Cross-layer scheduler for video streaming over MPTCP

Xavier Corbillon, Ramon Aparicio-Pardo, Nicolas Kuhn, Géraldine Texier, Gwendal Simon

Cross-Layer Scheduler for Video Streaming over MPTCP

Xavier Corbillon
Télécom Bretagne, IRISA, France

Ramon Aparicio-Pardo
Université Nice Sophia Antipolis, I3S, France

Nicolas Kuhn
Centre National d'Etudes Spatiales (CNES), France

Géraldine Texier
Télécom Bretagne, IRISA, France

Gwendal Simon
Télécom Bretagne, IRISA, France

ABSTRACT

Transport protocols that can exploit multiple paths, especially MPTCP, do not match the requirements of video streaming: high average transmission delay, too strict reliability, and frequent head-of-line phenomenons resulting in abrupt throughput drops. In this paper, we address this mismatch by introducing a cross-layer scheduler, which leverages information from both application and transport layers to re-order the transmission of data and prioritize the most significant parts of the video. Our objective is to maximize the amount of video data that is received in time at the client. We show that current technologies enable the implementation of this cross-layer scheduler without much overhead. We then demonstrate the validity of our approach by studying the performance of an optimal cross-layer scheduler. The gap between the performance of the traditional scheduler versus the optimal scheduler justifies our motivation to implement a cross-layer scheduler in practice. We propose one implementation with basic cross-layer awareness. To evaluate the performance of our proposal, we aggregate a dataset of real MPTCP sessions and we use video stream encoded with HEVC. Our results show that our cross-layer proposal outperforms the traditional scheduler. Viewers not only benefit from the inherent advantages of using MPTCP (such as a better resilience to path failure) but also get a better QoE compared to the traditional scheduler.

CCS Concepts

•**Networks** → *Packet scheduling;*

Keywords

MPTCP; HEVC; Video Streaming; Cross-Layer Scheduler

1. INTRODUCTION

The multi-path communication protocols, and especially the Multi-Path Transmission Control Protocol (MPTCP), are at a turning point of their development. Since most devices have several available network accesses (typically Wi-Fi and cellular for mobile devices), MPTCP is expected to enable quicker and more stable communication by concurrently exploiting several network paths. Researchers promoting MPTCP at the IETF record successful deployment experiments [4, 6]. Meanwhile, popular devices now natively support MPTCP (for example the iOS7 [1]). However, MPTCP has not been broadly adopted by multimedia content providers for at least two reasons. First, experiments have revealed some weaknesses, including performance degradations when the underlying paths are not homogeneous [11] and the lack of support in middleboxes [5, 16, 17]. Second, the current behavior of MPTCP introduces more frequent *head-of-line blocking* phenomenons than in TCP. These phenomenons, which result in short but abrupt throughput drops, make MPTCP a poor transport protocol for video streaming, for which network stability is a requirement.

In this paper, we address the current concerns about MPTCP regarding the transport of multimedia content. For video streaming, transport protocols based on TCP have some drawbacks, including high average transmission delay, no consideration of application error concealment techniques, and lack of throughput stability. The issue related to head-of-line phenomenon in MPTCP is an additional weakness. To make MPTCP friendlier to video streaming, our idea is to enable and exploit interactions between the application and the transport layers. The interactions between these layers have recently received a surge of interest, as epitomized by the Transport Services (TAPS) working group at the IETF [14]. The advent of companies that all together develop their own video player, implement mobile Operating Systems (OSs), and manage the delivery chain make the borders between the application and transport layers more porous. We show in this paper that these interactions improve the performance of MPTCP regarding video streaming. More generally, our study extends to any multi-path transport protocol mimicking the behavior of TCP (including reliability, congestion control, and in-order packet delivery).

We focus on the multipath streaming of a video. Our objective is to increase the quality of experience (QoE) of the viewers by maximizing the reception of decodable video data in difficult conditions (video bit-rate close to the available bandwidth and small application buffer). We do not deal with adaptive mechanisms like implemented in HTTP Adaptive Streaming (HAS). Rather, we consider

MMSys'16, May 10-13, 2016, Klagenfurt, Austria
DOI: http://dx.doi.org/10.1145/2910017.2910594

that the adaptive mechanism has selected one representation for a video chunk (or segment) and we deal with the transport of this chunk. Our proposal applies to both live and on-demand services.

Paper Contributions. We introduce the concept of a *cross-layer scheduler*, which leverages information from both application and transport layers to schedule the delivery of video packets in MPTCP. We provide a theoretical analysis of the *potentials* of our cross-layer scheduler by the mean of an Integer Linear Program (ILP). This optimization model validates the motivation for video-aware MPTCP. We also propose an implementation based on realistic cross-layer interactions. To evaluate the performance of this cross-layer scheduler in realistic environments, we gather a dataset containing the traces of real streaming traffic using MPTCP on Ethernet, Wi-Fi and cellular network accesses. These traces[1] include the timestamps of both sending and arrival times of each packet in each path. Our evaluation reveals that our cross-layer scheduler improves the QoE of users, thus it transforms MPTCP into a more efficient multimedia transport protocol. We also show that our implementation is still far from being as efficient as the optimal, so our proposal is a first step toward better video-aware implementations of MPTCP.

Paper Structure. We provide in Section 2 a tour on both video steaming and MPTCP. We detail the main changes that are inherent to the use of multiple paths at the transport layer and we justify the rationale behind using MPTCP in our study. We introduce our cross-layer scheduler in Section 3 with different types of information awareness. We describe our optimization model in Section 4. The set of tools we implemented to acquire the datasets is described in Section 5. Finally, we evaluate the performance in Section 6 and conclude the paper with a discussion in Section 7.

2. BACKGROUND

This Section shows background material related to MPTCP and multimedia stream.

2.1 Multi-Path Networking

We focus on a cross-layer solution between the application and the transport layers. The main features of transport protocols are the reliability (how to deal with packet losses), the in-order delivery (should packets be delivered at the destination application in the same order as they were emitted at the source) and the congestion control. The application layer is responsible for generating the data and guaranteeing that both ends can interpret the data. The interactions between the application and the transport layers are mainly done through the introduction of *cascading buffers*. We show in Figure 1 the packet transmission process from the standpoint of both application and transport layers. We consider an End-to-end (E2E) communication between a server and a client and we assume it is possible to divide an E2E flow into several *subflows*. Each subflow contains a series of data *packets*, which are carried out on different paths. The light gray TCP sending buffers represent the additional buffers introduced when multi-path is used at the transport layer.

Application Sending Buffer. The application pushes the data to be transmitted in this buffer. The application manages the input/output of the data; the default output process being First-In First-Out (FIFO).

Multi-Path Sending Buffer. The application sees the MPTCP connection as a unique socket even though multiple network paths are used. In particular the congestion and the flow controls are *coupled* among paths. The Multi-Path Sending Buffer is the main element of this global socket at the server side. It is implemented into specific libraries in the kernel of OSs. The data from the Application Sending Buffer is packetized into TCP packets, kept in the Multi-Path Sending Buffer, and then pushed into TCP Sending Buffers with respect to the global congestion control.

TCP Sending Buffer. Each network path has its own TCP Sending Buffer. Each subflow is a regular TCP connection but the congestion and flow controls are implemented in the Multi-Path Sending Buffer, so the Congestion Window (CWND) of the TCP connection is globally managed such that each TCP Sending Buffer receives sets of packets according to a global management.

Receiving Buffer. MPTCP implements only one receiving buffer, which gets all incoming packets from all subflows before the delivery to the application. When in-order delivery is required (which is the case in MPTCP), packets must be stored until all previous packets have been delivered. If a packet is missing (like packet 10 in Figure 1), *head-of-line blocking* can happen when too many subsequent packets are stored in this buffer (here packets ranging from 11 to 13).

