Cost-Efficient Framework for Mobile Video Streaming using Multi-Path TCP

Yeon-sup Lim
Department of Convergence Security Engineering
Sungshin Women's University, Seoul, Korea
[e-mail: ylim@sungshin.ac.kr]
*Corresponding author: Yeon-sup Lim

Received September 3, 2021; revised March 8, 2022; accepted April 2, 2022; published April 30, 2022

Abstract

Video streaming has become one of the most popular applications for mobile devices. The network bandwidth required for video streaming continues to exponentially increase as video quality increases and the user base grows. Multi-Path TCP (MPTCP), which allows devices to communicate simultaneously through multiple network interfaces, is one of the solutions for providing robust and reliable streaming of such high-definition video. However, mobile video streaming over MPTCP raises new concerns, e.g., power consumption and cellular data usage, since mobile device resources are constrained, and users prefer to minimize such costs. In this work, we propose a mobile video streaming framework over MPTCP (mDASH) to reduce the costs of energy and cellular data usage while preserving feasible streaming quality. Our evaluation results show that by utilizing knowledge about video behavior, mDASH can reduce energy consumption by up to around 20%, and cellular usage by 15% points, with minimal quality degradation.

Keywords: Multi-Path TCP, Video Streaming, Dynamic Adaptive Streaming over HTTP, Energy Efficient Communication, Mobile Computing

This work was supported by the Sungshin Women's University Research Grant of H20200096.

http://doi.org/10.3837/tiis.2022.04.009 **ISSN : 1976-7277**

1. Introduction

Video streaming is now the most popular mobile application of which traffic is more than 65% of worldwide mobile downstream traffic [1]. Dynamic Adaptive Streaming over HTTP (DASH) standardized by MPEG and 3GP aims to support high-quality streaming of media content through conventional HTTP Web servers [2] and commercial streaming services such as YouTube and Netflix utilize technologies based on DASH. Since network bandwidth on the Internet changes from time to time, DASH client players use Adaptive Bit Rate (ABR) techniques to manage this bandwidth variability. A player with an ABR technique measures bandwidth dynamically and requests fixed-length chunks of a video with an encoding rate that the network bandwidth can support [3-8].

Multi-Path TCP (MPTCP) has recently been gaining interest in the research [9-17], and standardization [18, 19], communities. MPTCP provides path diversity for end hosts by simultaneously utilizing multiple network interfaces. Based on the path diversity, MPTCP achieves greater throughput, robustness, and availability than the standard TCP, while preserving compatibility with existing TCP applications [12]. Thus, it benefits for reliable high-definition video streaming services.

DASH over MPTCP incurs additional resource costs for utilizing multiple networks, such as energy consumption and traffic charges for use of cellular networks. However, there have been relatively few experimental studies of the impact on energy consumption of DASH over MPTCP. A notable recent exception is [20]. Also, a recent study [21] shows that the MPTCP gains for streaming depend on the status of underlying networks. Oblivious use of multiple interfaces may not be necessary for streaming to maintain reasonable playback quality. In addition, the ON-OFF traffic pattern from a streaming client player [22] can cause inefficient use of multiple network paths: it is difficult for MPTCP to efficiently utilize the additional bandwidth that comes from using multiple subflows if the transfer size is insufficiently large [9]. Note that each video chunk in DASH typically contains 5-10 seconds of video, resulting in a chunk size of a few MB. The ON-OFF cycle due to such small chunks particularly affects cellular energy consumption, since each cycle triggers the high-cost promotion and tail states of cellular interfaces [23-26].

To address the issues in video streaming over MPTCP, we propose a client-side framework to adjust path usage and streaming behavior to reduce costs related to energy and cellular usage. This paper makes the following contributions:

- We introduce mDASH, a mobile video streaming over MPTCP, of which goal is to reduce both energy consumption and cellular data usage while providing feasible streaming quality. mDASH allows a client player to access information about the underlying paths and to decide the proper bit rate, burst traffic shaping, and path usage to this end. mDASH requires changes only to the client-side, not the server.
- We evaluate mDASH across several scenarios with a real implementation on an Android mobile device. Our experiments on several classical metrics (bit rate, rebufferings, etc.) use three different adaptive bit rate schemes and show that mDASH can reduce energy consumption up to around 20% with minimal degradation of video QoE metrics such as bit rate, rebuffering duration, etc. Cellular data usage is reduced by up to 15% points.

The remainder of this paper is organized as follows. Section 2 provides the background context for our work. We present our mDASH framework in Section 3. Section 4 evaluates the performance of mDASH with experiments using a real implementation. Related work is discussed in Section 5, and we conclude in Section 6.

2. Background

2.1 Behavior of DASH Client Player

A DASH server provides multiple representations of video content with different qualities, which are determined by encoding bit rates. Each representation consists of small video chunks for several seconds with corresponding quality. Based on estimated bandwidth, a DASH client player requests a chunk with an appropriate quality (i.e., bit rate) from a DASH server, which is called an adaptive bit rate selection mechanism.

A streaming session begins with an initial buffering phase during which a DASH client player fills its playback buffer to the maximum level (B_{max}). Once the player fills the buffer with the minimum number of chunks for video playback, it starts playing the video while downloading video chunks until the initial buffering completes. After completing the initial buffering phase, the player pauses downloading, and it again starts downloading when the buffer level becomes lower than the maximum level by playback. This behavior results in an ON-OFF traffic pattern where the player triggers a download period for filling the buffer and then an idle period until consuming a specific number of chunks [13, 22]. If the playback buffer depletes, the player stops playback and downloads chunks until the buffer has enough video chunks to start playback again, called the rebuffering phase.

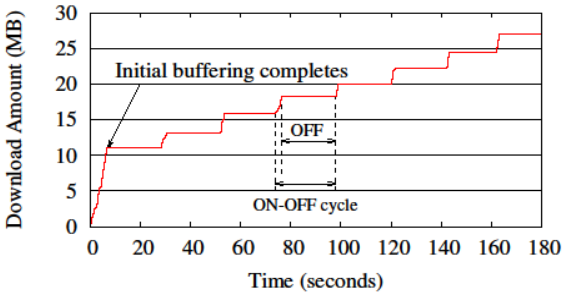


Fig. 1. Example Download Behavior in Android Netflix Player [13]

2.2 Multi-Path TCP

MPTCP utilizes multiple paths for a single data stream by subflows. The subflows are established through all end-to-end interface pairs so that they become associated with all available physical paths. MPTCP exposes these subflows as one standard TCP connection to the application.

