Multimedia Tools and Applications (2019) 78: 7419–7452
https://doi.org/10.1007/s11042-018-6460-0

A scalable WebRTC-based framework for remote video collaboration applications

Stefano Petrangeli, et al. ***[full author details at the end of the article]***

Received: 15 December 2017 / Revised: 8 June 2018 / Accepted: 23 July 2018 /
Published online: 11 August 2018

Abstract
Remote video collaboration is common nowadays in conferencing, telehealth and remote teaching applications. To support these low-latency and interactive use cases, Real-Time Communication (RTC) solutions are generally used. WebRTC is an open-source project for real-time browser-based conferencing, developed with a peer-to-peer architecture in mind. In this peer-to-peer architecture, each sending peer needs to encode a separate, independent stream for each receiving peer participating in the remote session, which makes this approach expensive in terms of encoders and not able to scale well for a large number of users. This paper proposes a WebRTC-compliant framework to solve this scalability issue, without impacting the quality delivered to the remote peers. In the proposed framework, each sending peer is only equipped with a limited number of encoders, much smaller than and independent of the number of receiving peers. Consequently, each encoder transmits to a multitude of receivers at the same time, to improve scalability. A centralized node based on the Selective Forwarding Unit (SFU) principle, called conference controller, forwards the best stream to the receiving peers, based on their bandwidth conditions. Moreover, the conference controller dynamically recomputes the encoding bitrates of the sending peers, to maximize the quality delivered to the receiving peers. This approach allows to closely follow the long-term bandwidth variations of the receivers, even with a limited number of encoders at sender-side, and increase the delivered video quality. An integer linear programming formulation for the bitrate recomputation problem is presented, which can be optimally solved when the number of receivers is small. An approximate, scalable method is also proposed using the K-means clustering algorithm. The gains brought by the proposed framework have been confirmed in both simulation and emulation, through a testbed implementation using the Google Chrome browser and the open-source Jitsi-Videobridge software. Particularly, we focus on a remote collaboration scenario where the interaction among the remote participants is dominated by a single peer, as in a remote teaching scenario. When a single sending peer equipped with three encoders transmits to 28 receiving peers, the proposed framework improves the average received video bitrate up to 15%, compared to a static solution where the encoding bitrates do not change over time. Moreover, the dynamic bitrate recomputation is more efficient than a static association in terms of encoders used at sender-side. For the same configuration mentioned above, the same received bitrate is obtained in the static case using four encoders as in the dynamic case using three encoders.

Keywords Real-time communication · Remote video collaboration · WebRTC · Selective forwarding unit · Integer linear programming · Jitsi-videobridge

1 Introduction

Remote video collaboration is widely used in a variety of applications nowadays [8, 15, 20]. From videoconferencing, to telehealth and remote teaching, it allows to remotely perform tasks that would otherwise require a physical meeting, and it is therefore an important enabler for fast and efficient exchange of information. Remote collaboration communication can be roughly divided in two categories, depending on the type of interaction established among the remote participants. In many-to-many communication, all the remote participants have the same importance and frequently interact among each other. In one-to-many communication instead, the interaction among the participants is usually dominated by a single entity. A classical example of such one-to-many communication is represented by remote teaching applications. In this *virtual classroom*, the students remotely attend a live lecture given by the lecturer. Interactivity is required in this case, as the students can ask questions and actively participate to the discussion. Nevertheless, most of the communication occurs from the lecturer to the students. In both many-to-many and one-to-many scenarios, the remote peers are usually geographically distributed and can experience different bandwidth and network conditions.

Classical streaming techniques as HTTP adaptive streaming are characterized by high latencies, in the order of seconds in the best case [28], and can therefore not guarantee the required degree of interactivity of remote video collaboration applications. On the other hand, remote conferencing solutions can be used. Particularly, the Web Real-Time Communication (WebRTC) framework is an open-source project started by Google in 2011 that provides plugin-free real-time communication capabilities to browser-based applications [18]. Even though this technology guarantees the low-latency and interactivity degree required in remote collaboration, it is affected by another drawback. The WebRTC framework has been developed with a peer-to-peer architecture in mind, where a small group of clients can directly communicate with each other. This approach can suffer from scalability issues when many participants are present at the same time. In standard WebRTC indeed, the peers in communication, or *senders*, would need to encode a separate stream for each receiving peer, the *receivers*. This aspect entails that each receiver is associated with an independent and dedicated encoder at sender-side. This architecture is particularly inefficient in a one-to-many scenario, as in this case, a single peer is usually responsible for the largest part of the communication.

The goal of this paper is to solve the aforementioned scalability problem, without negatively impacting the video quality delivered to the receivers. In the proposed WebRTC-compliant framework, the sender is only equipped with a limited number of encoders, much smaller than and independent of the number of receivers participating in the remote session. This way, the proposed framework improves the scalability of classical WebRTC systems, as each encoder at sender-side can now transmits to a multitude of receivers at the same time. Moreover, instead of keeping the encoding bitrates of the sender fixed to predefined static values, we propose to periodically recompute them based on the changing bandwidth conditions of the receivers. This approach allows to better follow the long-term bandwidth conditions of the receivers and to maximize the delivered quality, even though only a limited number of encoders is actually used at sender-side. This dynamic bitrate recomputation is carried out by a centralized node, called conference controller, which is aware of the bandwidth conditions of the WebRTC receivers. Moreover, the conference controller dynamically forwards to the receivers the best stream at the best bitrate from the sender, in order to accommodate the short-term bandwidth variations of the receivers.

It is worth noting that the goal of this paper is not to optimize the WebRTC protocol itself, which already guarantees the required interactivity of remote collaboration applications, but rather relieving its scalability problems using the conference controller. The controller mainly optimizes the delivery of the real-time streams from sender to receivers, as in a one-to-many scenario most of the communication follows this path. In the WebRTC domain, the conference controller functionalities can be carried out by a Selective Forwarding Unit (SFU), whose task is to receive all the streams and decide which stream should be sent to which participant [10]. Particularly, in this work, the Jitsi-Videobridge software is used as WebRTC SFU.[1]

The main contributions of this paper can be summarized as follows:

1. We present a WebRTC-compliant framework to improve the scalability of classical WebRTC systems. In our framework, each sender is equipped with a limited number of encoders, whose encoding bitrates are dynamically recomputed by the conference controller. Particularly, we model the bitrate recomputation problem as an Integer Linear Programming (ILP) formulation, which is periodically solved by the centralized node. We also propose a fast and scalable algorithm, using the K-means clustering algorithm, to solve the aforementioned problem in an approximate way when the number of receivers is too large.
2. We present an emulation testbed, implemented using state-of-the-art WebRTC software and Jitsi-Videobridge, to evaluate the performance of the proposed framework in a realistic environment. The code has been made available open-source.[2]
3. Detailed results are presented to quantify the gains brought by the proposed approach. Particularly, simulation results are presented to theoretically evaluate the performance of the WebRTC framework in a large number of configurations. The emulation testbed is then used to confirm these results in a realistic setting.

The remainder of this paper is structured as follows. Section 2 presents related work on remote conferencing solutions for WebRTC. A table is also provided to highlight the main differences between the analyzed works, and to clearly point out the advantages and drawbacks of the proposed approach. Section 3 introduces the general architecture of a WebRTC-based communication system, with a particular focus on the bandwidth estimation and congestion control performed by WebRTC endpoints. Particularly, this Section gives a general introduction on the WebRTC standard, its main components and the messages exchanged among the remote peers to estimate the end-to-end bandwidth. The architecture of the proposed framework is described in Section 4, where the applicability of the proposed solution is described for both a many-to-many and one-to-many communication scenario. The functionalities of the conference controller, in terms of dynamic stream forwarding and dynamic bitrate recomputation are presented in detail in Section 5. Particularly, the bitrate recomputation is formulated as an ILP problem, described in Section 5.2. Section 6 details the implementation of the emulation testbed using the Google Chrome browser and the Jitsi-Videobridge SFU. An in-depth analysis of the proposed framework, by means of obtained simulation and emulation results, is presented in Section 7. Simulation results are used to show the gains of the proposed approach in a large number of network scenarios, while emulation allows to validate the results in a realistic setting. Finally, Section 8 concludes the paper.

[1] https://github.com/jitsi/jitsi-videobridge

[2] https://github.com/twauters/WebRTC_dynamic_SFU

2 Related work

Peer-to-peer systems have been widely used for video delivery. As an example, Shehab et al. design a framework for the efficient delivery of Video-On-Demand content in peer-to-peer networks [26]. In this work, the peers are assigned to two different mesh networks, for super peers and regular peers, respectively. Super-peers are the main content distributors, while regular peers mainly consume content. Hossain et al. analyze the problem of privacy and security in distributed video sharing applications, which is extremely relevant for WebRTC systems as well [14]. Alsmirat et al. propose and implement a framework for the bandwidth optimization of distributed, automated video surveillance systems, which share similar characteristics as WebRTC systems in terms of bandwidth variability, latency and geographical distribution of the remote peers [3]. Xu et al. perform a measurement study on real-world conferencing systems [31]. The authors report that a purely peer-to-peer architecture is not popular among these systems, as it does not scale to a large number of users. To improve the scalability of this architecture, an intermediate media server can be used. In WebRTC, this can be achieved using two different components: a Multipoint Conferencing Unit (MCU) or a selective forwarding unit. Table 1 provides a high level summary of the main works available in literature in the domain of WebRTC systems design and optimization.

An MCU receives all the streams from the participants, decodes and composes them in a single common stream that is sent back to the peers. This way, each peer only needs to send and receive a single stream. Two popular MCU implementations are already available, the Janus gateway and Kurento [5, 16]. Janus is conceived as a general purpose gateway that only allows, in its core functionality, to setup a WebRTC communication among the peers [5]. Higher level functionalities are implemented as Janus plugins, as the video MCU plugin, which implements an MCU and supports both many-to-many and one-to-many communication scenarios. Kurento is another example of WebRTC MCU, which also provides advanced functionalities as computer vision and augmented reality [16]. A set of engineered and coherent APIs are also available for the control of the media server, to allow developers to quickly deploy new functionalities [17]. Ma et al. investigate how to improve the encoding/decoding process of an MCU to save bandwidth [19]. The MCU transcodes the sender stream and adjusts it to the viewing conditions of the receiver. The authors consider the viewing distance and the pixel density of the receiver's screen to transcode the stream to an optimal bitrate, in order to save bandwidth. A network-wide system of MCUs is investigated by Granda et al. [9]. The participants are divided into regional clusters, each associated to an MCU. Peers located in different clusters can communicate via the system of MCUs, connected among each other using a peer-to-peer network. This hybrid architecture allows to support a large number of users. Nevertheless, MCU operations are extremely computationally intensive, due to the decoding-mixing-encoding processes that have to be carried out. To reduce this issue, MCU functionalities can be dynamically migrated among conference participants to meet certain bandwidth, latency and CPU constraints [13]. Alternatively, MCU low-level functionalities can be virtualized and deployed on-the-fly [24, 27]. As an example, Rodriguez et al. divide the low-level functionalities of an MCU into independent broadcasters, which can run in distributed environments [23].

