Last N: Relevance-Based Selectivity for Forwarding Video in Multimedia Conferences

Boris Grozev
Jitsi.org
boris@jitsi.org

Lyubomir Marinov
Jitsi.org
lyubo@jitsi.org

Varun Singh
Aalto University, Finland
varun.singh@aalto.fi

Emil Ivov
Jitsi.org
emcho@jitsi.org

ABSTRACT

Multiparty conferencing has traditionally been a relatively expensive application that was only used in enterprise scenarios. Recently, however, the landscape has started to shift in ways that could change this. Ever-increasing bandwidth and processing capabilities make it possible for mobile endpoints and laptop computers to easily handle multiple incoming media streams (both audio and video). The development of Web Real-Time Communications (WebRTC) has also significantly simplified the development of video conferencing applications and made them mainstream. Both of these changes provide a way of replacing expensive video mixers (that produce composited videos) with light-weight video routers (that selectively forward streams). In this paper, we describe a Multipoint Control Unit (MCU) that identifies and selects the last N dominant speakers and forwards their streams to all the conference participants. We evaluate the performance of this Selective Forwarding Unit (SFU) against a simplistic everyone-to-everyone (full-star) MCU. Our results show that the SFU uses 45% less CPU and 63% less bandwidth when forwarding media for 10 of the endpoints in a 30-participant conference.

Categories and Subject Descriptors: H.4.3 [**Communications Applications**] Computer conferencing, teleconferencing, and videoconferencing.

General Terms: Algorithms, Design, Experimentation, Performance

Keywords: Videoconferencing, WebRTC, RTP, SFU, MCU, LastN

1. INTRODUCTION

Deployments of WebRTC have proliferated peer-to-peer video communication. Currently, multiparty conferencing is achieved via different topologies. One approach is to use full-mesh connectivity, wherein each endpoint sends media to every other, thus the endpoints quickly exhaust their available upstream bandwidth. Further, the congestion control for each media stream pair is maintained separately, hence, sending media from the same source to several endpoints competes for available capacity, which effectively produces inconsistent quality at each recipient [17].

Another approach is to use a centralized architecture. Traditionally such systems use Multipoint Control Units (MCUs) as the central component; they perform some form of signal processing on the media streams that they receive from each endpoint. MCUs usually mix audio and spatially combine video, which requires substantial computational resources. Contrarily, Selective Forwarding Units (SFUs) forward RTP packets, optionally changing their headers, but without processing the payload [22]. They do not need to decode or encode any media and this makes them generally more CPU-efficient. In order to be bandwidth efficient as well, SFUs do not forward all packets, but use a selection algorithm to decide which packets to forward to which endpoints. This algorithm is critical for the application, because the network and CPU usage as well as the user experience depend on it.

In this paper we propose a novel approach for endpoint selection. We introduce a general scheme, Last N, which orders endpoints according to their audio activity. We use an algorithm for dominant speaker identification adapted to work solely with audio-level information and which operates without decoding the audio streams. This allows the Last N scheme to be used in the context of an SFU. We implement our proposal in an open-source SFU and evaluate it in terms of the CPU and network capacity used by the SFU. Our results show significant improvements, especially in cases where a single conference has a large number of participants.

We structure the remainder of this paper as follows: Section 2 describes the implementation of the video conferencing system. Sections 3 and 4 give the specifics of the algorithms used for keeping track of the "dominant" speaker in the conference, and for selecting which video streams to forward. Section 5 describes the experiments which we performed in order to evaluate the new system, and presents the results.

2. JITSI VIDEOBRIDGE: AN SFU

In this section we describe the architecture and features implemented in a Selective Forwarding Unit (SFU).

General overview: *Jitsi Videobridge* is an SFU implementation, mainly targeted at serving multi-party conferences with WebRTC-enabled endpoints. Presently it is deployed in conjunction with Jitsi Meet – a JavaScript WebRTC application, which provides real-time multi-party audio and video conferencing. The use-cases range from casual personal meetings to business meetings with shared remote presentations. It is our endevour to increase the number of participants which can be supported in a typical conference by enhancing the functionalities provided by SFUs.