MPTCP benefits mobile devices as follows: First, MPTCP aggregates bandwidths from multiple subflows to provide larger throughput than the standard TCP [14]. Second, if connectivity in one network becomes unstable (i.e., intermittently connected), MPTCP continuously provides a stable TCP connection over subflows through another network [15]. Finally, since MPTCP provides a standard TCP socket to user applications, it is transparent to existing TCP applications, which means that no modification is required for the application to use MPTCP [18]. In this work, we expose information at MPTCP layer to an mDASH client player to improve user experience in terms of energy, traffic cost, and streaming quality. Even though our client player is designed for this modified MPTCP, other TCP applications can still work using standard MPTCP.

2.3 Streaming, Energy Efficiency, and MPTCP

MPTCP energy efficiency depends on the available bandwidths through each interface (higher bandwidths are more efficient) and on transfer size (larger chunks spend relatively less time in the slow start phase). The energy efficiency also depends on how often the cellular interface is activated and deactivated, due to the high cost of the promotion and tail [23-26]. Therefore, MPTCP energy usage in video streaming can increase even more than typical file downloads due to its ON-OFF traffic cycle, since each cycle triggers a promotion and tail, incurring the fixed energy overhead. As an example, **Fig. 2** shows the energy consumption of Google Nexus 5 when retrieving the same 1332 sec video via two methods: direct download with *wget* and streaming using a video player. In these experiments, we measure only the energy consumption for the network transfer (measurement details are in Section 4). The figure shows the energy consumption of TCP over 10 Mbps WiFi only and MPTCP over both 10 Mbps WiFi and 10 Mbps LTE. It presents averages over three experiments, with error bars showing standard deviations. We use a state-of-art ABR scheme [6] for the streaming experiments, but regardless of the ABR scheme, the video is almost always downloaded at the maximum bit rate, since the bandwidth is sufficient to support it. Thus, for a fair comparison, the video chunks retrieved by *wget* are the ones encoded at the maximum bit rate.

As shown in **Fig. 2**, in the streaming case, the energy consumption of MPTCP is almost twice that of TCP over WiFi, even though it yields the same average bit rate. Comparing streaming with the *wget* download, we see that MPTCP streaming consumes nearly 40% more energy (410J) than the *wget* download due to the promotion and tail overhead, while TCP over WiFi streaming yields almost same energy consumption to the *wget* download since WiFi has negligible promotion and tail costs. This shows that the ON-OFF pattern in ABR streaming over MPTCP incurs additional energy costs. In this scenario, it makes sense to utilize only the WiFi for streaming, since WiFi alone can support high quality streaming, and the power consumed is lower. However, in the case of poor WiFi, all interfaces may need to be utilized to preserve video streaming quality. Given the variability in network bandwidth, this shows that *dynamic fine-grained control of MPTCP at run time* may be advantageous, to ensure good streaming quality while minimizing power consumption and cellular data usage.

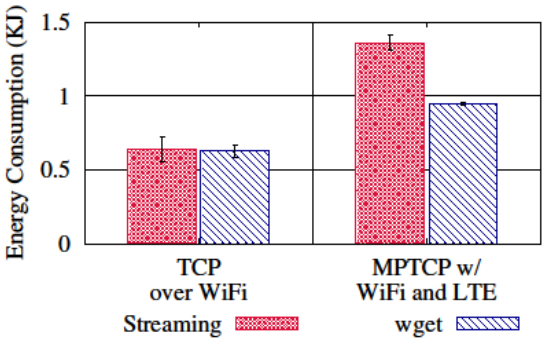


Fig. 2. Streaming vs. File Download using TCP over WiFi and MPTCP

3. Proposed Framework

3.1 Overview

mDASH is a framework that enables streaming clients to obtain high quality video streaming while reducing energy consumption and cellular data usage compared to DASH over standard MPTCP. It dynamically manages path usage based on expected energy efficiency, and accordingly adjusts the encoding bit rate of the requested video and the burst size of traffic. By avoiding the use of cellular paths that are not needed to maintain streaming quality, it also reduces the amount of cellular traffic and the corresponding expense. Note that mDASH requires changes only to the client-side, not the server-side: therefore, a standard DASH streaming server can work with mDASH clients.

Fig. 3 presents the architecture of our mDASH framework. mDASH consists of several components, two that reside in the Kernel and the remainder in user space, including the video player. The *Network Interface Monitor* (NIM) is implemented in the kernel. It measures throughput for individual MPTCP subflows and makes that information available to user-space. The *Path Usage Processor* (PUP) also deployed in the kernel, dynamically enables, and disables network interfaces for individual MPTCP flows, based on control from user-space. The *Bandwidth Predictor* (BWP) receives information from the NIM and predicts available bandwidth for MPTCP sub-flows. The *Download Controller* (DLC) monitors downloads and determines whether video chunk downloads will finish before they are needed. Finally, the *Path Usage Controller* (PUC) decides the paths to use, according to a parameterized energy model and information retrieved from the BWP. The PUC then passes activation and deactivation commands to the PUP.

