is evaluated using a *video composition* use case in which different incoming video streams are quality assessed and can then be composed in a quality-aware manner. To simulate this scenario, we have implemented the video composition approach by Shrestha et al. [11] which regularly switches between video streams of high quality. The application assumes that the video composition can be achieved in or near real-time.

In this scenario, we assume a composition based on the principles defined by Shrestha et al. [11]: a video quality which can vary over time for a single uploading device, and despite lower qualities the resulting media stream should be diverse. As a result, we imply a simplified composition engine in which the server selects, from multiple incoming streams, the highest quality (Q_{max}) video stream $v_{t,Q_{max}}$ at time t. After t_{div} seconds, the media stream is switched to $v_{t+t_{div},Q_{max}}$ with the restriction that $v_{t+t_{div},Q_{max}} \neq v_{t,Q_{max}}$. This implies that the video stream composed at a time t, should not be selected again at a time $t + t_{div}$, where t_{div} is the time between two composition decisions. t_{div} is set to five seconds. If $v_{t+t_{div},Q_{max}} = v_{t,Q_{max}}$, the video stream with the second highest video quality is selected. In our simulation, the quality of the generated media streams changes regularly (after t_{div}) based on a randomly generated Mean Opinion Score (MOS) between 1 (lowest quality) and 5 (highest quality). A composition service provider may define different optimization goals depending on the quality of the media streams. We have implemented three optimization goals: (1) uploading nodes which generate a high quality video stream (MOS ≥ 3.5) optimize towards a high goodput, (2) mid-quality videos (MOS ≥ 2) are uploaded in order to minimize the overhead and (3) low quality videos (MOS < 2) are not transmitted at all.

4.2 Results: Concurrent Upload Scenario

As mentioned, we assess the potential flexibility in the upload protocols in terms of setting the adaptation goal towards minimizing the *join time*, *latency*, *overhead traffic* or maximizing the *goodput*. The protocols perform very different in respect of each metric. This illustrates that depending on the application requirements, different protocols perform differently well (see Figure 3). For a single optimization goal the best performing protocol can be determined, but no single protocols exists which performs best under all conditions. In respect of the initial join time the RTMP streaming and DASH-U protocols, due to their multi-step join procedure, are the slowest protocols. RTMP uses a three-way handshake which requires multiple messages to be transmitted until the first video segment is sent. DASH-U requires the creation and delivery of a manifest file (MPD) to the server and the selection of an appropriate bitrate until the streaming begins. The quickest joining procedure is achieved by DASH-P, as it immediately starts uploading video using HTTP POST requests; it does not negotiate the streaming session. Interestingly, this is even faster than the LiViU joining procedure, even though the HTTP underlying TCP should require significantly more time in comparison to LiViU's UDP sockets. Regarding the protocol overhead, both HTTP-based approaches create by far the highest overhead. The request/response pattern of DASH-U is very verbose, thus, it wastes data traffic and performs worst in respect of the generated overhead. Once a session is established RTMP and LiViU are the most efficient protocols.

Under challenged network conditions, the average goodput of the protocols is essential. DASH-based approaches are less efficient due to the HTTP overhead including its verbose headers.

DASH-U seems to be inflexible and non-beneficial for any metric. Thus, in this scenario the adaptive case – illustrated in Figure 3 – did not select this protocol. We will elaborate a bit more on why this protocol is still beneficial in Section 4.3.

We can define optimization goals in respect of specific metrics such as latency or join time. We define a strategy that starts with (1) a low join delay when the streaming begins. In the next step, when a reliable streaming session is established, (2) the MBS aims for a high goodput in unchallenged conditions. In situations, when the bitrate of the stream needs to be decreased as the upload speed falls below the video bitrate (3) the system tries to minimize the overhead and adapts to (2) when enough bandwidth is available again. When two upload protocols achieve a similar goodput (4) the system optimizes towards the latency. This scenario is depicted in Figure 3. The different protocols behave differently depending on the goals.