Application Receiving Buffer. Packets are delivered to the application. More details for video streaming application can be found in Section 2.2.

Each of these buffers introduces some extra-delay [7] but they guarantee reliable, in-order E2E communication on multi-paths. The recent research activities related to multi-path transport protocols, such as Concurrent Multipath Transfer SCTP (CMT-SCTP) or MPTCP, have dealt with the scheduling of packets at the Multi-Path Sending Buffer (which packets to send to which TCP Sending Buffer) [2, 22, 25] and the management of retransmission in case of packet losses [24, 27].

We focus on MPTCP [15] since it mimics the behavior of TCP in a multi-path environment, and thus it is friendly to most today's applications based on HTTP. Other transport layer protocols include CMT-SCTP [18] and Multi-Path Real-time Transport Protocol (MPRTP) [28]. The former features the same coupled congestion control as MPTCP (Opportunistic Linked-Increases Algorithm (OLIA) [20, 26]) but it has no default options, so parameter setting is hard, and it suffers from deployment issue since it is based on SCTP. The latter essentially targets video delivery based on the UDP. In the recent years, most video applications have adopted HTTP and TCP since these protocols are not filtered by middleboxes in the E2E channel, as opposed to UDP. Finally, it is worth pointing out that even though we choose MPTCP as the transport layer protocol, the idea and general concept of our

[1]To enable more realistic and fair comparisons with future works on MPTCP, these traces and all the material used in this paper are publicly available at http://dash.ipv6.enstb.fr/mmsys2016

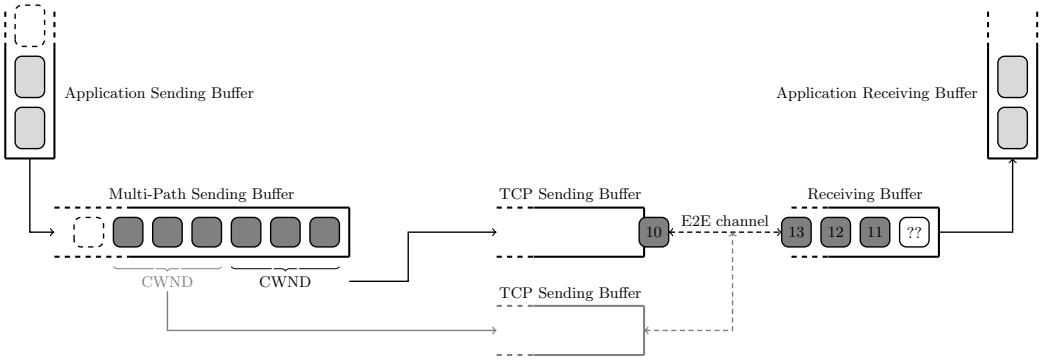

Figure 1: Multi-path data transmission at the application and transport layers. The light gray rectangles represent data unit generated by the application (for example a video frame), while the dark grey rectangles are TCP packets resulting from data packetization. In this example, the packet with id 10 was lost and it is retransmitted.

scheduler can easily be applied to other solutions.

2.2 Multimedia Stream Structure

The goal of a video encoder is to convert the original sequence of images (arrays of pixel values) into a video bit-stream. The decoder does the opposite. The idea that is now adopted in video compression is the principle of hierarchical structure of video stream data. The bit-stream is cut into *frames*, which have temporal dependencies with regards to their types: Intra (I), Predicted (P) or Bidirectional (B) pictures. Each frame is cut into sets of macroblocks, including *tiles* in High Efficiency Video Coding (HEVC). In the following, the term *video unit* refers to an independent piece of data generated by the application with dependencies to other units. The term *video unit* can be interpreted as either frame or tile.

A drawback of current video compression approaches is the propagation of errors due to the causal dependency between video units. Indeed, the loss of one video unit can make the decoder unable to process all the video units that depend on the missing video unit. The video decoders implement error-concealment strategies to minimize the distortion on the video. In the case of frames, errors may propagate until the next I-frame in the worst case. In the case of tiles, the propagation of errors is similar since a tile in a given frame may also depend on tiles in other frames. In this case, the error is spatially restricted in the displayed image.

To fix the problems due to video unit losses, the client application implements an Application Receiving Buffer as explained in Section 2.1 and shown in Figure 1. This buffer introduces a delay at the reception before the video unit is consumed (decoded and displayed), this delay being hopefully large enough to allow data retransmission in case of packet losses. To increase the probability to get all video units in time, it is tempting to implement a large Application Receiving Buffer, but it is at the expense of a larger *video playback delay*, *i.e.*, the difference between the time the client requests the video and the time the video starts playing. Studies have shown that a large playback delay is a major cause of abandonment for viewers [21].

The management of data in the Application Sending Buffer can also introduce extra-delay. The motivation for this buffer comes from the fact that the order in which the frames are displayed is not the same as the order in which the frames are encoded and should be decoded. In today's systems, data are streamed in the *frame decoding order*, so, in the case of live streaming, the application has sometimes to wait for the encoding of the next frames before sending an encoded frame to the Multi-Path Sending Buffer. Note that in the case of Video on Demand (VoD) streaming where the video has already been fully encoded, the application does not need to buffer video units in the Application Sending Buffer since the video data are stored in the frame decoding order.

Finally, the video that is streamed from the server to the client is carried out in a *package*. The application that generates the video prepares the encoded video data so that standard video players can interpret and read the stream. The preparation of the content is mainly its integration into a *multimedia container*. In general, recent video containers follow the ISO base media file format (ISOBMFF), which specifies a structure and metadata for multimedia content. The format contains a series of *boxes*, which provide information on the timing and the structure of the actual video data (the *mdat* box).

2.3 Existing Work on Carrying out Video Through Multiple Paths

Although many papers have studied multi-path video streaming in the past, few of them have addressed both application and transport layers, and even fewer have dealt with realistic features of MPTCP. In the multimedia community, papers have mostly focused on the application level (a seminal work is [19]). In the network community, a research axis is to deal with streaming in specific wireless environment (for example [8]). Our paper is between these approaches (without any assumption on the physical layer).

In one of the first publications on live tests performance evaluation of MPTCP [9], the authors have claimed that the default MPTCP can reasonably be used to transport Netflix and Youtube videos. However, some issues inherent in the use of multiple paths to carry out videos have already been identified in the literature [19], in particular in heterogeneous environments where the use of multiple paths may result in poor QoE.

The scheduler is identified as one key component driving the performance of MPTCP [4] but some other approaches have also been considered. Using an adequate scheduler has shown to be a "winning game" change with other

purposes than carrying out video [2, 22, 25]. Some related works consider either a cross-layer Forward Error Correction (FEC) coding [30] or "transport layer aware" prefetching process [9]. These proposals are in line with the services the TAPS working group at the IETF aim to provide: smart interactions between the multi-path transport and the video generation to improve QoE.

Some papers have been discarded from the analysis, because (*i*) they use old versions of MPTCP and simulations [12]; (*ii*) the clients download data from various servers [10], which is not in the scope of this paper nor a default use case of MPTCP.

To the best of our knowledge, the only cross-layer proposal for MPTCP and video streaming is in [24]. The authors propose a cross-layer scheme between the Media Access Control (MAC) and the transport layers to detect when a path fails so that the traffic can be proactively re-routed, or some packets discarded at the receiver level to prevent the video from freezing. Since this proposal deals with other layers than ours, both approaches are complementary and the advantages are cumulative.

3. VIDEO-CONTENT AWARE SCHEDULER OVER MPTCP

This section presents our video-aware scheduler over MPTCP. First, we expose limitations that are inherent to having a protocol stack with independant layers. Then, we describe how cross-layering approaches can be introduced to assess those limitations. We then show how to pass information from the application layer to the scheduler of MPTCP. Finally, we introduce the algorithm that is exploited in the remaining of this paper.