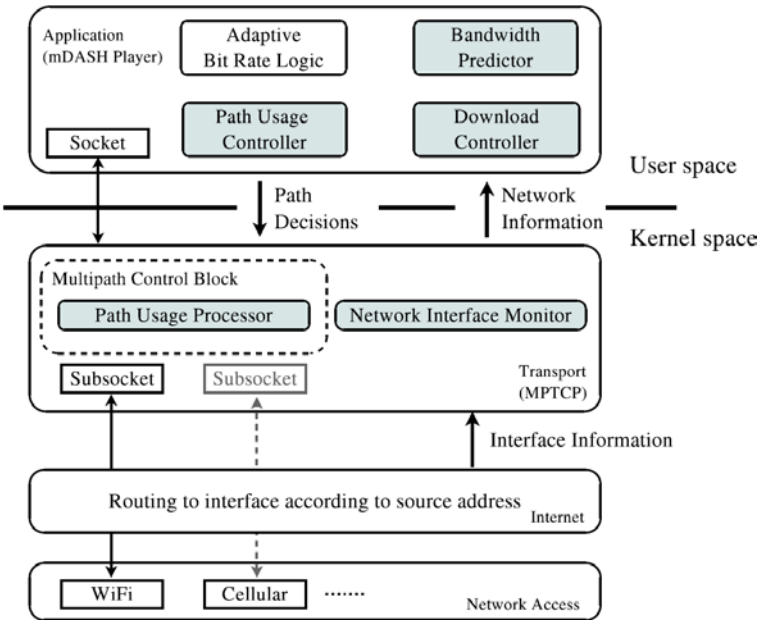


Fig. 3. mDASH Architecture

3.2 Kernel-Space Components

1) Path Usage Processor (PUP)
The PUP activates or deactivates interfaces according to commands from the PUC. When the PUP receives a command to deactivate an interface, it must handle two possible cases: there exists an established subflow on the interface, or there does not. If there exists an established subflow, the PUP de-prioritizes any subflow using that interface, using the MP_PRIO option [18], which changes subflow priority. An MP_PRIO option is added to all subsequent packets for that MPTCP connection, indicating that any subflows for that interface are low priority, and that other subflows should be preferred. If there is no established subflow, the PUP continues to block any subflow establishment over the interface. Once a path is deactivated, it remains inactive until the mDASH client reactivates it. In the case of an activation command, the PUP processes it by unblocking subflow establishment procedures and adding MP_PRIO headers that indicate normal priority. Note that the decision logic for path usage is located at the path usage controller in the mDASH client player. Once the kernel components are in place, *any* control strategy can be applied without recompiling the kernel.

2) Network Interface Monitor (NIM)
The NIM provides information about the amount of data transferred over each interface. Based on the routing information at the Internet layer, the NIM identifies the interfaces associated with each subflow. To this end, the NIM checks the destination entry of the socket as follows: The destination entry (*struct dst_entry*) includes a pointer to the associated network device (*struct net_device *dev*). This pointer has a name of the interface and another pointer to the wireless device information (*struct wireless_dev *ieee80211_ptr*). Thus, the NIM checks if *dev* of a socket has a valid *ieee80211_ptr* to identify whether a socket is for a connection over a WiFi interface [12, 28].

Both the NIM and the PUP provide kernel interfaces to clients. In our current implementation, a client must have super-user access to deliver control commands to the kernel. By extending a user-level API such as *setsocketopt*, our framework can work without super-user access. However, Extending the APIs to allow non-privileged access to clients we leave as future work.

3.3 User-Space Components

1) Bandwidth Predictor (BWP)
The BWP predicts future throughputs of all active subflow based on samples. Once a chunk download completes, the BWP computes a bandwidth sample as transferred bytes over each interface divided by the chunk download time. Based on collected samples, BWP applies a Holt-Winters time-series forecasting algorithm [27] to obtain predictions. If an interface becomes inactive during or at the end of a chunk download, the BWP will not be able to collect any throughput information from that interface. In this case, the BWP uses the previous prediction based on old observed samples.

2) Download Controller (DLC)
The DLC monitors the download progress of each chunk to ensure smooth streaming with efficient traffic shaping. If the DLC detects a buffer depletion during a chunk download due to a sudden bandwidth drop, the DLC tries to cancel the current download

and choose a chunk with a proper bit rate for the network bandwidth measured by the DLC. To this end, the DLC individually monitors download bandwidth for a chunk and it abandons a download if the buffer level becomes lower that a threshold and the remaining download time is expected to be longer than the buffer level (i.e., expecting a buffer-depletion) like the download abandonment mechanism in [46]. In this paper, we use $2B_K/3$ for the threshold, where B_K is the playback buffer level at the beginning of kth chunk download. Note that the DLC does not cancel the current download and continues with the current chunk if the expected chunk size of the new bit rate is larger than the remaining amount of the current chunk download. Once detecting the possibility of buffer depletion, the DLC signals the path usage controller, notifying it that more bandwidth is required, forcing PUC to activate all available network interfaces.

Recall that the client player stops downloading chunks when the buffer reaches the maximum buffer size B_{max}. After then, the player periodically starts downloads, resulting in continuous cellular tail energy consumption if the cellular interface is in use. Transferring data in large bursts is thus more energy efficient for cellular interfaces due to such tail overhead. To mitigate unnecessary energy consumption for the cellular tail state, the DLC dynamically adjusts B_{max} according to the buffer level as with [42]. If the network bandwidth is sufficient, it takes short time for the client to fill the buffer to the maximum level. In this case, the client does not download for a while, and then it tries to download a small number of chunks after consuming for playback, which results in continuous ON-OFF traffic patterns. To mitigate such ON-OFF traffic patterns, if the buffer level is larger than a threshold B_{th}, the DLC increases B_{max} by L seconds up to B_{MAX} to trigger a longer download (a large burst transfer), and thus reduces the energy waste of cellular tails. However, B_{MAX} must be constrained since a mobile device has limited memory and buffering large amounts can be wasteful if a user does not play the entire buffered video.

3) Path Usage Controller (PUC)

The PUC dynamically decides what paths to use according to energy efficiency when the client initiates a chunk download. By doing so, it also reduces data usage over an unneeded interface. To this end, the PUC utilizes an energy model for each interface to estimate the energy consumption for downloading a specific amount of data based on predicted bandwidth. There exists a large literature about the energy models of mobile devices concerning bandwidth, signal strength, simultaneous use of interfaces, etc. [11, 12, 23, 24, 25, 26]. In this work, we utilize an existing parameterized energy model described in [11, 12].