Unlike an MCU, an SFU does not require decoding/encoding operations and it is therefore more lightweight. Its main task is to receive all the streams from the participants and selectively forward one or more streams to the peers [29]. When the number of participants is large, the amount of forwarded streams should be limited to avoid wasting bandwidth. For this reason, Grozev et al. develop a speaker identification algorithm to be deployed on

Table 1 Summary of the related work in the domain of WebRTC systems design and optimization

Related work	Type of study	Main approach	Goal of the study
Amirante et al. [5]	Measurement	MCU	Show scalability of the Janus MCU, for a small number of participants (<200)
Lopez et al. [16]	Prototyping	MCU	Present Kurento MCU functionalities and open APIs
Ma et al. [19]	Optimization/ Experiments	MCU	Online video transcoding to increase the bandwidth efficiency of an MCU system
Granda et al. [9]	Prototyping	MCU	Investigate an overlay network of MCUs connecting geographically distributed WebRTC clients
Hossain et al. [13]	Optimization/ Experiments	MCU	Migrate computational and bandwidth intensive MCU functionalities among the remote participants
Rodriguez et al. [23]	Prototyping	MCU	Distribute MCU functionalities among the remote participants, to improve scalability
Grozev et al. [10]	Optimization/ Experiments	SFU	Reduce number of streams forwarded by an SFU to the last N active participants
Xhagjika et al. [30]	Measurement	SFU	Characterization and modeling of the request load on an operational cloud SFU
Grozev et al. [11]	Experiments	SFU	Implement a simulcast scheme to improve the bitrate forwarding efficiency of an SFU
Proposed approach	Optimization/ Experiments	SFU	An SFU to dynamically adapt the encoding bitrates of the encoders at sender-side, based on the bandwidth conditions of the receivers

an SFU, to identify the last N dominant speakers of the conference [10]. To save bandwidth, only these N streams are forwarded to the conference participants. In a measurement study, Xhagjika et al. find that the load pattern on an operational system of SFUs is periodic and can be easily predicted [30]. The prediction can be used to allocate the streams to the right SFU and avoid overloading the system. Similar schemes relying on user behavior modeling and prediction can be successfully applied to improve the efficiency of peer-to-peer systems, as shown by Elhoseny et al. for a Video-On-Demand platform [7]. The software-defined networking principle can be used to optimize a system of distributed SFUs, by dynamically creating a multicast tree for the optimal delivery of the streams [32]. The functionalities of an SFU can be matched with the concept of simulcast in WebRTC [11]. In simulcast, each sending peer encodes the stream at different bitrates, which are then forwarded to the receivers by the SFU. In the work by Grozev et al. [11], each participant can send up to three streams. The SFU forwards the highest quality to participants involved in the conversation, and the lowest quality to the remaining ones. The authors also point out that simulcast is one of the less mature parts of the WebRTC standard, and that it still needs further development and optimizations.

In this work, the conference controller is implemented using the SFU functionalities. Unlike previous works though, the controller not only forwards the streams to the remote peers, but also periodically recomputes the set of encoding bitrates of the sending peer. This bitrate recomputation is modeled as an ILP formulation, which can be optimally solved when the number of receivers is small. Otherwise, an approximate solution is obtained using the K-means clustering algorithm. A preliminary evaluation of the proposed framework has already been presented in previous work [21]. This paper provides a more detailed explanation of the conference controller functionalities, supported by extensive simulation and emulation results to prove its effectiveness.

3 The WebRTC standard

This section presents a general introduction of the WebRTC architecture and how a WebRTC session is established between two peers. Moreover, the bandwidth estimation performed between remote WebRTC endpoints is also briefly discussed. It is worth stressing that the goal of this paper is not to optimize the low-level components of WebRTC described in this section, but rather improving its video delivery architecture to guarantee better scalability, as detailed in Sections 4 and 5.

3.1 WebRTC architecture and session initiation

WebRTC is a collection of communication protocols and APIs that enable real-time communication among remote peers[3] and is currently being standardized by the world wide web consortium and the Internet engineering task force. WebRTC is by design browser-based and does not rely on any external plug-in or other third-party software, which means that it is platform and device independent. Ideally, each device equipped with a browser is capable of initiating a WebRTC session, which makes it an ideal candidate for new interactive streaming applications, as telehealth or remote teaching. WebRTC is not only

[3] We refer to a general WebRTC protocol in this paper by referring to all the functionalities and protocols that are included in WebRTC itself.

limited to video and audio calls, but can also be used for peer-to-peer file sharing and text messaging.

When two or more WebRTC peers want to establish a connection, three types of information have to be exchanged among them. First, session control messages have to be exchanged to initialize (or close) the communication and report errors. Second, network configuration messages are sent to communicate the IP addresses of the remote peers. In WebRTC in fact, the media, be it video, audio or text, is transferred in a peer-to-peer fashion between two or more participants. This aspect entails that the remote peers have to be aware of the respective IP addresses. Third, information related to the media are exchanged among the peers to find the media configuration that can be supported by all participants (e.g., codec and resolution in the case of video). This message exchange is called signaling and is not part of WebRTC itself, but relies on pre-existing protocols (e.g., SIP). All these message exchanges, which take place before the actual communication can start, are carried out with the help of an external server, called the signaling server, which supports the Session Description Protocol (SDP). SDP is used by the remote peers to inform the others about the transport protocol, ports, codecs and relevant parameters to be used during the media transfer.

As the media is exchanged directly among the participants, the protocol has been developed to cope well with firewalls and Network Address Translation (NAT). For this purpose, a Session Traversal Utilities for NAT (STUN) server is used. A STUN server allows NAT clients to find out their public address, the type of NAT they are behind and the port associated by the NAT with a particular port. In most cases, a STUN server is only used by the WebRTC peers during the connection setup, while the actual media is exchanged directly once the session is established. However, when direct media traffic is not allowed (e.g., because of a firewall), a Traversal Using Relays around NAT (TURN) server relays the messages and the media between two or more clients.

3.2 Congestion control and bandwidth estimation in WebRTC

WebRTC uses UDP instead of TCP at the transport layer, as TCP cannot guarantee the low latency required in real-time communication, since its main focus is on reliability. In WebRTC, the congestion control mechanism is implemented at the application layer using the Real-Time Transport Protocol (RTP) and its control protocol RTCP. The congestion control mechanism tries to estimate the capacity of the channel connecting two remote WebRTC peers. This way, the sending peer can adjust the video encoding bitrate to the available bandwidth of the receiving peer. In contrast with classical video streaming techniques based on HTTP adaptive streaming, in WebRTC the sending peer directly controls the rate at which the video is sent.

The congestion control algorithm currently implemented in WebRTC is the Google Congestion Control (GCC) [6], which attempts to detect congestion before it actually occurs, by using the inter-arrival delay of consecutive packets. Particularly, the congestion control is divided in two separate parts: a delay-based controller located at the receiving peer, and a packet loss-based controller located at the sending peer. The receiver analyzes the inter-packet delay and generates an estimation of the available bandwidth. This report is sent back to the sender with an RTCP message called Receiver Estimated Maximum Bitrate (REMB), usually every 250 to 500 ms. To decide the final value of the encoding video bitrate, the sender also uses the latest packet loss feedback, which is reported by the receiver in particular RTCP messages. The system is designed to slowly increase the estimated bandwidth and the video rate as long as no congestion is detected and to ensure that the available

bandwidth of the channel is eventually matched. As soon as congestion is detected, the estimated bandwidth is decreased.

This approach gives a particular and characteristic evolution to the end-to-end estimated bandwidth in WebRTC. An example of such evolution is presented in Fig. 1, which reports the result of an experiment with two WebRTC peers in communication among each other. In the experiment, a sending peer *A* is connected to a receiving peer *B* via a network link, whose available bandwidth is shaped as depicted in Fig. 1 (blue full line). The actual bandwidth estimated by peer *A* (orange dashed line), which drives the encoding bitrate of the video sent to peer *B*, slowly follows the available bandwidth and results in an exponential increase followed by sudden drops when the bandwidth decreases.

Congestion control algorithms in WebRTC are developed assuming that sending and receiving peers directly communicate and exchange media traffic among each other. In the proposed framework instead, this assumption is not true anymore, as the conference controller behaves as the endpoint for both the sender and the receivers. The impact on the bandwidth estimation of the receivers of this modification is discussed in Section 5.3.

4 Architecture of the proposed WebRTC framework

In this section, we describe the architecture of the proposed framework in a general remote collaboration scenario. The WebRTC framework presented in this paper is proposed to optimize the delivery of real-time WebRTC streams in the context of remote video collaboration applications. Particularly, a centralized node called the conference controller is used to guarantee cost-efficiency in terms of the number of encoders used at sender-side and scalability when the number of remote peers is large. Consequently, the framework does not aim to optimize the WebRTC protocol described in Section 3, but rather to improve its delivery architecture.

The high-level architecture of the proposed framework is presented in Fig. 2a. In a general remote collaboration applications, several peers participate in the same remote session and are in communication among each other via the conference controller, which is positioned between them. Each peer is equipped with a limited number of encoders, usually much smaller than the number of participants. We assume that at any given point in time a set of peers is transmitting to a set of receiving peers. It is worth noting that, given the interactive nature of this communication, the role of sender and receiver is not static but can dynamically vary among the peers.

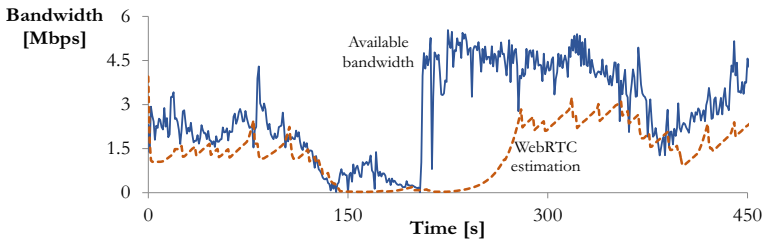
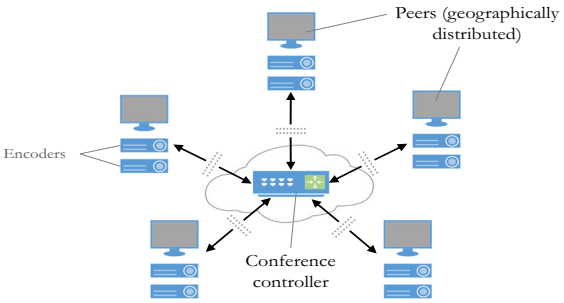
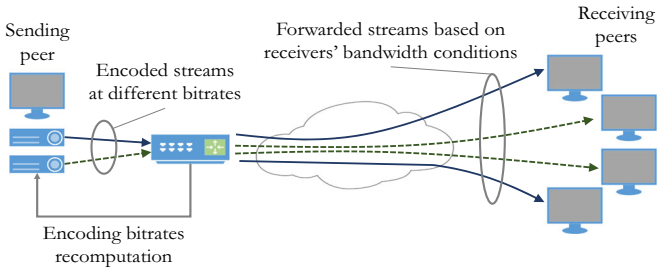


Fig. 1 Illustrative example of the bandwidth estimation evolution in WebRTC. The WebRTC estimated bandwidth (orange dashed line) slowly follows the actual available bandwidth (blue full line), and is characterized by exponential increases and sudden decreases



(a) The proposed framework is composed of several peers, each equipped with a number of encoders. The communication among the peers occurs through the conference controller, located between them (e.g., in the cloud). The peers are geographically distributed and can experience different bandwidth conditions.



(b) *From sender to receivers:* the conference controller behaves as the terminal endpoint for both the sender and the receivers, and performs the dynamic stream forwarding and dynamic bitrate recomputation tasks.