Jitsi Videobridge uses XMPP/Jingle [8] for signalling, with one of the endpoints acting as conference focus [14] and initiating a

NOSSDAV'15, March 18-20 2015, Portland, OR, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM. ACM 978-1-4503-3352-8/15/03 ...$15.00
http://dx.doi.org/10.1145/2736084.2736094.

Jingle session with each other endpoint. The focus allocates channels/resources to each participant and additionally controls the SFU through the COnferences with LIghtweight BRIdging (COLIBRI) protocol [5]. In addition to Jitsi Meet instances running in WebRTC-capable browsers, *Jitsi Videobridge* is capable of interacting with other full-fledged client software, gateways to legacy networks (e.g. using SIP), and provides enhanced server-side services (e.g. recording the participants in a conference). Figure 1 shows the structure of a typical conference.

Besides establishing channels for audio and video, each endpoint also maintains a control channel to the SFU. This is a bi-directional reliable transport channel (based on WebRTC's data channels), which can be used for notification and/or control messages in cases where RTCP cannot be used (the WebRTC API does not provide a mechanism to the JavaScript application to directly send or receive RTCP messages). The messages themselves have a custom ad-hoc format based on JSON. Relevant to the experiments in this paper are the "dominant speaker" notifications, which the SFU sends to endpoints to indicate that the main speaker in the conference has changed, and the "start" and "stop" messages which the SFU sends in order to instruct an endpoint to start or stop its video stream (See section 3 for details).

The system is in production use and there are several independent deployments making use of it; both *Jitsi Videobridge*[1] and Jitsi Meet[2] are open-source software.

RTCP Termination: The SFU supports two modes of handling RTCP. In the first mode (no termination) it just forwards all RTCP packets to every endpoint, and does not generate any new packets.

In the second mode, the SFU terminates RTCP and generates RTCP packets for the RTP session. Incoming RTCP packets are handled according to their type. The SFU does not propagate Receiver Reports (RRs), and instead it generates RRs for each incoming RTP stream. It propagates Sender Reports (SRs) to all endpoints. The SFU typically passes through other types of RTCP Reports, for example, RTCP Extended Reports, Negative Acknowledgements (NACKs), and Picture Loss Indications (PLIs). The Receiver Estimated Maximum Bandwidth (REMB) RTCP report [1] is used for performing congestion control [9]: i.e., the receiver estimates the available capacity based on variation in frame inter-arrival times. With multiparty conferencing, where there are several receivers, REMB reports will be generated by each receiver, and it is up to the SFU to either forward them to the sender or not. Passing through the REMB causes the sender to perform congestion control each time, leading to control or rate instability [17]. Therefore, *Jitsi Videobridge* does not propagate the received REMB packets and instead generates REMB reports for each sender, using the same congestion control algorithm as the endpoints [9].

RTP Header: The SFU changes two parts of the RTP header. 1) the payload type (PT); while the endpoints use the same codec profiles, it is possible for them to use different payload type (PT) numbers to indicate the same codec; 2) the abs-send-time RTP header extension [1] in case of video where it replaces the 24-bit timestamp in each outgoing RTP packet with a locally generated timestamp. This corrects the receiver's packet inter-arrival time calculation, so that only the path from the SFU to the endpoint is taken into account (but not the path from the sending endpoint to the SFU).

Security: There are several ways to address media security (encryption and signing) in an SFU-based media conference. Although the SFU does not change the payload of RTP packets in any way, it needs to change the RTP headers and generate RTCP packets. For this reason, it is not currently possible for the endpoints to exchange end-to-end encrypted media without the SFU being able to decrypt it. WebRTC mandates that all media is transported over Secure Real-time Transport Protocol (SRTP), with session keys exchanged over Datagram Transport Layer Security (DTLS) [13]. This forces the SFU to establish a separate SRTP context with each endpoint, and consequently to re-encrypt every RT(C)P packet individually.