Even though the PUC is designed to deal with all possible path combinations, we focus on the PUC cases that switch between only using WiFi and using both WiFi and LTE, since WiFi is typically free for use and its energy overhead is relatively small compared to cellular interfaces. Assume that a device retrieves streaming video through MPTCP over WiFi and LTE. When the kth chunk download starts, the expected download size $\widehat{M_k}$ and the transfer size through the WiFi and LTE interfaces $\widehat{M_k^W}$ & M_k^L can be calculated, respectively:

$$\widehat{M_k} = \mathrm{R_k} \times \max(\lceil (B_{max} - B_k)/L \rceil \times L, L) \tag{1}$$

$$\widehat{M_k^W} = \widehat{M_k} \times \frac{\widehat{C_k^W}}{\widehat{C_k^W} + \widehat{C_k^L}}, \qquad \widehat{M_k^L} = \widehat{M_k} - \widehat{M_k^W} \tag{2}$$

, where B_{max} is the maximum buffer level, L is the chunk length in seconds, and $\widehat{C_k^W}$ and $\widehat{C_k^L}$ are the predicted bandwidths of WiFi and LTE for k-th chunk download, respectively.

Given the expected transfer sizes and predicted bandwidths, the energy model provides the expected energy consumption of TCP over WIFI/LTE and MPTCP. For example, $E_W(M_k, C_k^W)$ returns the expected energy consumption of TCP over WiFi when transferring M_k bytes with , C_k^W bps. By comparing the expected power costs for the different configurations, the PUC selects the most energy efficient path usage for the next chunk download.

However, for smooth streaming, the PUC needs to guarantee that its decision does not trigger a buffer depletion, particularly when deciding to use only one interface. To prevent this, the PUC conservatively checks one more condition before switching to use only one interface based solely on energy efficiency: the predicted bandwidth of the selected interface must increase the buffer level after a chunk download with the chosen bit rate. In addition, the PUC may receive a signal to activate all available interfaces from the DLC in the middle of a chunk download: the DLC notifies the PUC after abandoning the current download and attempting to lower the bit rate when it recognizes that the chunk download with the currently selected bit rate cannot complete without depleting the buffer. In this case, to quickly prevent buffer depletion and a resulting playback stall, the PUC activates all subflows.

Note that when the PUC deactivates an interface, the BWP cannot obtain any bandwidth observations. To provide a minimum level of information about all interfaces for the BWP at the beginning, the PUC activates all available subflows for τ seconds at the beginning of an MPTCP connection: we choose $\tau = 10$s for the implementation.

4. Evaluation

4.1 Quality Metrics

We use following performance metrics to compare streaming strategies:

- Energy: we measure the amount of energy consumed to complete the entire playback, excluding the baseline energy consumption to play the video without any communication and disruption (see our experimental setup in Section 4.3 for details).
- Average bit rate: This is the average of the downloaded bit rates while downloading all chunks, i.e., $(\sum_{k=1}^{K} R_k)/K$ where K is the total number of chunks and R_k is the bit rate of the kth chunk.
- Number of bit rate changes: This is a count for bit rate changes, i.e., the number of times when the bit rate of a downloading chunk becomes different from the previous one, i.e., $\sum_{k=2}^{K} 1(R_{k-1} \neq R_k)$.
- Number of rebufferings: This is the number of times a playback stalls due to rebuffering.
- Total rebuffering duration: This is defined as the amount of time spent in the rebuffering phase, while waiting for the video buffer to fill, during the entire playback.
- Fraction of traffic downloaded over the LTE cellular interface.

4.2 Adaptive Bit Rate Schemes and Test Video

We utilize the following ABR schemes to evaluate mDASH:

- Tian: Tian et al. [5] suggest a feedback controller to control the playback buffer level according to predicted throughputs, the buffer level, and its trend. In our experiments, the control parameter K_p is set to 0.1 as described in [5].
- BBA: Huang et al. [6] implement a rate adaptation scheme that chooses bit rates as a function of the playback buffer level. They evaluate its performance through Netflix deployment. We use 20s and 70s for the reservoir and cushion parameters respectively.
- BOLA: Spiteri et al. [8, 38] propose BOLA, a buffer-based adaptation scheme that minimizes the drifts of buffer level, i.e., to try to maintain a stable buffer size. BOLA is now part of ABR schemes in the official DASH reference player dash.js [38]. BOLA also abandons the current download if the score of the currently selected bit rate becomes lower than that predicted from another bit rate. We set the parameter γ to $5.0/L = 1.0$ and use their log-based utilization function.

Several approaches have been recently proposed to utilize artificial intelligence algorithms for video streaming: Mao et al. [39] and Huang et al. [40] propose ABR schemes that utilize Deep Reinforcement Learning. Lee et al [41] implement PERCEIVE using an LSTM (Long Short Term Memory) model. Note that mDASH can be coupled with such kinds of ABR schemes as well. However, even AI inferences require a large computation overhead, resulting in high energy consumption, which can overwhelm communication and streaming costs. Also, mobile devices typically have a lack of computing resources and power supply to process such complex operations. Therefore, we choose the above schemes that require less computation overhead than AI-based schemes, minimizing the effect of energy consumed by ABR schemes themselves.

The common parameters B_{max} (maximum buffer level) and L are set to 100s and 5s, respectively. For the experiments, we select a video clip from [29]. The length and resolution of the video clip are 1332 seconds long and 2160p (3840 by 2160 pixels), respectively. We set the streaming server to have six representations with 144p to 1080p resolutions as Youtube does. To this end, we re-encode the video clip at each resolution and create DASH representations with 5s chunks ($L = 5$), which requires bit rates from 0.26 Mbps to 8.47 Mbps.

Table 1. ABR Parameters

	Parameter	Description
Common	L = 5s	Chunk Length
	$B_{max} = 100$s	Maximum buffer level where a player stops downloading chunks
Tian [5]	$p = 0.1$	Weight in buffer size adjustment factor
	$q_0 = 70$s	Reference buffer level
BBA [6]	$r = 20$s	Reservoir
	$cu = 70$s	Cushion
BOLA [8]	$\gamma = 1.0$	Input weight parameter

In our experiments, all ABR schemes begin by downloading a chunk at the lowest bit rate, i.e., 0.26 Mbps for 144p. Note that without the mDASH framework support, Tian, BBA, and BOLA always simultaneously utilize both the WiFi and LTE interfaces. The buffer adjustment parameters of mDASH framework, B_{th} and B_{MAX}, are set to 70s and 300s, respectively. **Table 1** summarizes the parameters used for the ABR schemes.