Fig. 2 General architecture of the proposed WebRTC framework (**a**), together with the high-level description of the tasks performaned by the conference controller when optimizing the communication from the sending to the receiving peers (**b**)

The goal of the controller is to optimize the delivery of the WebRTC streams from senders to receivers, as described in Fig. 2b (for the sake of simplicity, the figure only depicts one sender). At sender-side, the remote event is captured and encoded in different streams at different bitrates, which are sent to the conference controller. The controller is then responsible for the dynamic forwarding of the encoded streams to all the receivers participating in the session, based on their bandwidth conditions. This approach allows to relieve the peer-to-peer architecture of classical WebRTC and improve the scalability of the system. Indeed, each encoder at sender-side is now associated with multiple receivers, with the number of receivers usually much larger than the number of encoders. To maximize the rate delivered to the receivers, the video bitrate of the encoders is not static, but is periodically recomputed by the controller to better follow the bandwidth conditions of the receivers. The functionalities carried out by the conference controller are detailed in Section 5.

The proposed framework can be applied in any remote collaboration scenario. In the remainder of this paper though, we relax this condition and consider a one-to-many scenario only, typical in a virtual classroom, where one participant, called the *sender* is responsible for most of the communication towards the remaining peers, called *receivers*. In this case, the conference controller is located at the sender-side premises, so that enough bandwidth

is always available between sender and controller. Interactivity is still envisioned in this case, as the receivers can communicate among each other or with the sender (e.g., asking questions in a virtual classroom). Particularly, we will focus on the downstream side of the problem (e.g., from sender to receivers), as most of the communication follows this path. The operations performed by the conference controller in this case are detailed in the next section. It is nevertheless implied that the proposed framework can guarantee the required level of interactivity, by allowing the receivers to participate in the communication.

5 WebRTC conference controller design

The most important component of the proposed WebRTC framework is the conference controller, which performs two main tasks. First, the controller receives all the encoded streams from the sender and dynamically forwards them to the receivers, based on their available bandwidth (Fig. 2b). Second, it periodically recomputes the encoding bitrates of the sender to better follow the long-term network variations of the receivers. In the remainder of this section, we detail the operations performed by the conference controller in terms of dynamic stream forwarding (Section 5.1) and encoding bitrate recomputation (Section 5.2). Moreover, we discuss in Section 5.3 the bandwidth probing mechanism employed by the conference controller to make a better estimation of the receivers' bandwidth.

5.1 Dynamic stream forwarding

The conference controller acts as an endpoint for the sender and for the receivers, by receiving the encoded streams from the sender and forwarding them to the receivers. Particularly, the controller forwards the highest sustainable stream to each receiver, based on their estimated bandwidth. As explained in Section 3, in a classic peer-to-peer WebRTC architecture, a receiver reports statistics and feedback to the corresponding sender, which allow to estimate the end-to-end bandwidth. In the proposed framework, the conference controller acts as the actual sender for the receivers. This aspect entails that the controller can intercept the REMB messages, which are used to estimate the bandwidth of the remote receivers [4]. Each time an REMB message is received by the controller, the corresponding bandwidth estimation is updated and a new stream is selected for the receiver, if needed. REMB messages are usually generated by the receivers every 250 to 500 ms. Consequently, the dynamic forwarding is performed at a very fine-grained timescale, which allows to accommodate the short-term bandwidth variations of the receivers and guarantee a continuous playback.

5.2 Encoding bitrate recomputation

A second, more long-term optimization is performed by the conference controller to recompute the set of encoding bitrates at sender-side. In the proposed framework, each encoder at sender-side transmits to multiple receivers at the same time. In order to maximize the video rate received by the receivers, the encoded rate should be as close as possible to the actual bandwidth of the receivers. Static, fixed encoding bitrates are suboptimal, as the receivers' conditions can change over time, especially in wireless environments, where the available bandwidth can highly fluctuate. By allowing the encoding bitrates to dynamically vary, it is possible to follow the long-term bandwidth variations of the receivers and, therefore, maximize the delivered video quality.

We formulate the dynamic bitrate recomputation problem as an ILP formulation, which is executed every T_{opt} seconds by the conference controller. At time t when the recomputation takes place, the virtual classroom is composed of R receivers, each associated with a bandwidth measure b_r. Different options are possible on how to compute b_r for a given time window of multiple seconds, as for example the average or the minimum estimated bandwidth over the period $\left[t - T_{opt}; t\right]$. In Section 7, we identify the best method to compute the bandwidth measure b_r from the individual bandwidth estimations of the receiver in the interval $\left[t - T_{opt}; t\right]$. The main sender of the virtual classroom is equipped with l_{max} encoders (with $l_{max} \ll R$), which can encode the video in the range $[B_{min}; B_{max}]$, where B_{min} and B_{max} are the minimum and maximum encoding rates, respectively. We indicate with L the number of possible encoding levels in this interval, each associated to a rate B_l. The goal of the controller is to select the l_{max} encoding levels, among the possible L levels, which are the closest to the bandwidth measures b_r of the receivers. The complete ILP formulation is as follows:

$$
\begin{aligned}
&\min_{\alpha} \sum_{r=1}^{R}\sum_{l=1}^{L} \alpha_{r,l}\,(b_r - B_l)^2 \\
&\text{s.t.} \quad \alpha_{r,l} \in \{0,1\} \qquad \forall r \in \{1,\ldots,R\}, l \in \{1,\ldots,L\} \\
&\qquad \beta_l \in \{0,1\} \qquad \forall l \in \{1,\ldots,L\} \\
&\qquad \beta_l \geq \alpha_{r,l} \qquad \forall r \in \{1,\ldots,R\}, l \in \{1,\ldots,L\} \\
&\qquad \sum_{l=1}^{L} \alpha_{r,l} = 1 \qquad \forall r \in \{1,\ldots,R\} \\
&\qquad \sum_{l=1}^{L} \beta_l \leq l_{max} \\
&\qquad \beta_0 = B_{min} \\
&\qquad \sum_{l=1}^{L} \alpha_{r,l} B_l \leq b_r \qquad \forall r \in \{1,\ldots,R\}
\end{aligned}
\tag{1}
$$

The solution of the problem is characterized by two sets of boolean decision variables, namely $\alpha_{r,l}$ and β_l. $\alpha_{r,l}$ is equal to 1 when client r is associated to encoding level l, and 0 otherwise. Similarly, β_l is equal to 1 when encoding level l is selected for one of the encoders, and 0 otherwise. The optimization problem is designed to find the l_{max} encoding levels whose bitrates allow to minimize the quadratic difference with the receivers' bandwidth measures. The first three constraints of the ILP formulation set up a consistent relation between the decision variables α and β. The fourth constraint indicates that each receiver can only be associated with one specific encoding level. The last three constraints are representative of the analyzed problem. First, only l_{max} encoding levels can be selected out of the L available levels (constraint 5), as l_{max} indicates the number of encoders available at sender-side. Second, we always select the lowest possible encoding bitrate as a solution (constraint 6). This way, we guarantee the receivers can always play the lowest available quality and avoid playout interruptions. Third, the encoding bitrate associated to receiver r must be lower than the bandwidth measure for r (constraint 7), in order to guarantee a continuous playout.

The ILP formulation presented in (1) uses an objective, video-agnostic objective function in the video rate domain, namely the quadratic difference between the possible encoding levels and the bandwidth measures of the receivers. The objective function can be easily

modified to perform the optimization in the Peak-Signal-to-Noise-Ratio (PSNR) domain instead, in order to select the l_{max} encoding levels that allow to maximize the actual video quality of the receivers, as shown in (2):

$$\min_{\alpha} \sum_{r=1}^{R} \sum_{l=1}^{L} \alpha_{r,l} \left(PSNR\left(b_r\right) - PSNR\left(B_l\right)\right)^2 \tag{2}$$

The constraints of this ILP formulation are the same as those presented in (1). The main drawback of this approach is that the function mapping the video rate to the actual PSNR has to be estimated online by the conference controller. In Section 7, we show how the formulation in the PSNR domain can indeed be beneficial for the receivers' video quality, using a pre-computed PSNR curve.

Even though the presented ILP formulation can provide the optimal solution to the analyzed problem, it can suffer from scalability issues when the solution space, which depends on the number of receivers R and possible encoding levels L, is large. In order to be effective, the optimal solution should be computed in a fraction of the optimization period T_{opt}, which cannot be guaranteed when the solution space is too large. For this reason, we propose to use the K-means clustering algorithm to solve the bitrate recomputation problem in an approximate but highly scalable way. Despite its simplicity, the K-means method has proven to be very effective in a large variety of real-life problems, both in terms of performance and speed [1, 33]. In this case, the inputs of the algorithm are the bandwidth measures b_r of the receivers, which have to be clustered into l_{max} separate clusters. Once the clusters are generated, the smallest value associated to each cluster is chosen as encoding bitrate for the l_{max} encoders. As for the ILP formulation, also in this case the lowest available encoding level is always selected, to guarantee a continuous playback. In Section 7, we evaluate both the ILP formulation and K-means algorithm in terms of system performance and computing time.

As a final consideration, it is worth noting that the two tasks carried out by the conference controller, namely dynamic forwarding and bitrate recomputation, are performed at different timescales. The bitrate computation presented in (1) is executed on a timescale of seconds, and is used to adjust the encoding bitrates to take into account long-term variations of the receivers' network conditions. On the contrary, the stream forwarding is executed on a timescale of milliseconds, to closely follow the changing bandwidth conditions of the receivers.

5.3 Bandwidth probing

Another task carried out by the conference controller is the estimation of the receivers' bandwidth, which can be carried out using REMB messages (see Section 5.1) and probing. As explained in Section 3, in a standard peer-to-peer WebRTC architecture, the sender decides the encoding bitrate of the video sent to the receiver. The encoding bitrate is decided based on the available bandwidth towards the receiver, which is estimated using a congestion control algorithm. The encoding rate is slowly increased to detect the capacity of the channel. When the maximum channel capacity is about to be reached, congestion is detected by the receiver, which communicates this feedback to the sender. This mechanism allows to discover the actual capacity of the channel connecting the sender to the receiver, independently of the actual implemented congestion control algorithm. This process assumes that the sender and the receiver are in direct communication between each other, so that each perturbation of the encoded bitrate at the sender is reflected in the congestion experienced by the receiver.

Unfortunately, this assumption does not hold anymore when the conference controller is introduced. Indeed, the controller acts as an endpoint for both the sender and the receivers, which are no longer in direct communication with each other. Moreover, the encoding bitrates of the video are decided by the controller itself, which computes them solving the optimization problem presented in the previous section. In standard WebRTC, the video traffic is used by the endpoints to estimate the capacity of the channel, while this condition is not valid anymore when the conference controller is introduced. To relieve this issue, the conference controller can send additional *dummy* probing traffic towards the receivers, to improve the channel capacity estimation. The probing mechanism has to be designed to balance two opposite objectives. First, the probing traffic should not be too aggressive to avoid congesting the video itself. Second, it should be responsive enough to detect the actual channel capacity of the receivers. Each receiver in the remote session is associated with a separate and independent probing instance at the conference controller. This aspect allows to follow the bandwidth variations of each receiver, independently of each other.

In the remainder of this section, we detail the probing mechanism developed for the conference controller. It is worth mentioning that, due to the continuous ongoing development of the WebRTC standard, the proposed approach represents an initial attempt for bandwidth probing at an SFU, whose performance and limitations are discussed in Section 7.2. As an example, the Jitsi-Videobridge software already provides low-level probing functionalities.[4] Moreover, probing can be considered as an additional tool used by the conference controller. The main tasks performed by the controller, namely stream forwarding and bitrate recomputation, are completely independent from the implemented probing mechanism, and can be executed even if probing is not enabled or is not needed.

