Improving Quality and Scalability of WebRTC Video Collaboration Applications

Stefano Petrangeli, Dries Pauwels, Jeroen van der Hooft, Tim Wauters, Filip De Turck
Ghent University – imec
name.surname@ugent.be

Jürgen Slowack
Barco N.V. – Corporate Technology Center
jurgen.slowack@barco.com

ABSTRACT

Remote collaboration is common nowadays in conferencing, telehealth and remote teaching applications. To support these interactive use cases, Real-Time Communication (RTC) solutions, as the open-source WebRTC framework, are generally used. WebRTC is peer-to-peer by design, which entails that each sending peer needs to encode a separate, independent stream for each receiving peer in the remote session. This approach is therefore expensive in terms of number of encoders and not able to scale well for a large number of users. To overcome this issue, a WebRTC-compliant framework is proposed in this paper, where only a limited number of encoders are used at sender-side. Consequently, each encoder can transmit to a multitude of receivers at the same time. The conference controller, a centralized Selective Forwarding Unit (SFU), dynamically forwards the most suitable stream to each of the receivers, based on their bandwidth conditions. Moreover, the controller dynamically recomputes the encoding bitrates of the sender, to follow the long-term bandwidth variations of the receivers and increase the delivered video quality. The benefits of this framework are showcased using a demo implemented using the Jitsi-Videobridge software, a WebRTC SFU, for the controller and the Chrome browser for the peers. Particularly, we demonstrate how our framework can improve the received video quality up to 15% compared to an approach where the encoding bitrates are static and do not change over time.

CCS CONCEPTS

• **Information systems → Multimedia streaming**; **Web conferencing**; • **Networks →** *Public Internet*;

KEYWORDS

Real-Time Communication, Remote Video Collaboration, WebRTC, Selective Forwarding Unit, ILP, Jitsi-Videobridge

ACM Reference Format:
Stefano Petrangeli, Dries Pauwels, Jeroen, van der Hooft, Tim Wauters, Filip De Turck, and Jürgen Slowack. 2018. Improving Quality and Scalability of WebRTC Video Collaboration Applications. In *MMSys'18: 9th ACM Multimedia Systems Conference, June 12–15, 2018, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3204949.3208109

1 INTRODUCTION

Remote video collaboration is widely used in a variety of applications nowadays [6, 10]. Remote conferencing solutions as the open-source Web Real-Time Communication (WebRTC) framework are preferred in these scenarios over classical streaming techniques based on HTTP, which cannot guarantee the required degree of interactivity of these applications. Even though WebRTC guarantees the low-latency and interactivity required in remote collaboration, it is affected by a scalability issue. The WebRTC framework has indeed been developed with a peer-to-peer architecture in mind. In standard WebRTC, 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, which is expensive in terms of encoders and does not scale well.

In order to reduce the scalability issue of such a peer-to-peer architecture, we propose a WebRTC-compliant framework to support the delivery of real-time communication streams. In this framework, the WebRTC sender only needs to encode a limited number of streams, much smaller than the number of receivers, at different bitrates. Consequently, multiple receivers are assigned to the same encoder at sender-side. A centralized node, called the *conference controller*, is aware of the bandwidth conditions of the WebRTC receivers and dynamically forwards the stream at the best bitrate in order to follow their bandwidth variations. Moreover, instead of keeping the encoding bitrates of the sender fixed to predefined static values, the conference controller can periodically recompute these to better follow the bandwidth conditions of the receivers, even though only a limited number of encoders is actually used. 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 [3].

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, where the interaction among the participants is usually dominated by a single entity. A classical example of such one-to-many communication is a remote teaching application. In a virtual classroom, the students, or receivers, remotely attend a live lecture given by the lecturer, the sender. 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. Particularly, we focus on the downstream side of the problem (e.g., from sender to receivers), as most of the communication follows this path (Figure 1). It is nevertheless implied that the proposed

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

MMSys'18, June 12–15, 2018, Amsterdam, Netherlands
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5192-8/18/06.
https://doi.org/10.1145/3204949.3208109

framework can guarantee the required level of interactivity, by allowing the receivers to participate in the communication.