Our adaptive approach achieves comparable performance to the best single protocol. In comparison to RTMP transitions between different upload protocols generates 3.1% more overhead. In comparison to the superior protocol for a low join time (DASH-P), our adaptive approach saves approximately 9.47% of overhead. The overhead of our transition-capable uploading scheme is rather low in relation to the average video bitrate. 0.89% of the average traffic represent protocol overhead. Similarly, the adaptation to a goodput optimizing protocol is slightly worse than the best protocol.

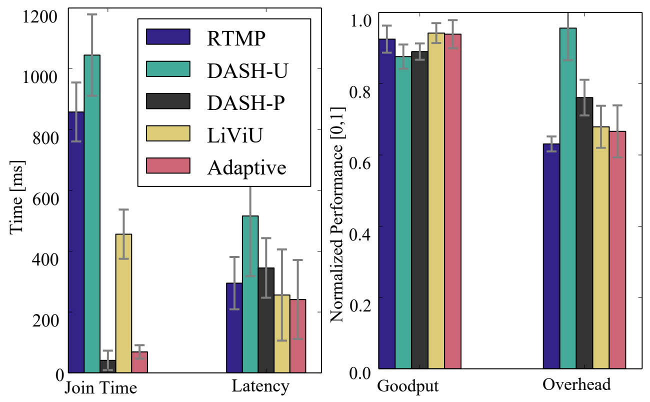


Figure 3: Benefit and costs of our adaptation of uploading protocol environment. In the left part the achieved join time and delay is depicted whereas in the right part the normalized goodput and the normalized overhead of the protocols are shown.

One observation in this scenario, where transitions are mainly induced by an internal switch to a new optimization goal, is that the per-node transition frequency is rather low with in average around 0.017 adaptations (equals one adaptation every 58 seconds). The transition itself takes some time. The faster our system can adapt, the better. The average adaptation time is related to the new protocol after the transition and it mainly consists of the join time. Thus, whereas a switch to a DASH-P is possible with a negligible delay, the remaining upload protocols need up to 1.1 seconds to establish a streaming state.

4.3 Results: Video Composition Scenario

Besides upload capacity, another reason for transitioning between protocol could result from the streaming provider's interest in prioritizing the media streams differently. A common use case is the video composition (see Section 4.1) in which only one outgoing media stream is composed from different incoming media streams at an event.

A first observation in this scenario is, that a protocol such as DASH-U can be very helpful. Using DASH-U, the composition server can easily select the video stream segments from the devices in a very flexible manner. Whereas, for the first scenario push-based protocols should be preferred in respect of all metrics, the second scenario demonstrates that pulling video (DASH-U) can be beneficial. It offers a significantly improved flexibility as the composition server can easily switch between video streams without the coordination overhead needed for push-based upload protocols.

During this evaluation the average transition rate increased to 0.167, meaning that every 5.92 seconds an adaptation was executed. In the majority of cases a transition was invoked by a composition decision. The protocol over-

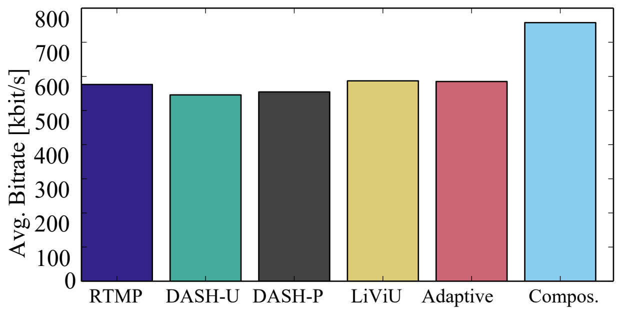


Figure 4: Comparison of the achieved average bitrates of the different uploading protocols including the adaptive approach and the video composition application.

head, due to the reporting of quality values as well as the assignment of composition decisions, increased slightly to 1.24%. The average quality increases of the video streams was around a bitrate of 757.5 kbit/s, significantly higher than the average quality in the non-adaptive scenario with 584 kbit/s and the single protocols (see Figure 4). The composition case using transitions outperforms the usage of a single upload protocol.This is achieved as low-quality recordings were never requested at all by the server.