3.1 Limitations Related to Protocol Stacks with Independant Layers

Current implementation of the transport layers leads to information loss between the application layer and the transport layer. Indeed, most transport protocols, including TCP, MPTCP and UDP, work at the bit-stream level and are not aware of the structure of the transported data. On the positive side, this design makes protocols independent from applications. On the negative side, this unawareness of the structure of the bit-stream (here the unit structure of the video stream) comes with some drawbacks. Particularly, data are transmitted from the MPTCP Sending Buffer in the same order as they arrived to this buffer, even if some data should no longer be sent, for example an obsolete video unit whose playback deadline has already expired. Such an event is likely to occur in transport protocols that guarantee in-order delivery, such as TCP and MPTCP because the loss of one data packet can delay the delivery of multiple following packets.

3.2 Motivation for Cross-layer Scheme

Our idea is to have a common view for both the application and transport layers as shown in Figure 2. In practice, this cross-layer vision is a *scheduler*, which is aware of the video content, the application deadlines, and the network conditions (see Section 3.3 for more details). The cross-layer scheduler decides if and when a video unit is given to the transport layer, so the cross-layer scheduler changes the order of video units *before* sending the packets to the TCP Sending Buffers. Our motivation is that the cross-layer scheduler can exploit the triple awareness to increase the number of video units that are decodable without error at the client side. This objective typically matters when the network conditions suddenly change and when the Applicative Receiving Buffer is small. The cross-layer scheduler can then choose to cancel the transmission of low-importance video units and rather to prioritize the transport of the most important ones.

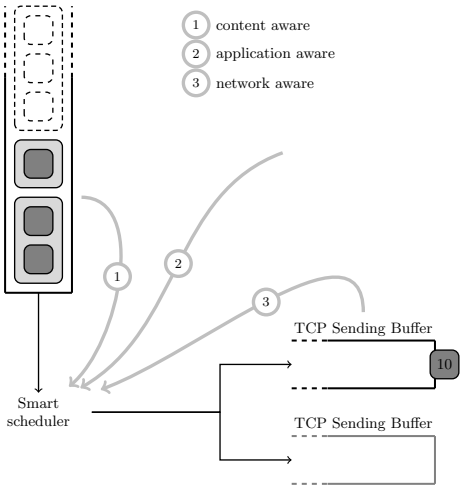

Figure 2: Our cross-layer scheduler

The cross-layer scheduler can be implemented either at the transport layer or at the application layer. In the following of the paper, we consider an implementation at the application level because it is more convenient. Here are some details.

At the application level the cross-layer scheduler relies on an existing transport protocol (in our case MPTCP), which is already deployed, without further modification. The application uses some system calls to get information on the MPTCP stack and to know when it can send new data to the transport layer. The unit structure of the video stream is known. At the client side, only the video player needs to be updated to deal with the frame order. Overall, a content provider that develops both the streaming server and the video client players (for example YouTube, Dailymotion, and Netflix) can implement a cross-layer scheduler without much difficulties.

The implementation of the cross-layer scheduler at the transport level is more challenging. At this level, the cross-layer scheduler has direct access to MPTCP stack in the kernel. So it can directly access the network information. Moreover it can decide how to cut the data into packets, when to send a packet on the network and on which path. But any proposal corresponds to a new MPTCP protocol, which has to be validated and standardized. The cross-layer scheduler also changes the behaviour of standard transport layer protocols since the order in which the data is given to the transport layer stack is not the order in which the data are received in the Multi-Path Sending Buffer. With the re-ordering process, the client should understand this new protocol, which requires a new implementation of the kernel at the client side.

3.3 Passing Information between the Different Layers

We present in this Section the different information that the cross-layer scheduler can use to take decisions. We distinguish between the awareness of (*i*) information from the video content, (*ii*) some network characteristics and (*iii*) information on the application.

3.3.1 Video content awareness

What benefits. The cross-layer scheduler knows it transports an encoded video and understands its structure. Each video unit may have causal dependencies with other video units. The cross-layer layer knows all those dependencies and so is able to infer whether the client can decode a video unit at a specific time or not. Each video unit is associated with a specific video frame (even for tiles). The cross-layer scheduler knows the picture order counts (POC) and the coding identifier (coding *id*) of each frame. With the POC, the scheduler knows the display deadline of the associated picture.

How to implement. To get video content awareness, the cross-layer scheduler reads the video container. The video units are part of the encoded video structure, so the cross-layer scheduler can extract the information from the right ISOBMFF boxes.

3.3.2 Network awareness

What benefits. The cross-layer scheduler knows the status of the network, including the smoothed Round-Trip Time (sRTT), the global moving average bandwidth and the loss probability of each path. With the sRTT, the cross-layer scheduler can estimate the arrival time of each packets, and choose to send a video unit close to its deadline on the path with the shortest sRTT. From the moving average bandwidth, the scheduler can choose to drop some less important video units to favor others. From the loss probability, it can choose to send highly important video units on the path with the smallest loss probability.

How to implement. The MPTCP socket in the kernel already knows the sRTT and has enough data to compute the global moving average bandwidth and the loss probabilities. A cross-layer scheduler implemented inside the MPTCP stack already has this information. At the application layer, the cross-layer scheduler needs to implement a feedback loop from the MPTCP socket. Although such feedback does not exist in open-source implementations yet, the development is not challenging since the server hosts both the MPTCP implementation and the application.

3.3.3 Application awareness

What benefits. The cross-layer scheduler knows the status of the video player on the client. It can estimate the *lag* at the client side. This lag is the delay introduced by the client before starting to display the decoded video. The cross-layer scheduler can also estimate when the client started playing the video. From the lag, the start timestamp and the POC, the cross-layer scheduler can estimate the deadline of each video unit.

How to implement. The cross-layer scheduler needs a feedback from the client. The client can add extra information in its HTTP requests or use Real-Time Transport Control Protocol (RTCP) to implement this feedback in practice. The lag is a fixed value, which can be communicated to the server. The hardest thing is to estimate the timestamp when the client starts playing the decoded video. We do not want to synchronize the clocks of both server and client. It is not needed to have an accurate timestamp as long as this timestamp is over estimated (*i.e.* in the future compared to the real start timestamp). Indeed, if the timestamp is under estimated, the cross-layer scheduler is more conservative (it does not send video units that have no chance to arrive in time).

3.4 Our Algorithm

The goal of our algorithm is to prioritize the video units that are the most likely to be received in time. The cross-layer scheduler is content-aware (it knows how to extract video units) and application-aware (it is able to estimate their playback deadline from an applicative feedback loop). It is also network-aware (it knows which path the MPTCP stack selects to send the next packet as well as the sRTT of this path). The cross-layer scheduler supposes the current packet arrives at the client side after a travel time of $\frac{sRTT}{2}$.

The available video units are sorted by coding id (*i.e.* the original video bit-stream order). The cross-layer scheduler iterates over the list and checks whether the estimated playback time of each video unit is before the estimated playback deadline. If one of the units validates the estimated transmission success test, then it is sent. Otherwise (if no available video unit is likely to arrive on time), the scheduler selects the video unit with the smallest coding id. Note that this last case is less likely to happen.

4. OMNISCIENT OPTIMAL MODEL

We now introduce a model, which computes the optimal transmission of a given video by exploiting multiple paths. We aim to compute an upper-bound of the performance of a scheduler so that we can objectively rate the performance of our proposal. The model determines which video data should be carried by which MPTCP packet so that the number of video units that are decodable at the reception side is maximum. It is a theoretically optimal cross-layer scheduler with a full awareness: the video content, the whole application interactions between server and client, and the network characteristics. These parameters are inputs of the model. The model goes beyond this awareness and is *omniscient* because all the events of the video transmission (including the time each packet takes to reach the reception) are known by the solver. In other words the optimal solution is a scheduler that can take decision with the knowledge of the future.