The amount of probing traffic $P(-)$ to be sent towards the receivers is re-computed every K_{prob} seconds by the conference controller. Particularly, at time instant k, an initial, tentative amount of probing traffic, indicated with $p(k)$, is computed as in (3):

$$p(k) = \begin{cases} \lambda(k) \times vr(k) & \text{if } bw(k) < bw(k - K_{prob}) \\ (1 + \lambda(k)) \times P(k - K_{prob}) & \text{otherwise} \end{cases} \tag{3}$$

$vr(k)$ is the rate of the video sent to the receiver at time k, $bw(k)$ is the estimated bandwidth and $P(k - K_{prob})$ is the probing traffic sent at time $k - K_{prob}$. When the estimated bandwidth of the receiver drops (first condition in (3)), the probing traffic is reset to a fraction of the video rate sent to the receiver. This sudden decrease avoids that the probing traffic congests the channel and impairs the actual video stream. Otherwise, the probing traffic is tentatively increased (second condition in (3)), following a multiplicative increase driven by parameter $\lambda(k)$. This parameter is not fixed to a static value, but depends on the video rate $vr(k)$. Particularly, $\lambda(k)$ is higher when $vr(k)$ is lower, and vice-versa. When the video rate is low, more probing traffic is needed to detect the channel capacity. Consequently, λ becomes larger to make probing more aggressive. On the contrary, λ decreases when vr is high, thus making probing more conservative and avoiding congesting the channel. The evolution of the λ parameter is presented in (4):

$$\lambda(k) = c + m \times vr(k) \qquad \text{with} \qquad m = -\frac{\lambda_{max}}{B_{max}}, \quad c = \lambda_{max} - m \times B_{min} \tag{4}$$

where B_{min} and B_{max} are the minimum and maximum encoding bitrates, and λ_{max} is the highest possible value for the λ parameter. $\lambda(k)$ decreases linearly as the video rate $vr(k)$

[4] https://github.com/jitsi/jitsi-videobridge/blob/master/src/main/java/org/jitsi/videobridge/cc/BandwidthProbing.java

increases, ranging between λ_{max} and $\lambda_{max} \times (B_{min}/B_{max})$, when $vr(k)$ is equal to B_{min} and B_{max}, respectively. As a final step, the actual amount of probing traffic $P(k)$ is limited in case it could congest the channel in the next interval $[k; k + K_{prob}]$:

$$P(k) = \begin{cases} \eta \times (bw(k) - vr(k)) & \text{if } p(k) + vr(k) \geq bw(k) \\ p(k) & \text{otherwise} \end{cases} \quad (5)$$

When the total amount of data sent to the receiver (i.e., $p(k) + vr(k)$) is higher than the current estimated bandwidth $bw(k)$, probing should be limited in order to avoiding congesting the channel. Therefore, the actual final probing $P(k)$ is set to a fraction of the remaining estimated channel capacity $bw(k) - vr(k)$, via the parameter η. A small η results in a more conservative but slower probing behavior, while a large value could result in congestion that could impair the video stream itself. Otherwise, the amount of probing traffic is simply set to the value $p(k)$ computed in (3).

6 Implementation details

In order to evaluate the performance of the proposed solution in a realistic environment, the framework is implemented on the imec iLab.t Virtual Wall emulation platform.[5] To implement the receivers and the sender, the Google Chrome browser is used. Nothing is changed of the Google's original WebRTC stack, which makes our solution completely WebRTC-compliant. From an implementation perspective, the sender is decoupled into l_{max} WebRTC sub-senders, each encoding the video stream at a different bitrate. To implement the conference controller, the Jitsi software[6] is used, a set of open-source projects to build and deploy secure videoconferencing solutions. Particularly, the Jitsi-Videobridge,[7] a WebRTC SFU, has been used as the main component. Its default functionality is to relay all the streams generated by the conference to all the participants. Jitsi-Meet,[8] a JavaScript application running on top of the browser WebRTC stack, is used at the sub-senders and receivers to interface with the Jitsi-Videobridge. In the remaining of this section, we explain the modifications we made on the Jitsi-Videobridge to implement the stream forwarding and bitrate recomputation functionalities.

6.1 Stream forwarding selection

By default, the Jitsi-Videobridge forwards multiple streams to a participant. The stream of the participant who is currently speaking, the so-called *dominant speaker*, is automatically detected by the software and is always included in these streams. We override this logic so that a different dominant speaker can be manually set per receiver. Moreover, we limit the amount of streams that can be sent to a specific receiver to only one, selected as previously explained. Using this mechanism, the Jitsi-Videobridge dynamically assigns a sub-sender per receiver, so that the encoding bitrate is lower than the receiver's estimated bandwidth.

To implement this functionality, the conference controller has to estimate the available bandwidth of the receivers first. This estimation is performed by default by the

[5] http://doc.ilabt.iminds.be/ilabt-documentation/virtualwallfacility.html
[6] https://jitsi.org/
[7] https://github.com/jitsi/jitsi-videobridge
[8] https://github.com/jitsi/jitsi-meet

Jitsi-Videobridge, which forwards all WebRTC traffic among the conference participants, implemented using the RTP/RTCP protocol suite. In WebRTC, bandwidth estimation is performed using RTCP REMB messages, a feedback used by a receiver to notify its media stream sender over the same RTP session of the estimated available bandwidth on the path to the receiving side. The Jitsi-Videobridge intercepts these RTCP REMB messages and is therefore aware of the available bandwidth of the receivers.

6.2 Dynamic encoding bitrate recomputation

As the long-term network conditions of the receivers can change over time, it is required to periodically recompute the set of encoding bitrates of the sub-senders, as described in Section 5. Once these values are computed, they have to be enforced on the WebRTC sub-senders. However, there is no standardized way to set the encoding bitrate of a WebRTC client. To perform this task, we use the RTCP REMB messages, which contain the receiver's estimated available bandwidth. In WebRTC, the congestion control mechanism of a sender considers this estimation as the maximum bitrate that can be sent to a receiver. Consequently, the sender's encoder uses this value as its current target bitrate. Once the new bitrates are computed, the Jitsi-Videobridge modifies the REMB feedback messages for the sub-senders by setting the newly computed bitrate instead of the bandwidth estimation of the receivers. This way, the sub-senders are forced to modify their encoding bitrates. To implement this mechanism, we changed the way RTCP messages are generated in libjitsi,[9] the underlying Java media library used by Jitsi-Videobridge. Instead of setting the maximum bitrates in the REMB messages for the sub-senders to the latest estimated remote bandwidth, we set them to the bitrates generated by the bitrate recomputation presented in Section 5.

7 Performance evaluation results

In this section, we perform a detailed analysis of the performance of the proposed framework. In Section 7.1, we first present a Java-based WebRTC simulator that allows to thoroughly test the theoretical performance of the system under a large number of configurations in terms of number of receivers, maximum number of encoders, bitrate recomputation period and employed algorithm (ILP or clustering). In Section 7.2, we then use the emulation testbed presented in the previous section to confirm the simulation results in a realistic environment and to highlight the main differences between simulation and emulation.

7.1 Simulation results

The Java-based simulator presented in this section has been developed to mainly test the performance of the bitrate recomputation presented in Section 5.2. Consequently, only the functionalities of the WebRTC receivers and the conference controller are implemented in the simulator, while no actual video encoding, dynamic forwarding or bandwidth probing is performed. This simplification allows us to isolate the gains of the proposed bitrate recomputation. A more realistic evaluation is instead performed in the next section using the emulation setup presented in Section 6. The role of the simulated WebRTC receivers is

[9] https://github.com/jitsi/libjitsi

to provide input on the bandwidth measurements that are periodically fed to the simulated conference controller, which performs the bitrate recomputation. The ILP formulation presented in Section 5.2 is solved using the CPLEX software,[10] while the K-means algorithm has been implemented using the Weka library [12].

Algorithm 1 WebRTC-like bandwidth evolution used in the Java-based simulator.

Require: t, current simulation time, in seconds
$ban(t)$, actual available bandwidth for the simulated WebRTC receiver at time t, in kbps
$webRTCBan(t-1)$, WebRTC-like receiver bandwidth estimation at the previous time step $t-1$, in kbps
Ensure: $webRTCBan(t)$, WebRTC-like receiver bandwidth estimation at time t, in kbps

1: **if** $t - dropTime \leq 15$ **then**
2: $webRTCBan(t) = (1 + 0.016) \times webRTCBan(t-1)$
3: **else**
4: $webRTCBan(t) = (1 + 0.075) \times webRTCBan(t-1)$
5: **end if**
6: **if** $webRTCBan(t) > ban(t)$ **then**
7: $webRTCBan(t) = ban(t)$
8: $dropTime = t$
9: **end if**
10: **return** $\min(webRTCBan(t), 30)$

In order to generate a realistic behavior of the WebRTC receivers, we empirically model the characteristic pattern of the bandwidth estimated by the WebRTC receiver, discussed in Section 3 and shown in Fig. 1. This modeling, presented in Algorithm 1, allows to obtain a WebRTC-like bandwidth evolution in the Java-based simulator. Given the actual available bandwidth at time t, the WebRTC bandwidth evolves following a multiplicative increase (lines 2 and 4). The multiplicative factor depends on the time elapsed since the last detected bandwidth drop. Particularly, the bandwidth tends to increase slowly during the few seconds following the drop (line 1), and faster afterwards. Moreover, the WebRTC simulated bandwidth cannot drop below 30 kbps (line 10), which is the minimum value returned by the bandwidth estimator implemented by libjitsi.[11] Figure 3 reports the evolution of the simulated WebRTC bandwidth presented in Algorithm 1, for the same illustrative bandwidth pattern presented in Fig. 1. The simulated bandwidth closely resembles the actual estimated bandwidth. It is worth stressing that this algorithm is only used in the simulator to obtain realistic values for the bandwidth estimations of the WebRTC receivers and is not intended to provide an exact modeling of the WebRTC bandwidth estimation algorithm. The constant values reported in Algorithm 1 have been set after fine tuning the algorithm on a set of bandwidth traces collected on a real 3G network [22].