To demonstrate the gains brought by the proposed approach, we developed a Proof of Concept (PoC) using state-of-the-art WebRTC software. The PoC is composed of one sender, equipped with three encoders, and ten receivers. Both sender and receivers uses the Google Chrome browser, while the conference controller is implemented using the Jitsi-Videobridge software, a state-of-the-art WebRTC SFU [7]. The PoC allows to clearly quantify the gains of the dynamic association of the encoding bitrates in terms of average played rate at the receivers, as opposed to a scenario where the bitrates are statically associated and do not change over time.

The remainder of this paper is structured as follows. Section 2 presents related work on conferencing solutions for WebRTC. Section 3 describes the functionalities of the conference controller and the dynamic encoding bitrates recomputation. In Section 4, the proof of concept is presented, while Section 5 concludes the paper.

2 RELATED WORK

Xu et al. perform a measurement study on real-world conferencing systems [15]. The authors report that a complete peer-to-peer architecture is never used as it does not scale to a large number of users. A Multipoint Conferencing Unit (MCU) can be used in WebRTC to improve scalability. The MCU receives all the streams, decodes and composes them in a single common stream that is sent back to the participants. Grand et al. divide the participants into regional clusters, each associated to an MCU [2]. All the MCUs are interconnected in a mesh network. This hybrid architecture allows to support a large number of users. Ma et al. use an MCU to transcode the sender stream and adjust it to the viewing conditions of the receiver [9]. 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. Nevertheless, MCU operations are extremely computationally intensive, due to the decoding-mixing-encoding processes 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 [5]. Alternatively, MCU low-level functionalities can be virtualized and deployed on-the-fly [14]. Unlike an MCU, an SFU does not require decoding/encoding operations. Its main task is to receive all the streams and selectively forward one or more streams to each participant. In this case, the amount of forwarded streams should be selected to avoid wasting bandwidth. Grozev et al. develop a speaker identification algorithm to be deployed on an SFU, to identify the last N dominant speakers of the conference [3]. Only these N streams are forwarded to the conference participants. The authors also propose to use simulcast in combination with an SFU [4]. Each participant can send up to three streams, encoded at different bitrates. The SFU forwards the highest quality to participants involved in the conversation, and the lowest quality to the remaining ones.

Unlike previous works, the proposed conference controller, implemented using the SFU principle, forwards the most suitable stream based on the bandwidth conditions of the receivers. Moreover, the dynamic computation allows the encoding bitrates to be always representative of the network conditions of the receivers, and therefore maximize the received bitrate. A preliminary evaluation of the proposed framework has been presented in previous work [11], where we showed that the dynamic bitrate recomputation results in 15% higher video rate at the receivers compared to a static, fixed association of the encoding bitrates.

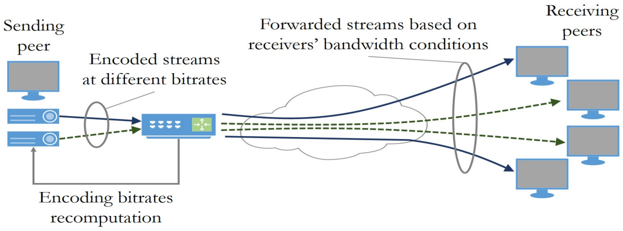
Sending peer
Encoded streams at different bitrates
Forwarded streams based on receivers' bandwidth conditions
Receiving peers
Encoding bitrates recomputation

Figure 1: The conference controller is the terminal endpoint for both the sender and the receivers, and performs the dynamic stream forwarding and encoding bitrate recomputation tasks.

3 WEBRTC CONFERENCE CONTROLLER

The central 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 (Figure 1). 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 3.1) and encoding bitrate recomputation (Section 3.2). It is worth noting that the two tasks are performed at different timescales. The bitrate recomputation is executed on a timescale of seconds, 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.

3.1 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 its estimated bandwidth. In a classic peer-to-peer WebRTC architecture, a receiver periodically reports statistics and feedback to the corresponding sender, which allows it 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 WebRTC Receiver Estimated Maximum Bitrate (REMB) messages, which are used in WebRTC to estimate the bandwidth of the remote receivers [1]. 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.