4.1 Input

We summarize in Table 1 the main notations that we introduce in the following.

MPTCP. Our model is based on the transmission of a file using the MPTCP protocol. The input contains the records of a full, completed transmission, including when each MPTCP packet of the bit-stream has been sent at the server and successfully received at the client. We denote by $\mathcal{P}$ the set of packets that have been successfully received by the client for the whole transmission. Each packet $p \in \mathcal{P}$ is

Name	Description
$s_v \in \mathbb{N}^*$	Number of packets needed to transmit video unit $v \in \mathcal{V}$
$t^p_{rx} \in \mathbb{R}^+$	Applicative reception timestamp of packet $p \in \mathcal{P}$
$t^v_{deadline} \in \mathbb{R}^+$	Decoding deadline of video unit $v \in \mathcal{V}$
D_v	Set of video units needed to decode video unit $v \in \mathcal{V}$ due to dependencies

Table 1: ILP notations

characterised by the timestamp $t^p_{rx} \in \mathbb{R}^+$ of its delivery to the client application.

The optimal solution of our solver is based on an actual MPTCP transmission, without interfering with it. Indeed, the cross-layer scheduler does not impact the scheduler within MPTCP since our scheduler re-orders the packets of the video units inside the bit-stream, so *before* sending the data to the MPTCP packet scheduler. The scheduler within MPTCP generates the same records of packet transmission regardless of the implementation of the cross-layer scheduler (provided that at any moment the packet scheduler has available data to send).

Video. Our model is based on the notion of video unit that is introduced in Section 2.2, *i.e.*, the model applies indifferently to either video frames or video tiles. We denote by $\mathcal{V}$ the set of video units that have to be streamed from one server to a client. Each video unit $v \in \mathcal{V}$ should be played by the application at the client side at a given time, which is noted $t^v_{deadline}$. In other words, the whole data related to video unit v must be in the Applicative Receiving Buffer at $t^v_{deadline}$. The transmission of a video unit v requires the transmission of s_v different packets. We consider packet padding if the size of the video unit is not a multiple of a Maximum Transmission Unit (MTU).

A video unit $v \in \mathcal{V}$ can be decoded and displayed if and only if all the video units on which v depends have been received and decoded on time. We denote by D_v the set of video units on which v depends. In practice, cross-layer error concealment strategies are implemented at the video decoder to allow partial decoding of a unit v even if one of the dependencies of v is missing. However, (*i*) the strategies depend on the video decoder, (*ii*) error concealment strategies are not widely implemented for tiles yet, and (*iii*) the distortion due to a missing unit depends on multiple parameters. Therefore we model here one of the most constraining environments for the QoE where we consider that only one missing video unit $v' \in D_v$ at the client side prevents the video unit v to be decoded.

4.2 Decision Variables

The main decision variable that our cross-layer scheduler should take is, for each MPTCP packet, which video unit data should be sent. We model it as:

$$x^p_v = \begin{cases} 1, & \text{if } p \text{ transports data for } v \\ 0, & \text{otherwise} \end{cases}, \forall (v,p) \in \mathcal{V} \times \mathcal{P}$$

This decision variable can be interpreted as follows. Every time a new packet can be sent through any of the TCP network paths, the cross-layer scheduler should buffer a MPTCP packet in the MPTCP Sending Buffer. This packet transports data related to a given video unit.

Integer Linear Program formulation

$$\max_{\boldsymbol{x},\boldsymbol{y}} \quad \sum_{v \in \mathcal{V}} y_v$$

$$\text{such that} \sum_{p \in \mathcal{P}} x^p_v = s_v \cdot y_v \qquad \forall v \in \mathcal{V} \qquad (1a)$$

$$\sum_{v \in \mathcal{V}} x^p_v \leqslant 1 \qquad \forall p \in \mathcal{P} \qquad (1b)$$

$$|D_v| \cdot y_v \leqslant \sum_{v_d \in D_v} y_{v_d} \qquad \forall v \in \mathcal{V} \qquad (1c)$$

$$x^p_v \cdot t^p_{rx} \leqslant t^v_{deadline} \qquad \forall (v,p) \in \mathcal{V} \times \mathcal{P} \qquad (1d)$$

$$y_v \in \{0,1\} \qquad \forall v \in \mathcal{V}$$

$$x^p_v \in \{0,1\} \qquad \forall (v,p) \in \mathcal{V} \times \mathcal{P}$$

Instead of using the FIFO policy depending on the frame decoding order, the cross-layer scheduler can re-order the data. The decision variable x indicates the data carried by the MPTCP packets that are successfully transmitted.

A video unit $v \in \mathcal{V}$ is successfully received and decodable if (*i*) s_v packets carrying data related to v are sent, (*ii*) the latest of these packets arrives before $t^v_{deadline}$, and (*iii*) all video units in D_v are received and decodable. We introduce a new decision variable, which is:

$$y_v = \begin{cases} 1, & \text{if } v \text{ was received and decodable} \\ 0, & \text{otherwise} \end{cases}, \forall v \in \mathcal{V}$$

4.3 Optimization Objectives

The main goal of our cross-layer scheduler is to improve the quality of the decoded video at the client side. In our model, we aim at maximizing the number of decodable video units, formally $\sum_{v \in \mathcal{V}} y_v$. In general, the more video units are received, the better is the QoE, but the distortion due to two missing minor video units is sometimes smaller than the loss of one major video unit. However, note that our objective tends to favour those video units on which a large number of video units depends, (typically the I-frames for video units as frames), and these video units are often the most impacting. Another option, which we do not choose in this paper, is to pre-compute the QoE of the decoded video for any solution, that is for any subset of missing video units. This option requires the computation of 2^n combinations for a video of n video units. Our future works include the development of more sophisticated objective functions, which would be a good trade-off between the simplicity of our current objective function and the accuracy of the full QoE computation.

4.4 Integer Linear Program

The ILP model is defined by the set of equations (1). The constraints (1a) ensure that a video unit v is received ($y_v = 1$) if and only if exactly s_v different packets are used. The constraints (1b) ensure that a packet can transport data related to one and only one video unit. The constraints (1c) state that a video unit v is decodable ($y_v = 1$) if and only if all video units on which v depend are also received and decodable.And finally, the constraints (1d) ensure a packet p can transmit a video unit v if and only if p is received by the client before the decoding deadline of v.

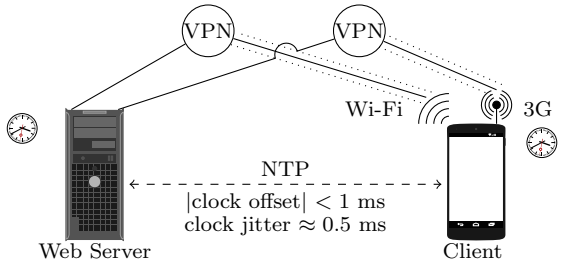

Figure 3: Testbed used for generating traces that characterize the transmission of a file using MPTCP

5. MPTCP DATASET

To evaluate the performance of cross-layer schedulers in realistic configurations, we need the logs of a real MPTCP transmission, including, for each packet, the transmission timestamp, the reception timestamp, the size of the payload and the path on which this packet has been carried. To the best of our knowledge, no such dataset is publicly available, so we collected a series of traces based on a campaign of measures. We describe the platform, and then this dataset. We finally highlight some observations, which confirm weaknesses of MPTCP.

5.1 Measurement Platform

To collect real traces of MPTCP traffic, we set up a platform composed of an MPTCP web server, an MPTCP client, and a layer three (L3) UDP Virtual Private Network (VPN). The platform is depicted in Figure 3.