Using the described simulator, a large-scale evaluation of the parameters of the bitrate recomputation problem is carried out, which is presented in Table 2. In total, 1890 different configurations have been evaluated. To analyze the approach under realistic network

[10] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[11] https://github.com/jitsi/libjitsi/blob/master/src/org/jitsi/impl/neomedia/rtp/sendsidebandwidthestimation/BandwidthEstimatorImpl.java

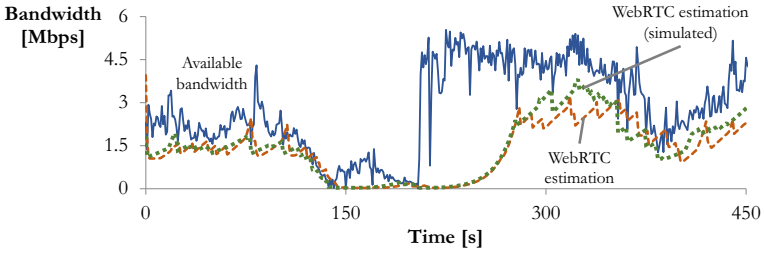


Fig. 3 WebRTC bandwidth estimation (orange dashed line) slowly follows the available bandwidth (blue full line). The WebRTC-like bandwidth evolution presented in Algorithm 1 (green dotted line) closely follows the real WebRTC bandwidth

conditions, we apply a different bandwidth pattern, collected on a real 3G network [22], on each simulated WebRTC receiver. Each experiment configuration has been repeated fifteen times to guarantee statistical significance, each iteration lasting 240 seconds. The minimum (B_{min}) and maximum (B_{max}) encoding rates are fixed to 50 and 2500 kbps. Forty possible encoding levels (L) are available, with bitrates equally spaced in the encoding rate interval. We tested several values for the number of encoders at sender-side (l_{max}), ranging from 2 to 10, and number of receivers (R), from 5 to 80. This choice allows to test the framework both when $R \simeq l_{max}$ and when $R \gg l_{max}$. We compare the performance of the proposed framework to a solution where the encoding bitrates are statically fixed and are not dynamically recomputed during the experiment. When the optimization takes place in the video rate domain, the static bitrates are equidistantly assigned to $50 + l \times (2500 - 50)/(l_{max} - 1)$ kbps, with $l = 0, 1, \ldots, l_{max} - 1$. In the PSNR domain, the l_{max} static bitrates are assigned to be equidistant in terms of PSNR values in the interval [50; 2500] kbps. We generated a PSNR-like curve to be used in the simulator, which follows a logarithmic function of the video rate, as proposed by Schroeder et al. [25]:

$$PSNR = 3.136 \times log(rate) + 18.297 \tag{6}$$

The generated PSNR values range between 30.56 dB and 42.77 dB for a video rate of 50 kbps and 2500 kbps, respectively. This approach allows to show the difference in performance between the optimization carried out in the video rate or PSNR domain. In the ILP and K-means cases, the recomputation period (T_{opt}) ranges from 2 to 16 seconds. A shorter

Table 2 Overview of the evaluated parameter configuration in the Java-based simulator, resulting in a total of 1890 different configurations

Parameter	Value
$[B_{min}; B_{max}]$	[50;2500] kbps
L	40
l_{max}	2, 3, 4, 5, 6, 8, 10
R	5, 10, 20, 40, 80
Bitrate recomputation	Static, ILP, K-means
Optimization domain	Rate, PSNR
T_{opt}	2, 4, 8, 16 s
b_r computation method	Latest, Minimum, Average

period is desirable, as it allows to closely follow the network conditions of the receivers, but it is computationally expensive when the solution space (defined by the number of receivers and possible encoding levels) is large. We also tested three different methods to compute the bandwidth measure b_r used in the optimization problem in Section 5.2. Assuming the optimization takes place at time t and BW indicates the vector containing all the N receiver bandwidth estimations in the interval $[t - T_{opt}; t]$, b_r is calculated as follows:

$$\begin{aligned} \text{Minimum: } b_r &= \min(BW) \qquad \text{Average: } b_r = \frac{1}{N} \times \sum_i BW(i) \\ \text{Latest: } b_r &= BW(N) \end{aligned} \tag{7}$$

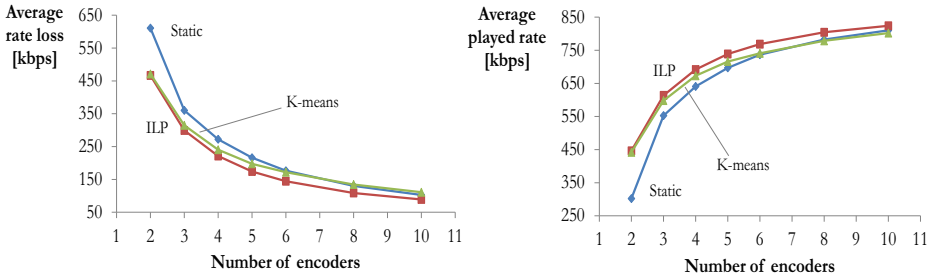
representing the minimum bandwidth estimation in the period $[t - T_{opt}; t]$, the average bandwidth estimation or the latest one, respectively.

For each receiver, we keep track of three metrics. First, the average video rate received during the experiment. Second, the average *rate loss*, computed as the difference between the available bandwidth at the receiver and the actual received rate. The rate loss represents the gap between the rate a receiver is able to sustain (i.e., the available bandwidth) and the rate actually received. Third, the average PSNR value achieved by the receiver, using the above mentioned equation. It is worth noting that the first two metrics are objective and video-independent. On the other hand, the PSNR results depends on the employed bitrate to PSNR curve, which is modeled as in (6). To investigate the impact of the parameters presented in Table 2, we compute for each of the 1890 experiment configurations the average rate loss, played rate and PSNR over the whole group of clients and over the fifteen different bandwidth configurations. When analyzing the impact of one of the parameters listed in Table 2 on the performance of the system, we keep the remaining parameters fixed to reference values that we consider realistic for the virtual classroom scenario. Particularly, we consider as reference scenario the case with $l_{max} = 4$, $R = 20$, $T_{opt} = 8$s and b_r computed as the latest available bandwidth estimation. This approach is used to better highlight the impact of each single parameter on the proposed framework. Nonetheless, very similar conclusions can be drawn when altering the reference scenario.

7.1.1 Optimization in the video rate domain

In this section, we report the results when the optimization carried out by the conference controller takes place in the video rate domain (problem formulation presented in (1)). The goal of this section is to clearly highlight when the optimization carried out by the proposed framework can effectively improve the video rate, compared to a static association of the encoding bitrates. Particularly, this analysis is carried out in terms of number of encoders available at sender-side, optimization period of the dynamic recomputation, bandwidth computation method and number of receivers.

Figure 4 reports the average rate loss and played rate as a function of the number of available encoders at sender-side. Increasing the number of encoders reduces the rate loss and increases the played rate, independently of the used bitrate recomputation algorithm. When more encoders are used, it is possible to follow in a more fine-grained way the bandwidth variations of the receivers. Ideally, when each receiver is associated to a dedicated encoder, as in classic peer-to-peer WebRTC, the rate loss would drop to zero and the played rate would be maximized. The optimal ILP formulation, solved using the CPLEX software, clearly provides the best results. The gains are particularly evident when few encoders are used. When three encoders are available, the proposed dynamic bitrate



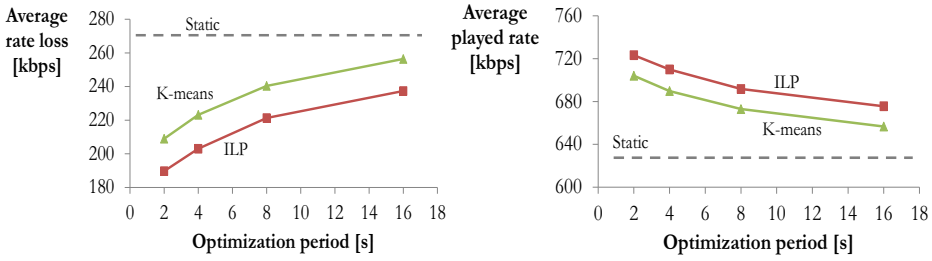
(a) Rate loss. (b) Played rate.

Fig. 4 Increasing the number of available encoders reduces the rate loss (**a**) and increases the played rate (**b**). The proposed approach outperforms a static one, and allows to achieve similar performance using less encoders

recomputation results in 17% reduced rate loss and 11% increased played rate, compared to a static approach. Moreover, our solution is more efficient in terms of the number of encoders, as similar performance can be reached using fewer encoders. For instance, the same rate loss and played rate is obtained in the static case using four encoders as in the dynamic case using three encoders. The K-means clustering approach reaches similar results, while being more scalable than the ILP formulation. In terms of rate loss, the K-means approach is less than 10% worse than the optimal formulation, when the number of encoders is small. When the number of encoders increases, the sub-optimal clustering algorithm tends to reach similar performance as the static case.

Another important parameter of the conference controller is the bitrate recomputation period, whose impact is shown in Fig. 5. As expected, a longer optimization period degrades the performance of the system, both when the optimal ILP formulation and the clustering approach is used. The relevance of the encoding bitrates tends to decrease over time, as the long-term bandwidth conditions of the receivers might change. A shorter period can reduce this side effect, but it is more computationally expensive, especially when the number of receivers is large.

The ILP formulation and clustering algorithm presented in Section 5.2 take as inputs a general bandwidth measure b_r for each receiver. This bandwidth measure is a function



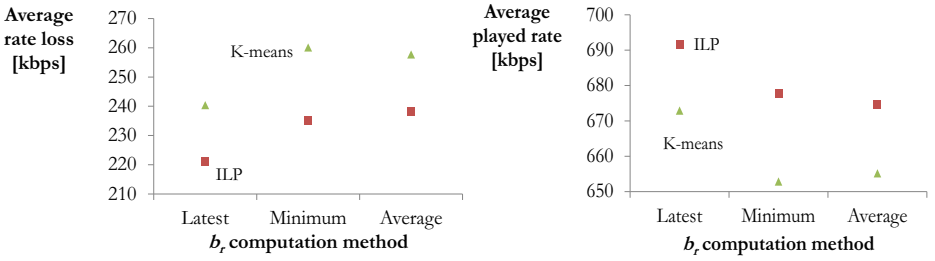
(a) Rate loss. (b) Played rate.

Fig. 5 Increasing the bitrate recomputation period has a negative effect on the performance of the system, as it is more difficult to follow the bandwidth variations of the receivers. Nevertheless, a longer optimization period reduces the computational complexity of the bitrate recomputation

of the bandwidth estimations of a particular receiver in the period $[t - T_{opt}; t]$ and can be computed as described in (7). Particularly, we compute b_r as the minimum or average bandwidth estimation in the period $[t - T_{opt}; t]$ or as the latest available bandwidth estimation. Figure 6 reports the rate loss and played rate, for both the ILP formulation and K-means algorithm, when the different bandwidth measure methods are used. In both the ILP and K-means cases, using the latest available bandwidth estimation as input for the bitrate recomputation yields the best results. The bandwidth estimation of WebRTC is not instantaneous but takes into account previous estimations as well. Moreover, the bandwidth evolution is rather slow in order to avoid congesting the video channel. This behavior is clearly visible in the exponential evolution captured by Algorithm 1 and visible in Fig. 3. When b_r is computed as the average bandwidth estimation over the entire optimization period, old, non-relevant values are also included. Conversely, the latest bandwidth estimation best represents the actual bandwidth conditions of the receivers.

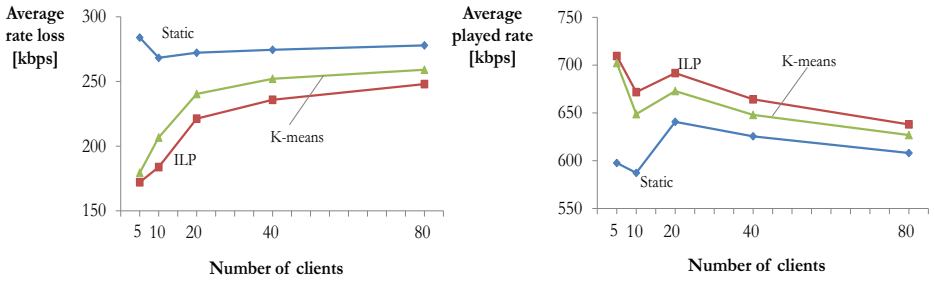
To conclude this analysis, we investigate whether the number of receivers has any influence on the performance of the system (Fig. 7). Interestingly, the relative gain of the dynamic bitrate recomputation decreases with the number of receivers. For ten receivers, the rate loss is decreased by 31% when using the ILP formulation. This value decreases to 10% when eighty receivers are simulated. When the number of receivers is large, it becomes more difficult to recompute the bitrates in order to follow the bandwidth evolution of each individual receiver. As an example, Fig. 8 reports the encoding bitrates evolution over time, for ten and eighty clients, with four available encoders. In the eighty receivers case, the recomputed bitrates are more stable than in the ten receivers case. In other words, when the number of receivers is large, the encoding bitrates tend to change less, resulting in a behavior that is more similar to a static association. In terms of scalability, even in the 80 receivers case, the optimal solution using the ILP formulation can always be found in less than 100 ms, when the experiments are carried out on a 2x Hexacore Intel E5645 (2.4GHz) CPU machine with 24GB RAM, running Ubuntu 16.04. A more in-depth analysis on the scalability of the proposed approach is reported in Section 7.1.4.