5. CONCLUSIONS

This work has introduced an improved MBS, which aims at efficiently delivering video from a mobile devices, such as smartphones, to central video streaming servers. We improve existing live upload services (MBS) by leveraging at runtime transitions between mobile upload protocols at runtime. The transition-enabled MBS helps to compensate for varying network conditions by replacing upload protocols with different performance characteristics. We have developed a transition-enabled uploading protocol layer with protocols from related work such as RTMP, DASH-U and DASH-POST and our own custom protocol LiViU. Yet, in many cases the integration of upload protocols into one stack may not be desired by the designers. Thus, our investigation shows the individual strengths and weaknesses of the upload protocols and allows system designers to develop new hybrid solutions. We show that for different metrics, the uploading protocols perform differently. By using transitions between the protocols, an MBS can ensure to use always the best approach for given application requirements and network conditions.

In future work, we plan to investigate the concept of transitions for MBS in more detail by introducing new applications and additional upload protocols. Furthermore, we want to complement our simulative evaluation by a prototypical test. And we plan to investigate new approaches to optimally configure the four upload metrics in an application-dependent way.

Acknowledgements

This work has been co-funded by the German Research Foundation (DFG) as part of the projects C03 within the Collaborative Research Center (CRC) 1053 – MAKI as well as by the project LiViU funded by the German Federal Ministry of Education and Research with grant no. 01IS12054.

6. REFERENCES

[1] Adobe Inc. Real-Time Messaging Protocol (RTMP) 1.0. Technical report, 2009.

[2] A. El Essaili, Z. Wang, E. Steinbach, and L. Zhou. QoE-Based Cross-Layer Optimization for Uplink Video Transmission. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 12(1):1–22, 2015.

[3] K. Ito, G. Hirakawa, and Y. Shibata. Omnidirectional Video and Sensor Data Collection and Distribution System on Challenged Communication Environment. In *IEEE Int. Conference on Advanced Information Networking and Applications*, 2014.

[4] D. Johansen, T. Johansen, T. Aarflot, J. Hurley, A. Kvalnes, C. Gurrin, S. Zav, B. Olstad, E. Aaberg, T. Endestab, H. Riiser, C. Griwodz, and P. Halvorsen. DAVVI: a Prototype for the Next Generation Multimedia Entertainment Platform. In *ACM Int. Conference on Multimedia*, 2009.

[5] K. Pires and G. Simon. YouTube live and Twitch. In *ACM Multimedia Systems Conference*, 2015.

[6] B. Richerzhagen, D. Stingl, J. Rückert, and R. Steinmetz. Simonstrator: Simulation and Prototyping Platform for Distributed Mobile Applications. In *EAI Int. Conference on Simulation Tools and Techniques*, 2015.

[7] B. Richerzhagen, J. Wulfheide, H. Koeppl, A. Mauthe, K. Nahrstedt, and R. Steinmetz. Enabling Crowdsourced Live Event Coverage with Adaptive Collaborative Upload Strategies. In *IEEE Int. Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2016.

[8] G. Riley and T. Henderson. The NS-3 Network Simulator. In *Springer Modeling and Tools for Network Simulation*, 2010.

[9] M. Saini, S. Venkatagiri, W. Ooi, and M. Chan. The Jiku Mobile Video Dataset. In *ACM Multimedia Systems Conference*, 2013.

[10] B. Seo, W. Cui, and R. Zimmermann. An Experimental Study of Video Uploading from Mobile Devices with HTTP Streaming. In *ACM Multimedia Systems Conference*, 2012.

[11] P. Shrestha, P. de With, H. Weda, M. Barbieri, and E. Aarts. Automatic Mashup Generation from Multiple-camera Concert Recordings. In *Int. Conference on Multimedia*, 2010.

[12] T. Stockhammer. Dynamic Adaptive Streaming over HTTP: Standards and Design Principles. In *ACM Conference on Multimedia Systems*, 2011.

[13] D. Stohr, T. Li, S. Wilk, S. S., and W. Effelsberg. An Analysis of the YouNow Live Streaming Platform. In *IEEE Workshop on Network Measurements*, 2015.

[14] C. Zhang and J. Liu. On Crowdsourced Interactive Live Streaming. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2015.

[15] L. Zhang, F. Wang, and J. Liu. Understand Instant Video Clip Sharing on Mobile Platforms. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2014.