We measured the reception and transmission timestamps respectively on the client and the server sides. To ensure the consistency of the timestamps on two machines, both clocks were synchronized. We used the Network Time Protocol (NTP) with the server as master and the client as slave. When both computers are close enough, NTP can lead to a synchronization offset smaller than 1 ms with a drift that can be neglected during a run. To guarantee the synchronization of both computers, we installed them in the same room.

Although both the client and the server are in the same Local Area Network (LAN), for the sake of the representativity of our results, we need the MPTCP traffic to face issues that are inherent to a transmission over the public Internet, such as Round-Trip Time (RTT) variations or different congestion levels within the network. To guarantee that packets travel on the public Internet, we used a layer three UDP VPN.[2] We opened a VPN tunnel for each network interface, and then we set routing and firewall rules to force MPTCP traffic to exclusively use those VPN tunnels. The client and the server were located in Rennes, France,and the VPN server in Roubaix, France, in the data-center of a major European cloud provider. Thus, the connection is realistic in the sense that the whole connection through the VPN is typical from a connection between an end-user and the server of a cloud service.

At the server side, we used a modified MPTCP linux kernel, which captures and logs the timestamp when a packet has been selected for transmission. This modified kernel also logs some useful information known by the MPTCP socket when the packet is transmitted. Those modifications are only related to the logs, and thus they do not impact the behavior of MPTCP.

At the client side, a *tcpdump* listened on each interface for MPTCP data packets and stored the reception timestamp and the data sequence number of each received packet. Combining with the logs from the server, we thus collected both sending and reception timestamps of each successfully received packets.

[2]By using UDP as the underlying transport protocol, we maintain the main properties of the link (loss, RTT variance, *etc.*) except for the MTU [23].

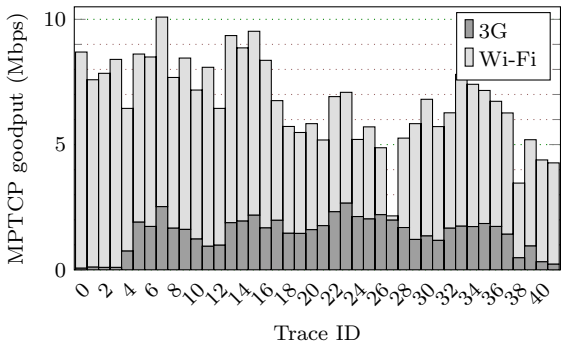

Figure 4: Average MPTCP goodput of all the traces

5.2 Description of the Dataset

We measured multiple transmissions of a 100 MB file from the server to the client during two weeks in September 2015. In the present paper, we limit ourself to a dual Wi-Fi/3G connection. Once a *trace* (the transmission of the file) is recorded, a random delay is set before another transmission is measured, thus the transmissions happened at various times and days.

The dataset captures all key events of series of MPTCP transmissions. The dataset records the time at which the packet scheduler chose the path for the emission of the packet, the time at which the packet is sent on the network, the time at which the packet is received by the client, and the time at which the data is delivered to the application layer on the client side.

In Figure 4, we show the average *goodput* of the subset of traces that we use in our paper hereafter. In this Figure, the average capacity of the 3G path can directly be read on the y-axis but the average capacity of the Wi-Fi path should be computed as the difference between the total MPTCP average goodput and the 3G goodput. The traces exhibit a high variability although the records have been picked in the same configuration. The benefits of MPTCP are noticeable on a series of traces where one of the network interfaces had a very low available bandwidth, but the other network path is well used, so MPTCP manages to guarantee an average goodput of around 6 Mbps for the vast majority of traces.

We show in Figure 5 the goodput, averaged every second, for both interfaces during one given trance. The head-of-line blocking phenomenon is visible twice, at the 28^{th} and 39^{th} seconds. Negative events on the 3G paths (packet loss on a slow path) prevent any new packet from being delivered on the fast Wi-Fi path. In that case, the server stops using the Wi-Fi links until the missing packets are received from the 3G path. Since the Wi-Fi is the fastest link, the overall goodput of the multi-path transmission significantly drops during a couple of seconds. For the delivery of a multimedia stream, the impact on the QoE can be disastrous.

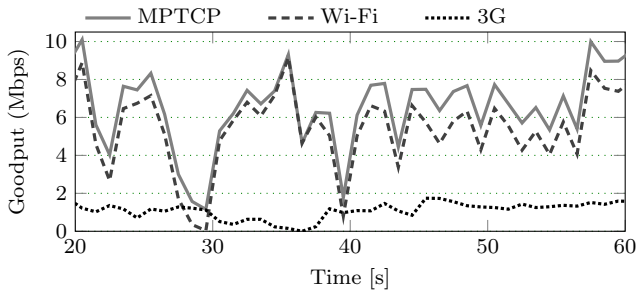

Figure 5: Measure of the goodput every second for a trace

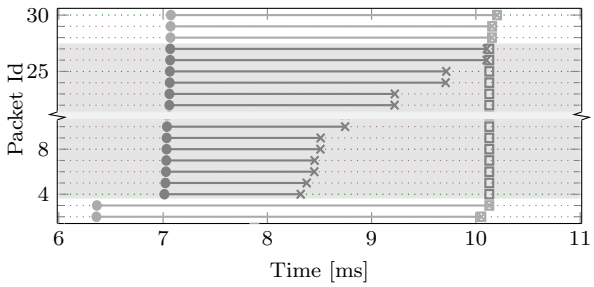

Figure 6: Head of line blocking: each line represents one packet, with the id on the y-axis. The circle mark on the left is the timestamp of the packet transmission, the cross mark is the reception timestamp, and the square mark is the timestamp of the delivery to the Application.

To further illustrate the head-of-line blocking issue, we show in Figure 6 the timestamps related to the transmission of 29 consecutive packets. Each line represents one packet. For each packet, the circle mark on the left is the time at which the packet has been sent. The cross mark is the time at which the packet has been received in the Multi-Path Receiving Buffer. And the square mark is the time at which the packet has been pushed to the Application Receiving Buffer. The two first packets (2 and 3) have traveled over the network for a long time. All the packets ranging from 4 to 27 arrived at the client *before* both packets 2 and 3. Since MPTCP guarantees in-order delivery, the packets from 4 to 27 have to be buffered into the MPTCP Receiving Buffer, which result in a head-of-line blocking phenomenon. The background is gray for all packets buffered before the delivery. Finally, all packets are pushed to the Application Receiving Buffer at the same time upon the reception of packet 3 but this late delivery may be too late depending on the deadline of the video units.

6. PERFORMANCE EVALUATIONS

We now deal with the performance evaluation. We first describe the settings of the simulations. We present a glimpse of all the results by analyzing one trace. Then, we measure the gap between the FIFO scheduler, which mimics the behavior of the traditional MPTCP scheduler, and the optimal omniscient model. We show here that significant gains can be achieved by the implementation of a cross-layer scheduler. We then compare our cross-layer scheduler to the FIFO scheduler. The results are encouraging, since they show that our scheduler improves the QoE. Finally, we compare our cross-layer scheduler to the optimal one. We see that our cross-layer scheduler is still far from the optimal one, which motivates further studies in this area.

6.1 General Settings

We distinguish two main streaming use cases: the VoD and the live-streaming scenario. One of the main differences between both use cases is the number of video units that are available in the Application Sending Buffer. For the live-streaming, only a few video units are available at any moment of the transmission (those that have been generated at that time), while, for the VoD case, all video units are in the Application Sending Buffer at the beginning of the transmission. The behaviors of the cross-layer scheduler are equivalent in both use cases. We present only the VoD use case results in this Section.