In conclusions, the proposed bitrate recomputation is more efficient than a static association of the encoding bitrates when the number of encoders at sender-side is small and when the optimization period is short. When few encoders are available at sender-side, a static association is particularly inefficient, as the different rates are usually very far from each other. Keeping a short optimization period, on the other side, allows to better follow the bandwidth conditions of the receivers, as expected. In terms of number of receivers, the



(a) Rate loss. (b) Played rate.

Fig. 6 The best performance is reached when the bandwidth measure b_r, the input of the bitrate recomputation problem presented in Section 5.2, is computed as the latest bandwidth estimation of the receivers in the period $[t - T_{opt}; t]$



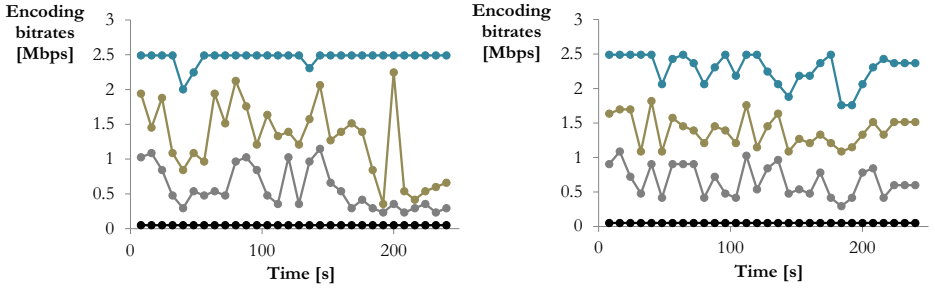
(a) Rate loss. (b) Played rate.

Fig. 7 The relative gain of the proposed approach compared to a static one tends to decrease as the number of receivers increases

relative gain of the proposed solution tends to decrease with the number of receivers, as it becomes more difficult to follow the individual bandwidth conditions of the receivers.

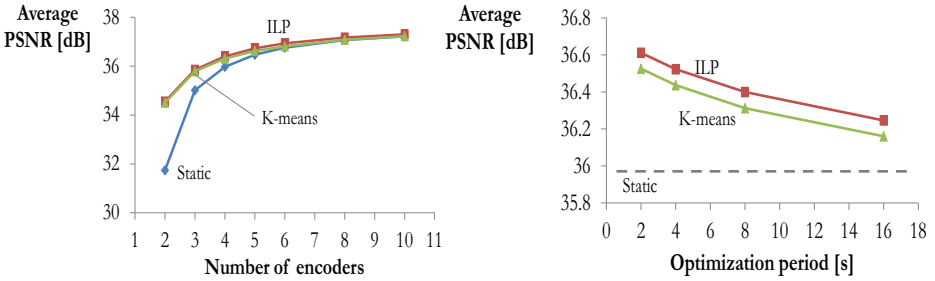
7.1.2 Optimization in the PSNR domain

In this section, we briefly present the results when the bitrate recomputation is carried out in the PSNR domain ((2) and (6)). In this scenario, the goal of the bitrate recomputation is to increase the actual quality of the video delivered to the receivers, rather than simply increasing the received rate. This aspect is important to consider, as the correlation between rate and video quality is not linear. Therefore, it is possible to obtain a better allocation of the encoding resources in this case. As for the optimization in the video rate domain, we consider as reference scenario the case with $l_{max} = 4$, $R = 20$, $T_{opt} = 8$s and b_r computed as the latest available bandwidth estimation. Figure 9a shows the evolution of the PSNR as a function of the number of encoders. Also in this case, increasing the number of encoders allows to increase the achieved PSNR. Compared to Fig. 4b, the gain of the proposed approach is higher when few encoders are used, but decreases faster as the number of encoders increases. The optimization period plays a similar role as in Section 7.1.1. A longer optimization period results in poorer performance, as the encoding bitrates might not be representative anymore of the varying bandwidth conditions of the receivers. Similar results as those reported in the previous section are obtained for the b_r computation methods



(a) Encoding bitrates – 10 clients. (b) Encoding bitrates – 80 clients.

Fig. 8 As the number of receivers increases, the dynamic recomputation results in more stable and less variable encoding bitrates



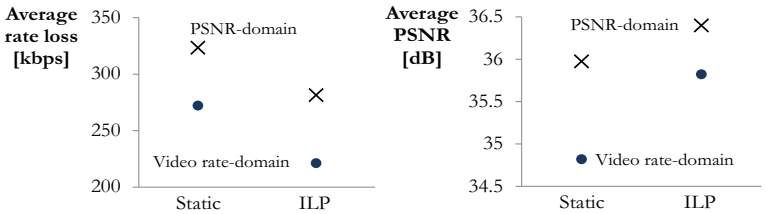
(a) Influence of the number of encoders. (b) Influence of the optimization period.

Fig. 9 Evolution of the PSNR as a function of the number of encoders (**a**) and the optimization period (**b**)

and the influence of the number of clients. For space reasons, these results are therefore omitted.

7.1.3 Comparison between video rate and PSNR domain optimization

The goal of this section is to highlight the difference in performance arising when the bitrate recomputation is carried out in the video rate or PSNR domain. For this reason, we selected a particular configuration with twenty receivers and four available encoders, and report the average rate loss and average PSNR obtained using a static approach and the ILP formulation. In this last case, the optimization period is fixed to eight seconds and b_r is computed as the latest available bandwidth estimations of the receivers. Figure 10a reports the rate loss for the static and ILP approaches. When the optimization is carried out in the video rate domain, the goal is to reduce the rate loss itself (see objective function in (1)). Therefore, the rate loss is lower compared to the optimization in the PSNR domain, for both approaches. A similar conclusion can be drawn for the PSNR. The improvement in terms of PSNR in the static case clearly shows how the bitrate allocation changes when the objective is to maximize the video quality itself, rather than the achieved video rate. Even though the rate loss increases by 18%, the PSNR itself increases by more than 1 dB. It is worth stressing though that the PSNR results depend on the streamed video, while the rate loss is an independent and objective metric. Moreover, we assume in this work that the PSNR curve is pre-computed and available at the conference controller. In reality, the PSNR curve is not available and has to be estimated online by the conference controller.



(a) Rate loss. (b) PSNR.

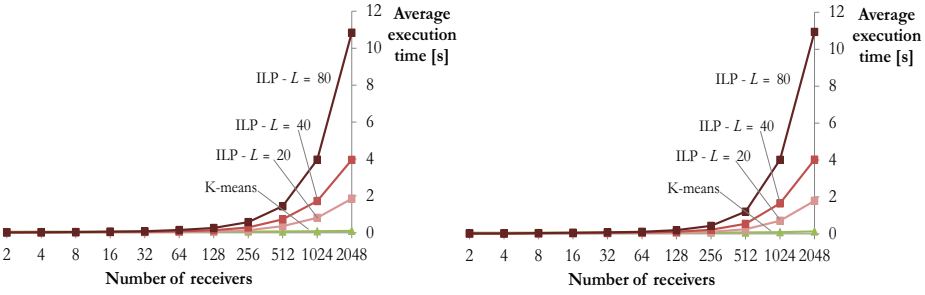
Fig. 10 The rate loss increases when the optimization is carried out in the PSNR domain. Similarly, the PSNR decreases when the optimization takes place in the video rate domain

7.1.4 Scalability analysis

Besides being able to provide good performance in terms of rate loss and played rate, the proposed recomputation algorithms have to be able to compute the new set of encoding bitrates in a fraction of the optimization period. This requirement is necessary to guarantee that the new set of bitrates is still representative of the receivers' bandwidth conditions. In order to test the scalability of the ILP formulation and K-means algorithm, we perform a series of experiments with increasing number of clients, from 2 to 2048, and number of encoders, equal to 3 and 12. In the ILP case, we also test the influence of the parameter L, which indicates the number of possible encoding levels in the $[B_{min}; B_{max}]$ interval (see Section 5.2 and Table 2). The solution space of the ILP formulation depends on the number of receivers R and the number of possible encoding levels L, among which the l_{max} final encoding bitrates are selected. The results of this analysis are presented in Fig. 11. All the experiments are carried out on a 2x Hexacore Intel E5645 (2.4GHz) CPU machine with 24GB RAM, running Ubuntu 16.04. Independently of the number of encoders and the number of possible encoding levels L, the ILP formulation can scale well up to 128 clients, providing a solution in less than 260 ms in the worst case scenario. This aspect entails that in a virtual classroom scenario, where the number of students is usually below 100, an optimal solution can always be computed. Depending on the value of L, more or less receivers can be supported using the ILP formulation. When $L = 20$ for instance, the ILP can still scale in the 512 clients case. Nevertheless, reducing this value too much can negatively affect the performance of the system. The number of possible encoding levels should indeed provide a fine-grained discretization of the encoding interval $[B_{min}; B_{max}]$, among which the ILP formulation can choose the encoding bitrates. The K-means clustering algorithm is instead able to scale extremely well, even for 2048 receivers, and provide the new encoding bitrates in less than 130 ms, in all cases. In light of the above, the ILP formulation can be used in nominal conditions, when the number of receivers is small, while the K-means algorithm can be employed when a large crowd of students attends the remote teaching session.

7.2 Emulation results

In this section, we investigate the performance of the proposed framework in a realistic virtual classroom scenario, using the emulation testbed presented in Section 6, which allows to test all the functionalities of the conference controller (namely dynamic forwarding, bitrate



(a)Three encoders. (b) Twelve encoders.

Fig. 11 The proposed ILP formulation can scale well up to 128 receivers. On the contrary, the K-means algorithm can always provide a solution in less than 130 ms, even when the number of receivers is high

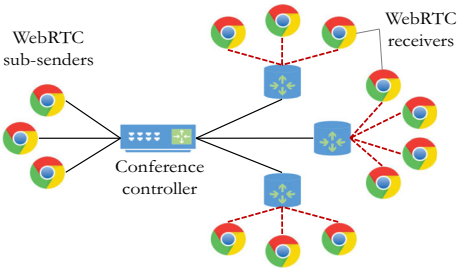


Fig. 12 The emulated setup on the imec iLab.t Virtual Wall is composed of several WebRTC sub-senders and receivers implemented using the Google Chrome browser and one conference controller implemented using the Jitsi-Videobridge (Section 6)

recomputation and probing) with real software components. In the testbed, each WebRTC receiver is a separate physical machine running a Google Chrome browser, connected to the conference controller via a set of routers (see Fig. 12). A differente bandwidth trace is applied on the red dashed links connecting the routers to the receivers, while the black full links are over-provisioned. The single sender is split into several WebRTC sub-senders directly connected to the controller.