4.3 Experiment Method

We use a Google Nexus 5 running Android modified with our mDASH implementation. The mobile device is connected to the Internet through a WiFi access point and AT&T LTE, i.e., it has WiFi and cellular interfaces. Note that a default primary interface is an interface to initiate a connection. In the experiments, we set WiFi as a default primary interface.

The mobile device communicates with a server running Ubuntu Linux 12.04 with the MPTCP implementation [30]. The server has a single Gigabit Ethernet interface connected to a campus network. We use Apache 2.2.22 as the HTTP server for the DASH video contents and enable HTTP persistent connections with the default Keep Alive Timeout (5 sec).

We collect energy traces using Qualcomm's Trepn Profiler [31] while setting the display brightness level of the mobile device to the minimum and connecting the device to external power after removing its battery. Energy consumption can come from two sources: the energy to play the video on the display, and the energy to download the video over the network. To distinguish these sources, we measure the energy consumption when the device completes playback of the video file stored in flash memory using the same settings. This measures the playback cost in isolation from the network.

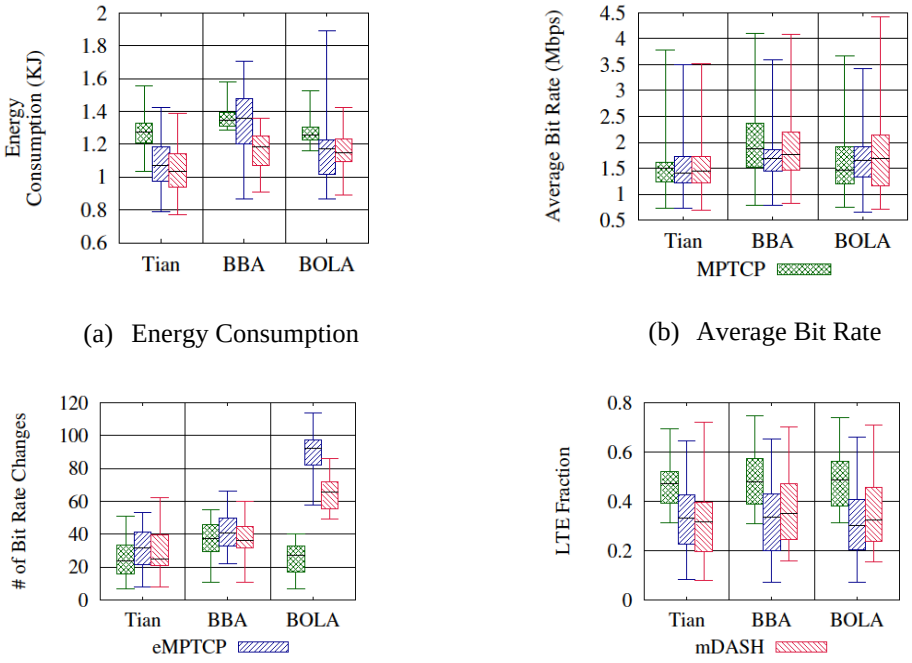
For experiments, we have implemented the mDASH framework and ABR schemes in the Google Nexus 5 using ExoPlayer [32]. **Fig. 4** exhibits a screenshot, which is measuring streaming quality, download activities, and energy/cellular usage, running our implementation at the device. For the experiments, we utilize a set of public cellular bandwidth traces [33]. We randomly take 30 combinations using the throughput traces as the bandwidth trace of each interface (we also conduct trace-driven simulations for all possible combinations and observe similar results to the experiments). In each experiment, for each bandwidth scenario, we rate-limit the bandwidths to match those from the chosen trace scenario. Rate-limiting is performed on the server-side using the Linux traffic control utility *tc* [34]. Note that the cost gain achieved by mDASH depends on bandwidth scenarios: if WiFi bandwidth is consistently low to support streaming quality, mDASH continues to utilize LTE, resulting in similar costs compared to streaming over MPTCP. If WiFi bandwidth is enough to support the highest streaming quality, mDASH never uses LTE, maximizing the cost gain.

Fig. 4. mDASH implementation

4.4 Results

Now we investigate experimentally whether mDASH provides greater energy efficiency and lower cellular data usage for the existing DASH strategies. In addition to using standard MPTCP, we also evaluate the ABR schemes over eMPTCP, which is an MPTCP variant to improve the energy efficiency with minimal impact on download latency [12]. Note that eMPTCP is designed to work at the transport layer without any application awareness: eMPTCP monitors subflow status at the transport layer, such as available bandwidth, and dynamically decides the most energy-efficient path usage. By comparing the three ABR algorithms with eMPTCP against their counterparts using mDASH, we can evaluate the benefit of application awareness.

Fig. 5, 6, and 7 present Whisker plots or CDFs of the metrics for each ABR scheme. **Table 2** also lists the average values of each metric. As shown in **Fig. 5(a)**, both eMPTCP and mDASH enable the ABRs to reduce energy consumption. Tian, BBA, and BOLA with eMPTCP exhibit 15.1%, 2.8%, and 6.5% less energy consumption than with MPTCP. mDASH yields average energy savings of 18.1%, 14.8%, and 7.2% compared to Tian, BBA, and BOLA with MPTCP, respectively. This shows that mDASH achieves more energy savings than eMPTCP, across all ABR schemes.