3.2 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. 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 Integer Linear Programming (ILP) formulation, which is executed every T_{opt} seconds by the conference controller. At time t when the recomputation takes place, the virtual room is composed of R receivers, each associated with a bandwidth measure b_r. The sender 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:

$$
\begin{array}{llcll}
\min\limits_{\alpha} & \sum\limits_{r=1}^{R}\sum\limits_{l=1}^{L} \alpha_{r,l}\,(b_r - B_l)^2 & & & \\
\text{s.t.} & \alpha_{r,l} & \in & \{0,1\} & \forall r \in \{1,\dots,R\}, l \in \{1,\dots,L\} \\
 & \beta_l & \in & \{0,1\} & \forall l \in \{1,\dots,L\} \\
 & \beta_l & \geq & \alpha_{r,l} & \forall r \in \{1,\dots,R\}, l \in \{1,\dots,L\} \\
 & \sum\limits_{l=1}^{L} \alpha_{r,l} & = & 1 & \forall r \in \{1,\dots,R\} \\
 & \sum\limits_{l=1}^{L} \beta_l & \leq & l_{max} & \\
 & \beta_0 & = & B_{min} & \\
 & \sum\limits_{l=1}^{L} \alpha_{r,l} B_l & \leq & b_r & \forall r \in \{1,\dots,R\}
\end{array}
\qquad (1)
$$

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.

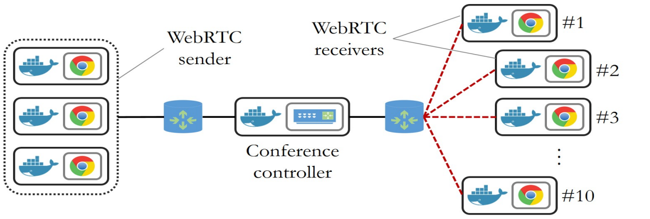

Figure 2: The PoC is composed of a single sending peer composed of three sub-senders (each encoding the video stream at a different bitrate), ten receivers, one conference controller and two layer 2 switches.

4 PROOF OF CONCEPT SETUP

The proposed framework has been implemented using state-of-the-art WebRTC software and the *containernet* network emulator to evaluate its performance in a realistic environment [12]. The setup of the PoC, shown in Figure 2, is composed of two layer 2 switches, ten receiving peers, one sending peer equipped with three encoders and one conference controller.

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 three WebRTC sub-senders, each encoding the video stream at a different bitrate. Each Google Chrome instance runs in an independent and separate docker container. Moreover, each receiver is applied with a different bandwidth trace (red dashed links in Figure 2), collected on a real 3G network [13].

To implement the conference controller, the *Jitsi-Videobridge* software is used, an open-source WebRTC SFU [7], which has been extended to implement the dynamic stream forwarding and bitrate recomputation functionalities[1]. The default behavior of the Jitsi-Videobridge is to relay a subset of the WebRTC streams in the conference to all the participants. 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. The dynamic stream forwarding is carried out by overriding this logic, so that a different dominant speaker can be manually set per receiver. Moreover, the amount of streams that can be sent to a specific receiver is limited 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.

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 3. 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

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

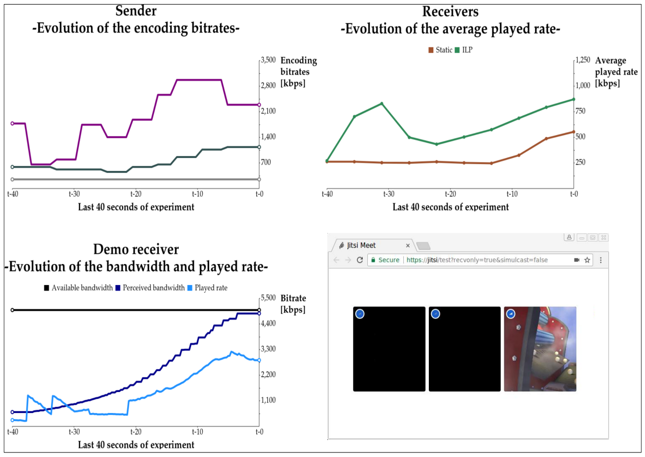

Figure 3: The HTML5 dashboard allows to control the PoC setup and to monitor the evolution of the encoding bitrates together with the performance of the receivers.

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 [8], 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 of the receivers, we set them to the bitrates generated by the ILP recomputation presented in Section 3.