As in Section 7.1, we evaluate the system under different configuration settings, summarized in Table 3. The number of encoder (l_{max}) varies between 3 and 5, while the number of receivers (R) is equal to 10 or 28, which well represent a typical virtual classroom scenario. Besides the static association, only the ILP formulation is tested. As shown in Section 7.1.4, an optimal solution can be found in less than 260 ms when the number of clients is less than 128. Therefore, we only present results for the optimal recomputation. The optimization takes place in the video rate domain only. The other parameters are similar to those presented in Table 2, to allow for an easy comparison between simulation and emulation. The latest available bandwidth estimation has been chosen as computation method for b_r. In Section 7.1.1, we showed that this choice leads to the best results. For consistency, the same 3G bandwidth traces used in the simulation experiments are also used in the emulation testbed. This aspect entails that the simulated and emulated WebRTC receivers are applied with the same bandwidth patterns, to guarantee consistency. Each experiment configuration has been repeated fifteen times to guarantee statistical significance, each iteration lasting 240 seconds. No real video capture is carried out by the encoders. Instead, a predefined video was used as input for the encoders.[12]

For each emulated iteration, we compute the average rate loss and played rate over the entire group of receivers, and present the average results over the fifteen emulated iterations. Each point of the graphs is also associated with the 10% and 90% quantiles over the fifteen iterations.

7.2.1 Bandwidth probing effect

We start our analysis by first investigating the impact of the bandwidth probing algorithm, presented in Section 5.3, on the performance of the proposed framework. The conference controller uses the proposed probing algorithm to send additional data on the channel towards the receivers and therefore improve the end-to-end bandwidth estimation. This aspect is extremely important for an accurate execution of the bitrate recomputation and stream forwarding tasks. We select a configuration with 28 receivers and the static encoding bitrate allocation. This choice allows to isolate the benefits of probing as opposed to those

[12] https://media.xiph.org/video/derf/y4m/factory_1080p30.y4m

Table 3 Overview of evaluated parameter configuration in the emulation testbed, resulting in a total of 24 different configurations

Parameter	Value
$[B_{min}; B_{max}]$	[50;2500] kbps
L	40
l_{max}	3, 4, 5
R	10, 28
Bitrate recomputation	Static, ILP
Optimization domain	Rate
T_{opt}	4, 8, 16 s
b_r computation method	Latest

due to the dynamic recomputation. The number of encoders varies between 3 and 5. When probing is enabled, the parameters λ_{max} and η (see Section 5.3) are fixed to 0.4 and 0.875, respectively. Preliminary results showed that these values provide the best probing performance. The results of this analysis are presented in Fig. 13. When probing is enabled, the average played rate increases by 2.6 and 1.9 times in the three and five encoders case, respectively. When probing is disabled, the receivers and the conference controller, which acts as an end-point for the receivers in our framework, can only rely on the video traffic itself to perform the bandwidth estimation, which is not sufficient to correctly estimate the capacity of the channel. For this reason, the controller has to directly perform the bandwidth probing task that in a standard WebRTC architecture would be performed by the senders themselves. The bandwidth probing procedure presented in Section 5.3 can effectively improve the estimation of the receivers' bandwidth, which consequently results in an increased video rate delivered to the receivers. In light of these results, the remaining experiments in this section have all been carried out by enabling bandwidth probing at the conference controller.

These results clearly show that the introduction of the conference controller can have a negative impact on the performance of the system if probing is not used. Even though preliminary, the algorithm presented in Section 5.3 can effectively improve the bandwidth estimation at the conference controller and guarantee that the receivers obtain the most suitable stream depending on their network conditions.

7.2.2 Algorithm comparison

In this section, we investigate the actual gains of the proposed framework using emulation. This analysis allows to show the gains of the proposed framework in a realistic setting, as the sender, conference controller and receivers are implemented using state-of-the-art WebRTC software, and communicate among each other using an emulated network (Fig. 12). The

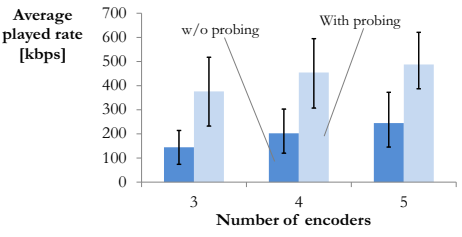
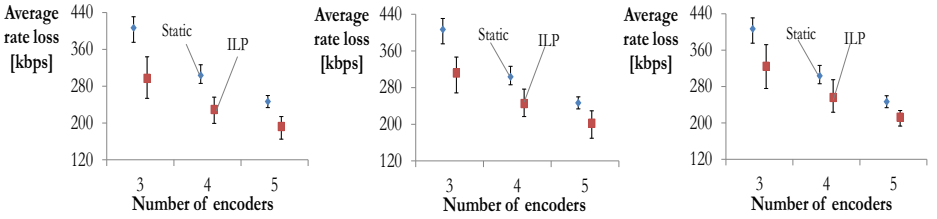


Fig. 13 In emulation, probing allows to more than double the played rate, compared to a scenario where probing is disabled



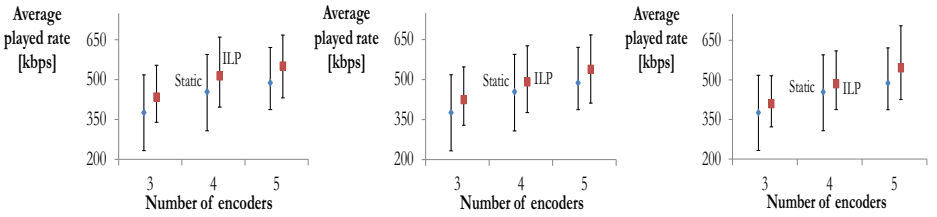
(a) $T_{opt} = 4$ s. (b) $T_{opt} = 8$ s. (c) $T_{opt} = 16$ s.

Fig. 14 Impact of the optimization on the rate loss of the different bitrate recomputation approaches. The ILP formulation provides the best results. The difference between a dynamic and static association decreases when the optimization period increases

results of the 24 different configurations summarized in Table 3 are reported in Figs. 14 and 15. For the sake of brevity, we only report the results for the experiments with 28 receivers. Very similar results are obtained in the case with 10 receivers, which we therefore omit. Overall, the rate loss decreases by 20% when the ILP formulation is used. The gains are higher for a low number of encoders and when the optimization period is shorter. For instance, for the ILP formulation, the average rate loss improves by 23%, 19% and 17% when three, four and five encoders are available, respectively. These results clearly show the benefits of the proposed framework. In terms of played rate, the average gain when using the ILP formulation is about 11%. The results presented in this section thoroughly confirm the trends and analyses performed using the Java-based simulator and the gains brought by the proposed framework. In the next section, we highlight the main difference between emulation and simulation and the optimizations that can further improve the performance of the proposed framework.

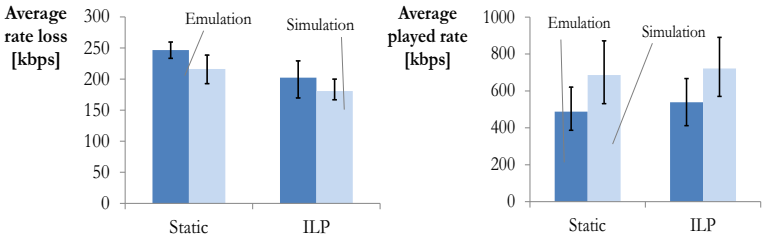
7.2.3 Comparison between emulation and simulation

This section investigates the difference in performance obtained between simulation and emulation, and allows to point out an important aspect of the proposed framework that would need further refining and improvement in future work, namely bandwidth probing at the conference controller. In order to compare the results obtained with the Java-based simulator and the emulation testbed, we select a configuration with 28 receivers, five available encoders and eight seconds optimization period, for the ILP formulation. The same



(a) $T_{opt} = 4$ s. (b) $T_{opt} = 8$ s. (c) $T_{opt} = 16$ s.

Fig. 15 Impact of the optimization period on the played rate on the different bitrate recomputation approaches. As expected, the rate decreases as the optimization period increases, with the ILP formulation reaching the best performance



(a) Rate loss. (b) Played rate.

Fig. 16 The performance reached by the proposed framework on the emulation testbed is worse compared to that obtained with the Java-based simulator, both in terms of average rate loss (**a**) and played rate (**b**)

experiment configuration was repeated in both simulation and emulation. The correspondent results are presented in Fig. 16. Compared to simulation, the rate loss increases by about 25 kbps for all the different approaches, while the average played rate decreases by almost 180 kbps in the emulation testbed. Nonetheless, the relative performance among the different approaches is very similar in both simulation and emulation. This difference is mainly due to the way the receivers' bandwidth is estimated in simulation and emulation. The WebRTC-like bandwidth estimation presented in Algorithm 1 is modeled after the real bandwidth estimation pattern of a classical WebRTC streaming, where the sender can probe the available bandwidth of the receiver by altering the encoding bitrate of the video itself. In emulation instead, probing traffic is artificially generated by the controller using the procedure explained in Section 5.3. Although effective, as shown in Section 7.2.1, the receivers' bandwidth estimation performed at the conference controller in emulation evolves more slowly than in simulation and is on average lower. This aspect is directly reflected on the average rate loss and played rate of the receivers. This analysis points out the need for an efficient probing algorithm at the conference controller, which can further improve the end-to-end bandwidth estimation of the receivers. This estimation is crucial in the proposed framework, as it is the input for both the stream forwarding and the dynamic bitrate recomputation performed by the conference controller.

8 Conclusions

In this paper, a framework has been proposed for the efficient delivery of WebRTC streams in the context of remote video collaboration applications. Classical streaming techniques as HTTP adaptive streaming cannot guarantee the low latency and interactivity required in these scenarios. Consequently, the open-source browser-based WebRTC protocol has been used instead. In order to overcome the scalability issues of a classical WebRTC peer-to-peer architecture, only a few encoders are used at sender-side, where each encoder transmits to several WebRTC receivers at the same time. This choice allows to scale the proposed approach to a large number of receivers and reduce the encoding costs. A conference controller, implemented using the Jitsi-Videobridge software, dynamically forwards the most suitable stream to each receiver, to accommodate fast bandwidth variations and ensure a continuous playout. Besides this short-term adaptation, the controller periodically recomputes the set of encoding bitrates to better follow the long-term network conditions of the receivers. The optimal solution to the bitrate recomputation problem has been found using an ILP formulation when the number of receivers is small, and in an approximate but

fast way using the K-means clustering algorithm. Both simulation and emulation results confirm the gains brought by the proposed approach, which was tested in a one-to-many remote collaboration scenario, typical in virtual classrooms, for example. In an emulated scenario with 28 receivers and one sender equipped with three encoders, the proposed framework can increase the delivered video rate up to 15%, compared to a static, fixed association of the encoding bitrates. Moreover, the dynamic recomputation is more efficient than a static approach, as less encoders are needed to obtain similar performance. For the same configuration mentioned above, similar results in terms of rate loss and played rate are obtained using a static association and four encoders at sender-side and the proposed dynamic recomputation with three encoders.

Future work will focus on further improving the performance of the proposed framework and overcome its limitations. First, we will explore more complex clustering algorithms for the dynamic recomputation task, which can leverage the computational power of modern GPUs [2]. Another important area of improvement of the proposed framework is the design of a better probing mechanism compared to that proposed in Section 5.3. As shown in Sections 7.2.1 and 7.2.3, the effect of probing on the performance of the systems should be carefully investigated. Third, the proposed framework and bitrate recomputation formulation should be expanded to include forward error correction, in order to guarantee a reliable data stream among the remote peers. Moreover, in Section 7.1.3, we show how the optimization in the video rate and PSNR domain can differ among each other. As the ultimate goal of the system is to improve the video quality obtained by the receivers, a PSNR optimization should be preferred. This aspect entails that the conference controller would need to estimate online the relationship between rate and PSNR, in order to perform the optimization proposed in (2).

Acknowledgments Jeroen van der Hooft is funded by grant of the Agency for Innovation by Science and Technology in Flanders (VLAIO). This research was performed partially within the imec PRO-FLOW project (150223).