The reference FIFO scheduler. We compare our cross-layer scheduler and the optimal solution to the *reference FIFO* scheduler, which mimics the current implementation of MPTCP without any cross-layer scheduler. The video units are processed in the bit-stream order. The scheduler splits them into multiple data packets of size MTU (with padding if needed). The scheduler then gives the packets to the MPTCP stack, which sends them to the client by using one of the available paths.

Video Settings. We consider a 25 s-long video segment. This duration is large enough to observe the impact of the variability of the network on the streaming and not too far from a typical video segment length (10 s) in HAS. Our experiment can be interpreted as a HAS segment download: a segment with a specific bit-rate has been selected and now, due to the short playback delay, the segment has to be downloaded to avoid stopping the video playback. This video segment is an extract from Blender's *Tears of Steel* video [3]. It was encoded from the raw pictures into an HEVC video, main profile, with a resolution of 1920x1080 pixels, a bit-rate of 6 Mbps. We used *ffmpeg* version 2.7.2-2+*b*1 with the *libx265* version 1.7 module. We used the *medium* preset and we forced an I frame every 94 frames.

Video Quality. Recent studies have shown that the quality of the video experience depends on multiple criteria, including not only the video quality but also the video *lag*, *i.e.*, the delay between the transmission start and the video display start to the client [21, 29, 31]. In the following, we consider these two axis.

We consider four different lags ranging from 500 ms, which corresponds to a tight and challenging configuration with a small Applicative Receiving Buffer, to 2 s, which is reasonable in today's video services. These configurations are more constraining than what is observed today because we anticipate lag reduction from video providers.

As for the video quality, we first have to reconstruct the video as the client would see it after the transmission losses. The three algorithms that are evaluated here (the FIFO scheduler, our cross-layer scheduler, and the omniscient optimal algorithm) produce a file with all the video units that are received in time by the client. To reconstruct the video, we take the original video (the one that is transmitted) and we remove from its bit-stream the

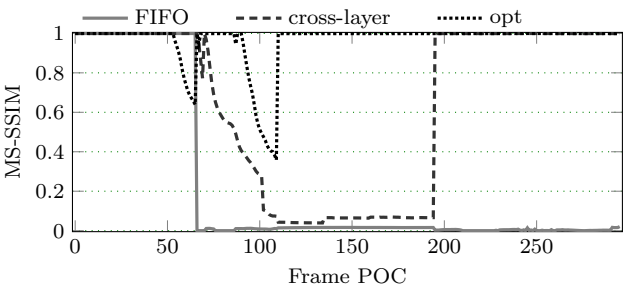

Figure 7: QoE frame by frame

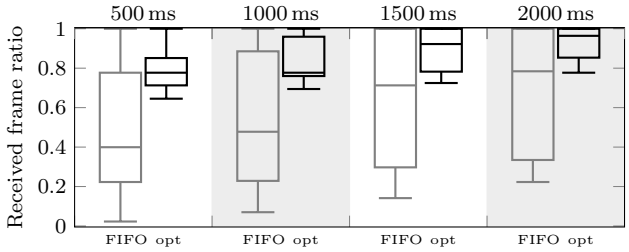

Figure 8: Ratio of frame received in time by the client for a 25 s 1080p HEVC video with an average bit-rate of 6 Mbps

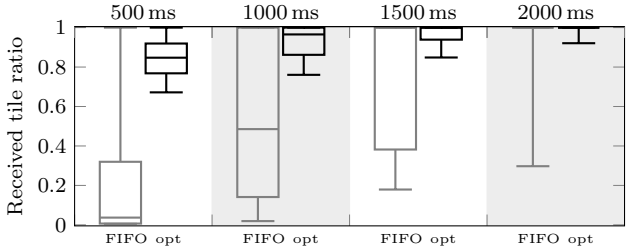

Figure 9: Ratio of tile received in time by the client for a 4 s 1080p HEVC video with four tiles per frame with an average bit-rate of 6 Mbps

lost video units. Then we use *ffmpeg* to decode the video without the lost units and to generate a *yuv420p* video. To compute an objective value of video quality, we used the Video Quality Measurement Tool (VQMT) software [13] and the traditional Peak Signal to Noise Ratio (PSNR). We used the original video (the one before the transmission) as reference to compute the PSNR or the Multiscale - Structural Similarity (MS-SSIM) metrics.

Network Settings. The transport layer behavior is based on the real MPTCP dataset (see Section 5.2). This dataset uses jointly a 3G and Wi-Fi connections. We chose this multi-path configuration for mainly two reasons. Firstly because this is the most common setup for today's mobile devices. Secondly because using MPTCP jointly on 3G and Wi-Fi is known to create jitter in the delivery of data due to the head-of-line blocking, which adds unpredictable variations in the transport layer behavior. In the following, the presented results are statistics from 168 independent transmissions. Those transmissions are simulated on 168 independent sections of our MPTCP dataset. The three schedulers (FIFO, cross-layer and optimal) use the same set of transmissions.

6.2 The Main Idea in a Nutshell

We present here the main idea of our proposal in one plot, which depicts the MS-SSIM measurement *frame by frame* for one of our trace. It is shown in Figure 7. While this result is only one trace, the behavior that is shown here is typical from our performance evaluation campaign and the following results confirm it.

At the frame 68, both the cross-layer and the FIFO schedulers fail in transmitting a frame, which results in a severe drop of the video quality because this frame contains key information. The cross-layer explicitly decides to stop the transmission of frame 68 to manage to transmit some following frames. This results in a less violent quality degradation. On the contrary, the optimal scheduler anticipates that the frame 68 requires saving a large bandwidth and it drops some frames of minor importance between frame 50 and 68. It thus manages to get the frame 68 on time. The same process happens for frame 108.

Another interesting observation is that the FIFO strategy does not succeed in concealing from the miss of frame 68. However, the cross-layer scheduler manages to get back to a reasonable video quality at the end of the transmission by getting the frame 195 and the followings.

6.3 Traditional MPTCP vs. Optimal

To fairly compare the schedulers (either FIFO or cross-layer) versus the optimal results, we used the same decoding hypothesis as in the optimal model (see Section 4). First, the video unit size is a multiple of the MTU otherwise padding is used. A MPTCP packet thus transports data from only one video unit. Second, a video unit cannot be decoded if all its data are not received in time. Third, the client cannot decode a video unit if all its dependencies are not decoded in time (*i.e.* before their display deadlines).

We first compare the FIFO scheduler to the optimal on the ratio of video units that have been received in time during one transmission. In Figure 8 (respectively Figure 9) we represent the ratio of *frames* (respectively *tiles*) that are received in time, with all its dependencies, for each of the 168 transmissions. The results are shown with a box plot, with the 10^{th}, the 25^{th}, the median, the 75^{th}, and the 90^{th} percentiles. For example the line at the bottom of the box shows the performance of the 16^{th} *worst* transmission over the 168 transmissions. We show the results for four different video lags.

Our main observation is about the variance of the performance for the FIFO scheduler (which, to recall, is the current MPTCP scheduler). For the 10^{th} percentile, less than one quarter of the frames are decoded while all the frames are received for some other transmissions. The variability of the results slightly decreases when the lag increases, but, even for the 2 s lag, the difference between the tenth and the nineteenth percentiles is significant. For the same set of transmissions, the optimal scheduler shows that it is *possible* to schedule the data so that more than three quarters of the frames are well decoded, even for the tenth percentile. **The current MPTCP scheduler is unable to address the cases of unstable throughput for video streaming**. We also note that, for short lags (500 ms and 1 s), more than half of the transmissions are

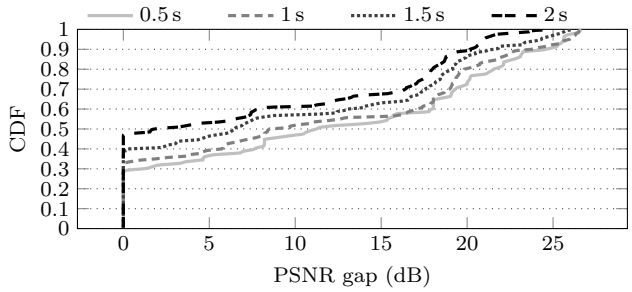

Figure 10: PSNR gap between the optimal scheduler and the FIFO scheduler. Positive gaps means that the PSNR of the optimal is greater than the one of the FIFO. We represent the CDF of the 168 transmissions.