(a) Energy Consumption

(b) Average Bit Rate

(c) # of Bit Rate Changes

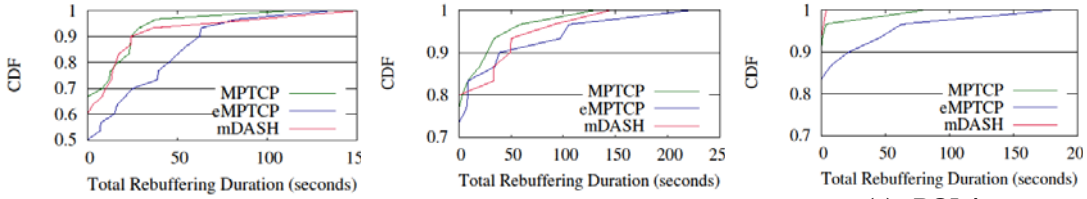
(d) LTE Fraction

Fig. 5. Comparison with MPTCP, eMPTCP, and mDASH

eMPTCP and mDASH slightly degrade the average bit rate while saving energy. As shown in **Fig. 5(b)** and **Table 2**, Tian, BBA, and BOLA using eMPTCP exhibit 4.9%, 13.7%, 1.2% lower average bit rates than using MPTCP, respectively. In contrast, mDASH decreases the average bit rate of Tian and BBA by 2.4% and 5.9%, which are smaller degradation than eMPTCP. Compared to using BOLA with MPTCP, using mDASH improves the average bit

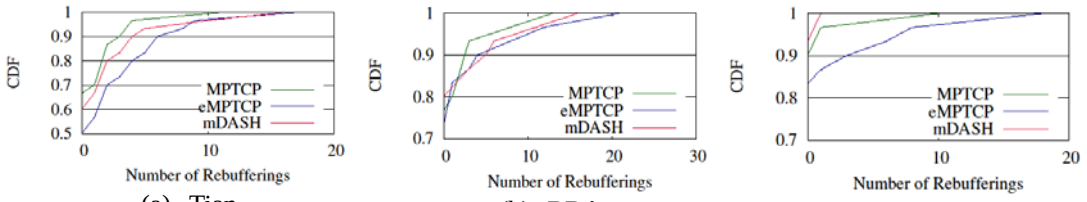
rate by 4.1%. This shows that mDASH seeks to maintain better streaming quality than eMPTCP while saving energy. This is due to the application awareness of mDASH to satisfy application requirements, while eMPTCP only tries to minimize energy consumption without any application awareness.

Using eMPTCP or mDASH, the ABRs bit rate changes more frequently. In **Fig. 5(c)**, we observe that the ABRs with eMPTCP exhibit the highest average number of bit rate changes, followed by those with mDASH and MPTCP. mDASH yields slightly more number of bit rate changes than MPTCP since it switches bit rates if a buffer depletion is expected.



(a) Tian (b) BBA (c) BOLA

Fig. 6. Total Rebuffering Duration with MPTCP, eMPTCP, and mDASH



(a) Tian (b) BBA (c) BOLA

Fig. 7. Number of Rebufferings with MPTCP, eMPTCP, and Mdash

Table 2. Average Metrics for ABR Schemes

	Tian			BBA			BOLA		
	MPTCP	eMPTCP	mDASH	MPTCP	eMPTCP	mDASH	MPTCP	eMPTCP	mDASH
Energy (J)	1273.53	1081.09	**1042.68**	1364.51	1325.92	**1163.23**	1266.86	1184.93	**1175.84**
LTE Fraction (%)	47.34	34.32	**33.60**	48.67	**34.19**	37.93	48.12	**33.64**	35.71
Avg. Bit Rate (Mbps)	**1.64**	1.56	1.60	**2.04**	1.76	1.92	1.72	1.70	**1.79**
# of Bit Rate Changes	**25.73**	31.13	28.97	36.53	41.6	**36.3**	**24.17**	90.2	65
Total Rebuffering Duration (s)	**9.99**	22.20	13.65	**9.38**	17.41	13.41	2.83	10.61	**0.19**
# of Rebuffering	9.08	9.25	**7.88**	**8.80**	9.50	9.15	7.07	8.85	**2.89**

As can be seen in **Fig. 5(d)** and **Table 2**, both eMPTCP and mDASH successfully reduce cellular data usage by adaptively changing path usage. By reducing cellular path usage, which usually requires more energy, both obtain energy savings. Using eMPTCP reduces cellular data usage by 13.0%, 14.5%, and 14.5% for Tian, BBA, and BOLA, respectively, while mDASH saves LTE data usage by 13.7%, 10.7%, and 12.4%. Note that mDASH causes ABRs to exploit the cellular path more aggressively than eMPTCP. Even with more use of the cellular path, mDASH obtains better energy savings than eMPTCP. This demonstrates that mDASH successfully shapes burst (large) transfers for streaming, resulting in more energy-efficient use of cellular paths than sporadic small transfers.

As shown in **Fig. 6**, ABRs with mDASH and eMPTCP suffer from total rebuffering durations more than MPTCP. In the case of Tian, 67% of the bandwidth scenarios with MPTCP do not experience rebuffering at all, while only 50% do with eMPTCP and 60% do with mDASH. In the case of BBA and BOLA, mDASH avoids rebuffering more than MPTCP and eMPTCP: there are no rebufferings in 80% of the bandwidth scenarios with mDASH while there are 76% with MPTCP and 73% with eMPTCP. In the average comparison in **Table 2**, Tian and BBA with mDASH yield longer rebuffering durations than with MPTCP and shorter than with eMPTCP, while BOLA significantly shortens rebuffering duration by using mDASH. This shows that eMPTCP extremely tries to reduce energy consumption, thus, it experiences frequent long rebufferings. In contrast, mDASH does a similar or slightly larger number of rebufferings than MPTCP.

Fig. 7 presents the CDF of the number of rebufferings with each streaming scheme. As shown in **Fig. 7**, Tian and BBA with mDASH yield slightly more rebufferings than with MPTCP and less than with eMPTCP while BOLA with mDASH does the smallest number of rebufferings. In the averages shown in **Table 2**, Tian and BOLA experience the least frequent rebufferings with mDASH. We see that BOLA works best with mDASH while effectively reducing energy and cellular data usage and preserving streaming qualities.

5. Related Work

Several approaches have been proposed to investigate and address topics in video streaming under mobile and harsh environments, such as bandwidth estimation errors, limited energy supply, smoothness in streaming quality, fairness across users, and multipath streaming. However, there are a few rigorous studies on enhanced MPTCP streaming for reducing costs in multiple aspects, such as energy and cellular usage, using a real implementation and experimental measurements.

Corbillion [35] proposes a scheduler for improving the performance of video streaming over MPTCP. Their approach needs to modify a video sender to work with the proposed scheduler. Also, they evaluate their approach via trace-driven simulation.