References

1. Abdeldaim AM, Sahlol AT, Elhoseny M, Hassanien AE (2018) Computer-aided acute lymphoblastic leukemia diagnosis system based on image analysis. Springer International Publishing
2. Alsmirat MA, Jararweh Y, Al-Ayyoub M, Shehab MA, Gupta B (2017) Accelerating compute intensive medical imaging segmentation algorithms using hybrid cpu-gpu implementations. Multimed Tools Appl 76(3):3537–3555
3. Alsmirat MA, Jararweh Y, Obaidat I, Gupta B (2017) Automated wireless video surveillance: an evaluation framework. J Real-Time Image Proc 13(3):527–546
4. Alvestrand H (2013) RTCP message for receiver estimated maximum bitrate internet-draft draft-alvestrand-rmcat-remb-03 (work in progress)
5. Amirante A, Castaldi T, Miniero L, Romano SP (2015) Performance analysis of the janus webRTC gateway. In: Proceedings of the 1st workshop on all-web real-time systems, AWeS '15. ACM, New York, pp 4:1–4:7
6. Carlucci G, De Cicco L, Holmer S, Mascolo S (2016) Analysis and design of the Google congestion control for web real-time communication (WebRTC). In: Proceedings of the 7th international conference on multimedia systems, MMSys '16. ACM, New York, pp 13:1–13:12

7. Elhoseny M, Shehab A, Osman L (2018) An empirical analysis of user behavior for p2p iptv workloads. In: The International conference on advanced machine learning technologies and applications (AMLTA2018). Springer International Publishing, pp 252–263
8. de Paiva Guimarães M, Dias D, Mota J, Gnecco B, Durelli V, Trevelin L (2016) Immersive and interactive virtual reality applications based on 3D web browsers. Multimedia Tools and Applications. https://doi.org/10.1007/s11042-016-4256-7
9. Granda JC, Nuño P, Suárez FJ, García DF (2015) Overlay network based on webRTC for interactive multimedia communications. In: 2015 International Conference on computer, information and telecommunication systems (CITS), pp 1–5
10. Grozev B, Marinov L, Singh V, Ivov E (2015) Last N: relevance-based selectivity for forwarding video in multimedia conferences. In: Proceedings of the 25th ACM workshop on network and operating systems support for digital audio and video. ACM, pp 19–24
11. Grozev B, Politis G, Ivov E, Noel T, Singh V (2017) Experimental evaluation of simulcast for WebRTC. IEEE Commun Standards Mag 1(2):52–59
12. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD Explor Newsl 11(1):10–18
13. Hossain MA, Khan JI (2016) Distributed dynamic MCU for video conferencing in peer-to-peer network. In: 2016 IEEE 35th International performance computing and communications conference (IPCCC), pp 1–8
14. Hossain MS, Muhammad G, Abdul W, Song B, Gupta B (2018) Cloud-assisted secure video transmission and sharing framework for smart cities. Futur Gener Comput Syst 83:596–606
15. Jang-Jaccard J, Nepal S, Celler B, Yan B (2016) WebRTC-based video conferencing service for telehealth. Computing 98(1):169–193
16. López L, París M, Carot S, García B, Gallego M, Gortázar F, Benítez R, Santos JA, Fernández D, Gracia VRT, López FJ (2016) Kurento: the WebRTC modular media server. In: Proceedings of the 2016 ACM on multimedia conference, mm '16. ACM, New York, pp 1187–1191
17. López-Fernández L, García B, Gallego M, Gortázar F (2017) Designing and evaluating the usability of an API for real-time multimedia services in the internet. Multimed Tools Appl 76(12):14,247–14,304
18. Loreto S, Romano SP (2017) How far are we from WebRTC-1.0? An update on standards and a look at what's next. IEEE Commun Mag 55(7):200–207
19. Ma L, Veer D, Chen W, Sternberg G, Reznik YA, Neff RA (2015) User adaptive transcoding for video teleconferencing. In: 2015 IEEE International conference on image processing (ICIP), pp 2209–2213
20. Oh H, Ahn S, Choi J, Yang J (2015) WebRTC based remote collaborative online learning platform. In: Proceedings of the 1st workshop on all-web real-time systems, AWeS '15. ACM, New York, pp 9:1–9:5
21. Petrangeli S, Pauwels D, van der Hooft J, Slowack J, Wauters T, Slowack J, De Turck F (2018) Improving quality and scalability of WebRTC video collaboration applications. In: Proceedings of the 9th ACM on multimedia systems conference, MMSys'18
22. Riiser H, Endestad T, Vigmostad P, Griwodz C, Halvorsen P (2012) Video streaming using a location-based bandwidth-lookup service for bitrate planning. ACM Trans Multimed Comput Commun Appl 8(3):24:1–24:19
23. Rodríguez P, Alonso A, Salvachúa J, Cervino J (2014) dOTM: a mechanism for distributing centralized multi-party video conferencing in the cloud. In: 2014 International Conference on future internet of things and cloud, pp 61–67
24. Rodríguez P, Alonso Á, Salvachúa J, Cerviño J (2016) Materialising a new architecture for a distributed MCU in the cloud. Comput Standards Interf 44(Supplement C):234–242
25. Schroeder D, Essaili AE, Steinbach E, Staehle D, Shehada M (2013) Low-complexity no-reference PSNR estimation for H.264/AVC encoded video. In: 2013 20th International packet video workshop, pp 1–6
26. Shehab A, Elhoseny M, El Aziz MA, Hassanien AE (2018) Efficient schemes for playout latency reduction in P2P-VoD systems. Springer International Publishing, pp 477–495
27. Trnkoczy J, Pašcinski U, Gec S, Stankovski V (2017) SWITCH-ing from multi-tenant to event-driven videoconferencing services. In:2017 IEEE 2nd International workshops on foundations and applications of self* systems (FAS*W), pp 219–226
28. van der Hooft J, Petrangeli S, Wauters T, Huysegems R, Bostoen T, De Turck F (2017) An HTTP/2 push-based approach for low-latency live streaming with super-short segments. J Netw Syst Manag, 1–28
29. Wenzel M, Meinel C (2016) Full-body webRTC video conferencing in a web-based real-time collaboration system. In: 2016 IEEE 20th International conference on computer supported cooperative work in design (CSCWD), pp 334–339

30. Xhagjika V, Escoda D, Navarro L, Vlassov V (2017) Load and video performance patterns of a cloud based WebRTC architecture. In: 2017 17th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID), pp 739–744
31. Xu Y, Yu C, Li J, Liu Y (2014) Video telephony for end-consumers: measurement study of Google+, iChat, and Skype. IEEE/ACM Trans Network 22(3):826–839
32. Yang Ez, Zhang Lk, Yao Z, Yang J (2016) A video conferencing system based on SDN-enabled SVC multicast. Front Inf Technol Electron Eng 17(7):672–681
33. Yuan X, Li D, Mohapatra D, Elhoseny M (2017) Automatic removal of complex shadows from indoor videos using transfer learning and dynamic thresholding. Computers and Electrical Engineering

Stefano Petrangeli obtained a master degree in Systems Engineering from University of Rome "La Sapienza", Italy, in December 2011. In February 2012, he joined Telecom Italia Laboratories and Polytechnic of Turin for a second level master in Networks and Services Innovation for the ICT Sector. He was a winner of an annual scholarship granted by Telecom Italia. In March 2013 he started a Ph.D. in Quality of Experience Management of Advanced Multimedia Services at Ghent University, supervised by Prof. Filip De Turck. Particularly, the focus of his research is on the end-to-end Quality of Experience optimization of internet video streaming delivery, ranging from in-network-based solutions to the design of adaptive client-based algorithms. He is author or co-author of more than 20 papers in the field of multimedia systems and delivery.

Dries Pauwels received his master's degree in industrial engineering from the University of Ghent in February 2016. In April of that year, he joined the Department of Information Technology at University of Ghent, where he is currently working as a software developer in the IBCN-IDLab group.

Jeroen van der Hooft obtained his M.Sc. degree in Computer Science Engineering from Ghent University, Belgium, in July 2014. In August of that year, he joined the Department of Information Technology at Ghent University, where he is currently active as a Ph.D. student. His main research interest is the end-to-end Quality of Experience optimization in adaptive video streaming.

Matúš Žiak is a third year student of Technical University of Košice in Slovakia, following a Bachelor course in Informatics at the Faculty of Electric Engineering. During summer of 2017 he was an intern at Ghent University in Belgium, where he was part of imec IDLab research group. He is currently working part time as a salesforce programmer in Cassacloud (Košice, Slovakia).

Jürgen Slowack received the M.Sc. degree in computer engineering from Ghent University, Ghent, Belgium, in 2006. He then joined the IDLab research group (Ghent University - IMEC) performing research in the context of distributed video coding, achieving his Ph.D degree in 2010. After continuing for two years as a post-doctoral researcher, he became a research engineer at Barco (www.barco.com) a company active in display and networking equipment for various markets, including digital cinema, healthcare, control rooms, and meeting rooms. Currently, he is an R&D project manager, managing projects in the domain of multimedia coding and streaming, cloud processing, and artificial intelligence.

Tim Wauters obtained his M.Sc. and PhD degrees in electro-technical engineering from Ghent University in 2001 and 2007 respectively. He has been working as a post-doctoral fellow of the F.W.O.-V. in the Department of Information Technology (INTEC) at Ghent University, and is now also active as a senior researcher at imec. His main research interests focus on network and service architectures and management solutions for scalable multimedia delivery services. His work has been published in about 80 scientific publications in international journals and in the proceedings of international conferences.

Filip De Turck is professor in the department of Information Technology of Ghent University with expertise in network software and strong research interests in adaptive large-scale data processing and software systems for healthcare, anomaly detection, and resilience of ICT infrastructures and services. In this research area, he is involved in several research projects with industry and academia, serves as Vice Chair of the IEEE Technical Committee on Network Operations and Management (CNOM), chair of the Future Internet Cluster of the European Commission, and is on the TPC of many network and service management conferences and workshops and serves in the editorial board of several network and service management journals. Prof. Filip De Turck regularly organizes international workshops on the above mentioned topics, serves as Associate Editor in Chief of IEEE Transactions on Network and Service Management (TNSM), steering committee member of the IEEE Conference on Network Softwarization (IEEE NetSoft) and chair of the IEEE SDN Initiative Conference Committee, which coordinates initiative IEEE events and conferences on Softwarized Networks. He is currently (co-)author of over 480 publications, his H-index is 34 and has successfully supervised 35 PhD students so far.

Affiliations

Stefano Petrangeli[1] · Dries Pauwels[1] · Jeroen van der Hooft[1] · Matúš Žiak[2] · Jürgen Slowack[3] · Tim Wauters[1] · Filip De Turck[1]

✉ Stefano Petrangeli
petrange@adobe.com

Dries Pauwels
dries.pauwels@ugent.be

Jeroen van der Hooft
jeroen.vanderhooft@ugent.be

Matúš Žiak
matusziak303@gmail.com

Jürgen Slowack
jurgen.slowack@barco.com

Tim Wauters
tim.wauters@ugent.be

Filip De Turck
filip.deturck@ugent.be

[1] Department of Information Technology, Ghent University - imec, IDLab, Technologiepark-Zwijnaarde 15, B-9052 Ghent, Belgium

[2] Technical University of Košice, Letná 9, 04200 Košice, Slovak Republic

[3] Barco N.V. – Technology Center, Beneluxpark 21, B-8500 Kortrijk, Belgium