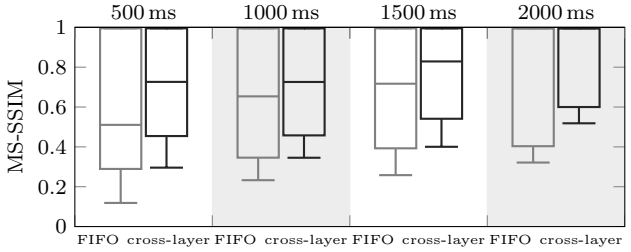

Figure 11: QoE comparison by the mean of MS-SSIM between the cross-layer and the FIFO schedulers

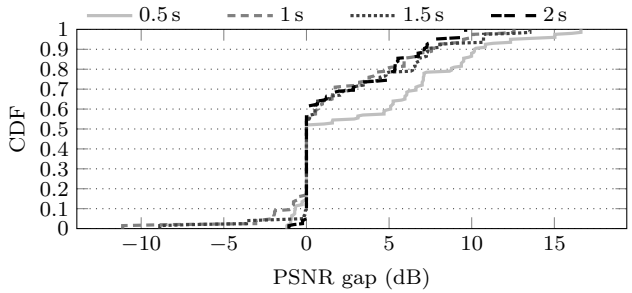

Figure 12: PSNR gap between the cross-layer scheduler and the FIFO scheduler. Positive gaps means that the PSNR of the cross-layer is greater than the one of the FIFO. We represent the CDF of the 168 transmissions.

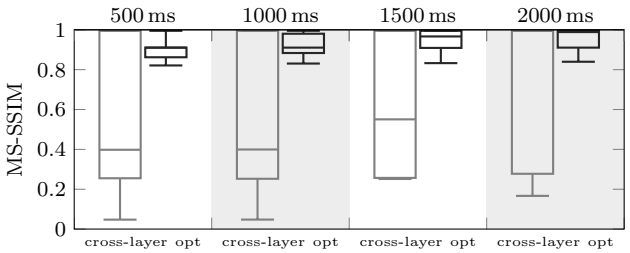

Figure 13: QoE comparison by the mean of MS-SSIM of the cross-layer and the optimal scheduler

disastrous with less than half decoded video units. Finally, since the differences between tiles and frames are not significant, we restrict our studies to the frame level in the remaining of the paper.

Our second observation is the gap between the FIFO scheduler and the optimal one. We measure the PSNR for the 168 transmissions and for both schedulers. We then compare both PSNR, which is a standard video quality process where videos are compared to the same video reference. We show the *PSNR gap* in Figure 10 for the four video lags. For a 500 ms lag (respectively 2 s), both the FIFO and the optimal schedulers perform similarly for one third (respectively one half) of the transmissions. But for the remaining of the transmissions, the PSNR gap is significant. Around half of the transmissions have more than 10 dB of difference with the optimal. The distortion gap can be severe with more than 20 dB. **The margin of progress for the MPTCP scheduler is huge**.

6.4 Cross-Layer Scheduler vs. Traditional MPTCP

We now compare our cross-layer scheduler to the FIFO scheduler. We use both MS-SSIM and PSNR as the main video quality for the four aforementioned video lags. We now relax the constraint that a video unit cannot be decoded if one of its dependencies misses. We let the *libx265* video decoder apply its error concealment strategy.

We present in Figure 11 the box plots for the MS-SSIM measures. Our main observation is that the MS-SSIM of the cross-layer scheduler is from 0.1 to 0.2 bigger than the MS-SSIM of FIFO, for the 10^{th}, the 25^{th} and the median percentiles. It is reflected by the observation that the 16^{th} worst transmission (10^{th} percentile) of the cross-layer is always better than the 42^{th} (25^{th}) worst of the FIFO. The median transmission is also always greater than 0.7 for the cross-layer, which means that our cross-layer scheduler succeeds in guarantying decent delivery in the majority of transmissions.

We represent the PSNR gap in Figure 12. It can happen that the FIFO scheduler results in a better PSNR than the cross-layer. Indeed the decision taken by the cross-layer scheduler can be counter-productive, for example when a video unit is prioritized over other video units but unpredictable packet losses prevents this video unit to be decoded in time. We can see however that these risks are overall positive since such PSNR degradation occurs in less than 15% of the transmissions although PSNR improvement occurs in more than 40% of the transmissions. Furthermore, the gains are bigger for the cross-layer with more than 20% of the transmissions with more than 5 dB.

To sum up, **the cross-layer scheduler manages to offer a better video quality for the client in comparison to the traditional MPTCP scheduler.**

6.5 Cross-Layer Scheduler vs. Optimal

We now compare the cross-layer scheduler to the optimal one, to check whether the cross-layer fills the gap to the optimal. To be fair with the optimal scheduler, we re-consider the constraint that a frame with a missing depending frame cannot be decoded. Hence the MS-SSIM results of the cross-layer scheduler are slightly worse than what shown in Section 6.4.

We use again the box plots to show the MS-SSIM in Figure 13. Our main observation is that **the cross-layer scheduler is far from being as efficient as the**

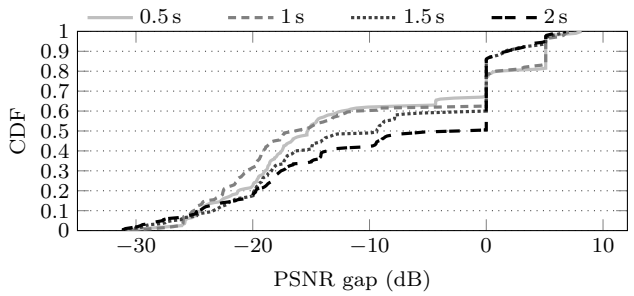

Figure 14: PSNR gap between the optimal scheduler and the cross-layer scheduler. Positive gaps means that the PSNR of the cross-layer is greater than the one of the optimal. We represent the CDF of the 168 transmissions.

optimal scheduler. In this challenging environment, the cross-layer scheduler does not succeed in dealing with the weaknesses of MPTCP and the constraints of the video decoder, although the optimal scheduler manages to find a data scheduling, which guarantees an MS-SSIM greater than 0.8. This observation also applies to the PSNR gap in Figure 14. Please note that, here again, in some rare cases, the optimal scheduler is less efficient than the cross-layer one, because the objective function is not to maximize the PSNR but to maximize the number of decodable video units. However, in two thirds of the transmission, the cross-layer has a larger video distortion with gaps sometimes bigger than 20 dB.

6.6 Overall Conclusive Evaluation

To conclude the evaluation, we provide the average MS-SSIM for all the transmissions and all the video lags (see Table 2). To echo what we previously said, we observe:

- The big gap between the traditional FIFO MPTCP scheduler and the optimal scheduler.
- Our algorithm for the cross-layer scheduler improves the performance of MPTCP regarding video streaming but it is still far from the optimal scheduler.