Wu et al. [36] examine streaming MPTCP environments, it does not consider DASH streaming, so their quality metric is based on distortion. They do not consider other costs such as energy and cellular data usage.

Ojanpera et al. [37] evaluate rate adaptation schemes with and without the support of the proposed network management system under MPTCP enabled networks. However, in this study, MPTCP is not integrated with their scheme, so MPTCP does not play any other role except for providing more bandwidth and the management scheme does not consider any MPTCP specific issues such as energy consumption.

Ferlin et al. [43] propose a framework that integrates forward error correction (FEC) into MPTCP for latency-sensitive application traffic such as video streaming. They implement an

XOR-based FEC scheme for MPTCP that yields low loss rates with relatively small FEC overhead. However, this work just focuses on reducing latencies without considering multiple aspects in video streaming such as energy usage.

Elgabli and Agarwal [44] formulate an optimization problem for streaming QoE subject to the available bandwidth and link preferences and propose an online algorithm to solve it. Even though their approach can be used to mitigate cellular usage by setting the preference, it does not consider a real energy model. Therefore, there is no guarantee that their solution yields the best energy efficiency. Also, this work is just based on the trace-driven simulation.

Xing et al. [45] propose an MPTCP scheduler for live video streaming. By dealing with the out-of-order packet problem caused by packets from multiple paths with different latencies, their proposed scheduler provides sufficient throughput for upper-layer applications. Although their scheduler enables streaming clients to obtain better throughput, it does not consider issues on energy and cellular usage.

The most closely related piece of work appeared recently by Han et al. [20]. MP-DASH [20] is a path scheduling framework for video streaming in MPTCP that schedules video chunks to preserve user quality of experience: MP-DASH activates a cellular subflow only when a WiFi subflow does not provide enough bandwidth to meet deadlines for video chunks. We compare our mDASH framework with MP-DASH as follows: First, MP-DASH does not deal with other streaming-related operations: it only manages path usage to satisfy deadline constraints. In contrast, our framework consists of more components for controlling streaming behaviors, such as DLC for abandonment and burst traffic shaping, to overcome situations of QoE degradation that can be encountered by bandwidth dynamics while controlling path usage. Second, although the MP-DASH path control operations reduce cellular usage and consequently power consumption, MP-DASH implementation does not rigorously take account for cost optimization in terms of energy and cellular usage; they also quantify energy use via simulation, whereas we measure energy directly on a smartphone. Third, since the MP-DASH path scheduler locates at the kernel and operates based on given chunk size and deadline, the MP-DASH streaming player needs to continuously call the scheduling function at the kernel, whereas our PUP sends the control signal to the kernel only when path usage changes are required while it reads interface information exposed from the kernel. Lastly, MP-DASH needs to modify both the client and server, whereas ours requires only client-side changes, allowing incremental deployability.

6. Conclusion

This paper proposes, implements, and evaluates mDASH, a cost-efficient framework for mobile video streaming over MPTCP. mDASH framework provides abstractions for its components so that it can extend for new mobile devices and communication technologies such as 5G by applying corresponding models. An mDASH client player dynamically chooses a proper video bit rate together with energy-efficient path usage that avoids unnecessary use of the cellular interface. Our experiments show that mDASH successfully saves energy and cellular data usage, compared to the ABR schemes running over standard MPTCP, while still yielding similar streaming quality.

References

[1] Sandvine, "The Mobile Internet Phenomena Report," 1H 2020. [Online]. Available: https://www.sandvine.com/download-report-mobile-internet-phenomena-report-2020-sandvine

[2] T. Stockhammer, "Dynamic adaptive streaming over HTTP – standards and design principles," in *Proc. of ACM MMSys*, pp. 133–144, 2011. Article (CrossRef Link)

[3] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with Festive," *IEEE/ACM Transactions on Networking*, 22(1), 326–340, Feb 2014. Article (CrossRef Link)

[4] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, 32(4), 719–733, April 2014. Article (CrossRef Link)

[5] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic HTTP streaming," in *Proc. of ACM CoNEXT*, pp. 109–120, 2012. Article (CrossRef Link)

[6] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 187–198, 2014. Article (CrossRef Link)

[7] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. of ACM SIGCOMM*, pp. 325–338, 2015. Article (CrossRef Link)

[8] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proc. of IEEE INFOCOM*, 2016. Article (CrossRef Link)

[9] Y.-C. Chen, Y.-S. Lim, R. J. Gibbens, E. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of Multipath TCP performance in wireless networks," in *Proc. of ACM IMC*, pp. 455–468, Oct. 2013. Article (CrossRef Link)

[10] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "WiFi, LTE, or both? Measuring multi-homed wireless Internet performance," in *Proc. of ACM IMC*, pp. 181-194, 2014. Article (CrossRef Link)

[11] Y.-S. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and R. J. Gibbens, "How green is multipath TCP for mobile devices?," in *Proc. of ACM AllThingsCellular*, pp. 3–8, 2014. Article (CrossRef Link)

[12] Y.-S. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, R. J. Gibbens, and E. Cecchet, "Design, implementation, and evaluation of energy-aware multi-path TCP," in *Proc. of ACM CoNEXT*, pp. 1-13, 2015. Article (CrossRef Link)

[13] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths," in *Proc. of ACM CoNEXT*, pp. 147-159, 2017. Article (CrossRef Link)

[14] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. of ACM SIGCOMM*, pp. 266–277, 2011. Article (CrossRef Link)

[15] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley, "Opportunistic mobility with multipath TCP," in *Proc. of ACM MobiArch*, pp. 7–12, 2011. Article (CrossRef Link)

[16] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. of USENIX NSDI*, pp. 399–412, 2012. Article (CrossRef Link)

[17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. of USENIX NSDI*, pp. 99–112, 2011. Article (CrossRef Link)

[18] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP Development," *RFC 6182 (Informational)*, 2011. [Online]. Available: https://datatracker.ietf.org/doc/rfc6182/

[19] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," *RFC 6824*, 2013. [Online]. Available: https://datatracker.ietf.org/doc/rfc6824/