Using the setup presented in Figure 2, the users will be able to test the PoC in two different scenarios. First, a *static* one, where the encoding bitrates are fixed and do not change over time. Second, a *dynamic* scenario, where the bitrates are recomputed as presented in Equation 1. An HTML5 dashboard (Figure 3) allows to visualize the performance of the system and compare the static and dynamic scenarios in three different ways. First, the evolution of the encoding bitrates is displayed (top left graph in Figure 3), to show how the dynamic recomputation can consistently alter the bitrates compared to a static association. Second, the evolution of the average played rate among the group of receivers (top right graph) is presented, both for the static and dynamic scenario. This graph allows to confirm that a dynamic recomputation can guarantee a higher video quality to the receivers compared to a static association, as the encoding bitrates are more representative of the actual bandwidth conditions of the receivers. A third graph (bottom left) displays the behavior of one particular receiver in terms of available and perceived bandwidth and played rate. The Google Chrome instance of this particular receiver will also be visualized (bottom right in Figure 3) to show how the conference controller can forward the best stream depending on the available bandwidth of the receiver.

5 CONCLUSIONS

In this paper, we presented a proof of concept 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 WebRTC protocol has been used instead, which is peer-to-peer by design and therefore presents scalability issues. In the proposed framework, instead, only a few encoders are used at sender-side, each transmitting to several WebRTC receivers at the same time. A conference controller, implemented using the Jitsi-Videobridge SFU software, dynamically forwards the most suitable stream to the receivers, based on their bandwidth conditions. Besides this short-term adaptation, the controller periodically recomputes the set of encoding bitrates using an ILP formulation, to better follow the long-term network conditions of the receivers. The gains of this approach are showcased using a virtualized network composed of ten receivers and one single sending peer equipped with three encoders. Particularly, the dynamic recomputation allows to increase the average played rate at the receivers up to 15%, compared to a static association of the encoding bitrates.

6 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 project G025615N "Optimized source coding for multiple terminals in self-organising networks" from the fund for Scientific Research-Flanders (FWO-V). This research was performed partially within the imec PRO-FLOW project (150223).

REFERENCES

[1] H. Alvestrand. RTCP message for Receiver Estimated Maximum Bitrate. *Internet-Draft draft-alvestrand-rmcat-remb-03 (work in progress)*, 2013.

[2] J. C. Granda et al. Overlay Network Based on WebRTC for Interactive Multimedia Communications. In *2015 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 1–5, July 2015.

[3] B. Grozev et al. 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, 2015.

[4] B. Grozev et al. Experimental Evaluation of Simulcast for WebRTC. *IEEE Communications Standards Magazine*, 1(2):52–59, 2017.

[5] M. A. Hossain et al. Distributed Dynamic MCU for Video Conferencing in Peer-to-Peer Network. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, Dec 2016.

[6] J. Jang-Jaccard et al. WebRTC-Based Video Conferencing Service for Telehealth. *Computing*, 98(1):169–193, Jan 2016.

[7] Jitsi-Videobridge. Webrtc compatible video router and sfu. https://github.com/jitsi/jitsi-videobridge.

[8] Libjitsi. Advanced java media library for secure real-time audio/video communications. https://github.com/jitsi/libjitsi.

[9] L. Ma et al. User Adaptive Transcoding for Video Teleconferencing. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 2209–2213, Sept 2015.

[10] H. Oh et al. WebRTC Based Remote Collaborative Online Learning Platform. In *Proceedings of the 1st Workshop on All-Web Real-Time Systems*, AWeS '15, pages 9:1–9:5, New York, NY, USA, 2015. ACM.

[11] S. Petrangeli et al. Dynamic Video Bitrate Adaptation for WebRTC-Based Remote Teaching Applications. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2018.

[12] M. Peuster et al. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016.

[13] H. Riiser et al. Video Streaming Using a Location-based Bandwidth-Lookup Service for Bitrate Planning. *ACM Transactions on Multimedia Computing, Communications and Applications*, 8(3):24:1–24:19, Aug. 2012.

[14] P. Rodríguez et al. Materialising a New Architecture for a Distributed MCU in the Cloud. *Computer Standards and Interfaces*, 44(Supplement C):234 – 242, 2016.

[15] Y. Xu et al. Video Telephony for End-Consumers: Measurement Study of Google+, iChat, and Skype. *IEEE/ACM Transactions on Networking*, June 2014.