	0.5 s	1 s	1.5 s	2 s
FIFO	0.583	0.645	0.682	0.746
cross-layer	0.697	0.705	0.749	0.810
optimal	0.892	0.909	0.932	0.944

Table 2: Average MS-SSIM depending on the scheduler strategy and the client lag

7. CONCLUSIVE DISCUSSION

In this paper, we have dealt with the delivery of video streams on multiple paths. We have focused on protocol issues related to application and transport layers. Despite the promises of multi-path networking, the existing solutions have not been widely adopted due to weaknesses of the protocol at these layers. The main idea that we have studied here is the interactions between both layers to enable a better video data scheduling. We have shown that the implementation of such a cross-layer scheduler is possible with reasonable development. To study the potentials of this idea, we have designed a theoretical model, which computes the optimal scheduling solution. This optimal scheduler allows fair performance evaluations. The results presented in Section 6 demonstrate two main statements: (*i*) the gap between the current scheduler and the optimal omniscient scheduler is huge and (*ii*) a relatively simple cross-layer scheduler provides a performance gain in terms of video quality.

This paper hopefully clarifies some of the questions scientists may have about the weaknesses of MPTCP as well as the solutions that can be designed to fix these weaknesses. Many perspectives are still open, and we highlight in the following the main scientific problems we will address in the future.

Better Scheduling Algorithms. The algorithm we have designed for our cross-layer scheduler makes a relatively simple use of the available information. Our goal was to demonstrate that the motivation for a cross-layer scheduler is valid. Now, more sophisticated and efficient algorithms can be designed. As shown in Section 6, our algorithm is still far from being as efficient as the optimal scheduler. In future works, we will study the different information available at each layer to better predict the near future behaviors of the transport layer in order to take better decision regarding the scheduling of the video units.

Implementation and Standardization. The implementation of a cross-layer scheduler requires the development of clear Application Programming Interfaces (APIs). As we noted in this paper, we need at least an API to get a feedback from the transport layer protocol to get a read-only access to key transport-layer information. Similarly, an API should also be defined at the cross-layer scheduler side so that the transport protocol could inform the cross-layer scheduler when a new packet is ready to be transmitted on the network. If the goal is to create a cross-layer scheduler independent of the video application, another API to communicate with the application layer should be defined.

Transport protocols. In this paper we have focused on studying a cross-layer scheduler using a TCP-like transport layer but using an UDP-based protocol is also possible, with potential better results. Indeed, with TCP protocols, the in-order delivery and the guaranty of reception of all packets introduce jitter in the data delivery to the application, especially when coupled in MPTCP. Those two mechanisms do not exist in the UDP transport protocol. A packet loss does not affect other packets and so, jitter in the application delivery exclusively comes from network loss. Following the works done on MPRTP, and considering the recent proposals for UDP-based protocols, like Quick UDP Internet Connections (QUIC), a more efficient multi-path transport protocol can be implemented by leveraging cross-layer interactions.

Client-Server Interactions. This paper studies only the server side of the problem, as if no intelligence existed at the client side. A feedback from the client has the potential to improve the quality of the cross-layer predictions. Moreover, the player can also predict some frames or some portion of frames that would be better to lose depending on the viewer behaviors. For instance, if the viewer focuses only on some part of the video (some faces or objects) then it is

less important to lose information on the part of the video the viewer does not pay attention to. This principle, which has been studied for video compression (for example [31]) can also be applied to cross-layer scheduling and transport protocols.

Acknowledgments

The authors are partially funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

8. REFERENCES

[1] Apple. iOS: multipath TCP support in iOS 7. Blog post, April 2015.

[2] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo. Impact of path characteristics and scheduling policies on MPTCP performance. In *Proc. of IEEE AINA Workshop*, 2014.

[3] Blender's Tears Of Steel. https://mango.blender.org/. Accessed: 2015-11-28.

[4] O. Bonaventure. Experience with multipath TCP. Technical report, IETF Presentation, July 2014.

[5] O. Bonaventure. Multipath TCP through a strange middlebox. Blog post, January 2015.

[6] O. Bonaventure, C. Paasch, and G. Detal. Use cases and operational experience with multipath TCP. Internet-Draft draft-ietf-mptcp-experience-03, IETF, October 2015.

[7] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl. Reducing internet latency: A survey of techniques and their merits. *Communications Surveys Tutorials, IEEE*, PP(99):1–1, 2014.

[8] S. Chen, Z. Yuan, and G. Muntean. An energy-aware multipath-TCP-based content delivery scheme in heterogeneous wireless networks. In *Proc. of IEEE WCNC*, 2013.

[9] Y. Chen, Y. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley. A measurement-based study of multipath TCP performance over wireless networks. In *Proc. of ACM IMC*, 2013.

[10] Y. Chen, D. Towsley, and R. Khalili. MSPlayer: multi-source and multi-path leveraged YoutubER. In *Proc. of ACM CoNEXT*, 2014.

[11] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. WiFi, LTE, or both?: Measuring multi-homed wireless internet performance. In *Proc. of ACM IMC*, 2014.

[12] C. Diop, G. Dugué, C. Chassot, and E. Exposito. Qos-oriented MPTCP extensions for multimedia multi-homed systems. In *Proc. of IEEE AINA Workshop*, 2012.

[13] EPFL's VQMT software. http://mmspg.epfl.ch/vqmt. Accessed: 2015-11-29.

[14] G. Fairhurst, B. Trammell, and M. Kuehlewind. Services provided by IETF transport protocols and congestion control mechanisms. Internet-draft draft-ietf-taps-transports-07, IETF, October 2015.

[15] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, IETF, 2013.

[16] B. Hesmans, F. Duchene, C. Paasch, G. Detal, and O. Bonaventure. Are TCP extensions middlebox-proof? In *Proc. of ACM HotMiddlebox*, 2013.

[17] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *Proc. of ACM IMC*, 2011.

[18] J. R. Iyengar, P. D. Amer, and R. R. Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Trans. Netw.*, 14(5):951–964, 2006.

[19] D. Jurca and P. Frossard. Video packet selection and scheduling for multipath streaming. *IEEE Trans. Multimedia*, 9(3):629–641, 2007.

[20] R. Khalili, N. Gast, M. Popovic, and J. L. Boudec. MPTCP is not pareto-optimal: Performance issues and a possible solution. *IEEE/ACM Trans. Netw.*, 21(5):1651–1665, 2013.

[21] S. S. Krishnan and R. K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. *IEEE/ACM Trans. Netw.*, 21(6):2001–2014, 2013.

[22] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli. DAPS: intelligent delay-aware packet scheduling for multipath transport. In *Proc. of IEEE ICC*, 2014.

[23] P. Likhar, R. S. Yadav, et al. Securing IEEE 802.11 G WLAN Using OpenVPN and Its Impact Analysis. *arXiv preprint arXiv:1201.0428*, 2012.

[24] Y. Lim, Y. Chen, E. M. Nahum, D. Towsley, and K. Lee. Cross-layer path management in multi-path transport protocol for mobile devices. In *Proc. of IEEE INFOCOM*, 2014.

[25] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental evaluation of multipath TCP schedulers. In *Proc. of ACM CSWS*, 2014.

[26] C. Raiciu, M. Handley, and D. Wischik. Coupled congestion control for multipath transport protocols. RFC 6356, IETF, 2011.

[27] 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*, 2012.

[28] V. Singh, S. Ahsan, and J. Ott. MPRTP: Multipath considerations for real-time media. In *Proc. of ACM MMSys*, 2013.

[29] S. Winkler and P. Mohandas. The evolution of video quality measurement: From PSNR to hybrid metrics. *IEEE Trans. Broadcasting*, 54(3):660–668, 2008.

[30] J. Wu, C. Yuen, B. Cheng, M. Wang, and J.-L. Chen. Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks. *IEEE Trans. Mob. Comp.*, PP(99):1–1, 2015.

[31] J. You, T. Ebrahimi, and A. Perkis. Attention driven foveated video quality assessment. *IEEE Trans. Image Proc.*, 23(1):200–213, 2014.