[20] B. Han, F. Qian, L. Ji, V. Gopalakrishnan, and N. Bedminster, "MP-DASH: Adaptive video streaming over preference-aware multipath," in *Proc. of ACM CoNEXT*, pp. 129–143, 2016. Article (CrossRef Link)

[21] C. James, E. Halepovic, M. Wang, R. Jana, and N. Shankaranarayanan, "Is multipath TCP (mptcp) beneficial for video streaming over DASH?," in *Proc. of IEEE MASCOT*S, pp. 331–336, 2016. Article (CrossRef Link)
[22] A. Rao, Y.-s. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous, "Network characteristics of video streaming traffic," in *Proc. of ACM CoNEXT*, pp. 1–12, 2011. Article (CrossRef Link)
[23] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proc. of ACM IMC*, pp. 280–293, 2009. Article (CrossRef Link)
[24] J. Huang, Q. Feng, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. of ACM MobiSys*, pp. 225–238, 2012. Article (CrossRef Link)
[25] N. Ding, D. Wagner, X. Chen, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *Proc. of ACM SIGMETRICS*, pp. 29–40, 2013. Article (CrossRef Link)
[26] A. Nika, Y. Zhu, N. Ding, A. Jindal, Y. C. Hu, X. Zhou, B. Zhao, and H. Zheng, "Energy and performance of smartphone radio bundling in outdoor environments," in *Proc. of WWW*, pp. 809–819, 2015. Article (CrossRef Link)
[27] P. J. Rockwell and R. A. Davis, *Introduction to Time Series and Forecasting*, Springer, 1994. [Online]. Available: https://link.springer.com/book/10.1007/978-3-319-29854-2
[28] Y.-S. Lim, "On Leveraging Multi-Path Transport in Mobile Networks," *Doctoral Dissertations*, 890, 2017. Article (CrossRef Link)
[29] JackFrags. 4k gaming montage. [Online]. Available: https://4ksamples.com/4k-gaming-montage/
[30] C. Paasch and S. Barre, "Multipath TCP in the Linux kernel,". [Online]. Available: https://www.multipath-tcp.org
[31] Qualcomm. Trepn power profiler, [Online]. Available: https://developer.qualcomm.com/forums/software/trepn-power-profiler
[32] Google. Exoplayer. [Online]. Available: http://google.github.io/ExoPlayer/.
[33] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: Analysis and applications," in *Proc. of ACM MMSys*, pp. 114–118, 2013. Article (CrossRef Link)
[34] Linux Foundation, Linux advanced routing and traffic control. [Online]. Available: http://lartc.org/howto/.
[35] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon, "Cross-layer scheduler for video streaming over MPTCP," in *Proc. of ACM MMSys*, pp. 1-12, 2016. Article (CrossRef Link)
[36] J. Wu, C. Yuen, B. Cheng, M. Wang, and J. Chen. "Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks," *IEEE Transactions on Mobile Computing*, 15(9), 2345–2361, 2016. Article (CrossRef Link)
[37] T. Ojanperä and J. Vehkaperä, "Network-assisted multipath dash using the distributed decision engine," in *Proc. of IEEE ICNC*, pp. 1–6, 2016. Article (CrossRef Link)
[38] K. Spiteri, R. K. Sitaraman, and D. Sparacio, "From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player," *ACM Transaction on Multimedia Computing, Communications, and Applications*, Vol. 15, No. 25, pp. 1-29, 2019. Article (CrossRef Link)
[39] H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," in *Proc. of ACM SIGCOMM*, pp. 197-210, 2017. Article (CrossRef Link)
[40] T. Huang, C. Zhou, X. Yao, R. Zhang, C. Wu, B. Yu, and L. Sun, "Quality-Aware Neural Adaptive Video Streaming with Lifelong Imitation Learning," *IEEE Journal on Selected Areas in Communications*, 38, 2324-2342, 2020. Article (CrossRef Link)
[41] J. Lee, S. Lee, J. Lee, S.D. Sathyanarayana, H. Lim, J. Lee, X. Zhu, S. Ramakrishnan, D. Grunwald, K. Lee, S. Ha, "PERCEIVE: Deep Learning-Based Cellular Uplink Prediction Using Real-Time Scheduling Patterns," in *Proc. of ACM MobiSys*, pp. 377–390, 2020. Article (CrossRef Link)
[42] Y. Kang, "A Novel Bit Rate Adaptation using Buffer Size Optimization for Video Streaming," *International Journal of Internet, Broadcasting and Communication*, Vol.12, No.4, pp. 166-172, 2020. Article (CrossRef Link)

[43] S. Ferlin, S. Kucera, H. Claussen, and O. Alay, "MPTCP Meets FEC: Supporting Latency-Sensitive Applications Over Heterogenous Networks," *IEEE/ACM Transactions on Networking*, Vol. 26, No. 5, pp. 2005-2018, 2018. Article (CrossRef Link)
[44] A. Elgabli and Vaneet Aggarwal, "SmartStreamer: Prefernce-Aware MultipathVideo Streaming over MPTCP," *IEEE Transactions on Vehicular Technology*, Vol. 68, No. 7, pp. 6975-6984, 2019. Article (CrossRef Link)
[45] Y. Xing, K. Xue, Y. Zhang, J. Han, J. Li, J. Liu, and R. Li, "A Low-Latency MPTCP Scheduler for Live Video Streaming in Mobile Networks," *IEEE Transactions on Wireless Communications*, Vol. 20. No. 11, pp. 7230-7242, 2021. Article (CrossRef Link)
[46] Y. Kang and Y.-S. Lim, "Bit Rate Adaptation Using Linear Quadratic Optimization for Mobile Video Streaming," *Applied Sciences*, Vol. 11, No. 1, 2021. Article (CrossRef Link)

Yeon-sup Lim is an assistant professor at Department of Convergence Security Engineering in Sungshin Women's University. Before joining Sungshin Women's University, he was a research staff member at IBM T. J. Watson Research Center, Yorktown Heights, NY USA from 2017 to 2020. He received the Ph.D. from College of Information and Computer Sciences at University of Massachusetts Amherst under the supervision of Professor Don Towsley. His research interests include the broad range of topics in the area of networks, with emphasis on multipath transport protocol, mobile computing, Internet measurement, and complex networks.