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

[2]https://github.com/jitsi/jitsi-meet/

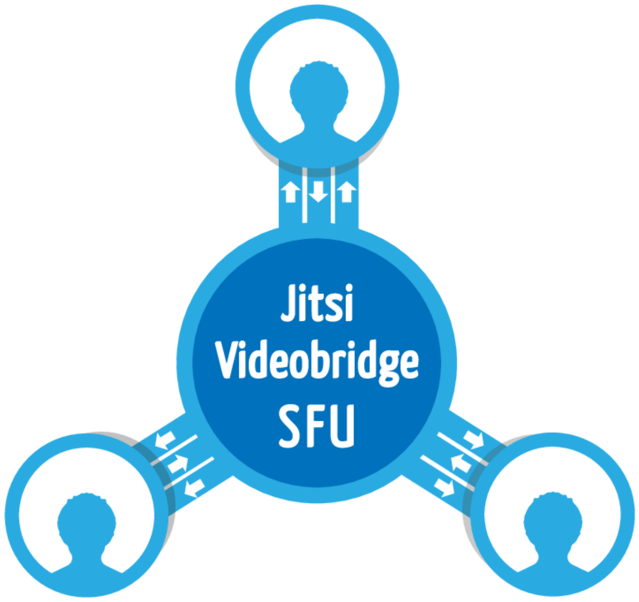

Figure 1: The media connections between endopints and the server component in a full-star conference with forwarded streams.

A different scheme which has been discussed for use with SFUs is Encrypted Key Transport (EKT) [10]. With EKT all endpoints and the SFU share a single SRTP context, hence, the SFU is not required to re-encrypt RTP packets (unless the headers were modified). Consequently, the proposals discussed in this paper can be applied without any modifications.

In the following sections we discuss our proposals for endpoint selection based on speaker order, and for identifying the dominant speaker(s) based on audio-level information.

3. LAST N: ENDPOINT SELECTION BASED ON AUDIO ACTIVITY

Media conferences deployed in a full-star topology do not scale well when the number of participants is substantial (≈ 50), and each endpoint connected to the conferencing server sends and receives streams from each other endpoint. At the endpoint we identified the following issues: 1) The user interface cannot display a large number of video streams in an efficient manner, because there is insufficient screen real-estate. 2) The required resources (CPU, network capacity) grow in proportion to the number of participants, since the endpoint has to decode, scale, and render all streams separately. At the conferencing server the issue is with the requirements for CPU and network capacity, which grow quadratically with the number of endpoints. Our evaluation in section 5 shows that either of these can be the bottleneck, depending on the hardware and the available network resources.

Here we propose Last N: a general scheme that defines a policy for endpoint selection based on audio activity. It is used by a Selective Forwarding Unit (SFU) [22] to choose only a subset of the streams to forward at any one time, alleviating the identified limitations for the endpoints and conferencing server.

Last N: The SFU only forwards a fixed number of video streams (N) to each endpoint, and changes the set of forwarded streams dynamically according to audio activity. Additionally, the receiving endpoint will continue receiving streams that may have been chosen by the participant but are currently outside of the Last N set. The Last N scheme only applies to forwarding video streams and not to audio; audio from all endpoints is always forwarded.

The integer N is a constant, configured for the whole conference and we denote the total number of endpoints in a conference as K. Then the SFU sends $K \times N$ video streams, instead of the $K \times (K-1)$ streams sent in the case of a full-star topology. This allows the SFU to scale down the requirements on the network capacity and CPU, and be capable of handling a larger number of conferences compared to the full-star topology.

From a user experience perspective, video from only a subset of all endpoints is displayed, but as soon as a participant of the conference starts to speak, their video is displayed automatically. In order to implement this scheme, the SFU relies heavily on identifying the dominant speaker (details discussed in section 4). The SFU maintains a list (L) with all endpoints currently in the conference, ordered by the last time that an endpoint was identified as the dominant speaker; thus the endpoint currently identified as the dominant speaker is always at the top of the list.

Suppose that an endpoint E has selected a set P of endpoints that it wants to permanently receive (P could be empty). Then the SFU forwards to E the selected endpoints and the first of the remaining endpoints, up to a total of at most N endpoints. That is, E receives P and the first (at most) $N - |P|$ endpoints from the list $L \backslash P \backslash \{E\}$.

Pausing video streams: When a stream from an endpoint is not forwarded by the SFU to any other participant, it is unnecessary for the endpoint to send the stream to the SFU, which is effectively ignoring it. The SFU sends control messages to endpoints instructing them to temporarily pause, or to resume transmitting their video stream. These messages can be sent over the control channel, or encoded as an RTCP extension report.

This approach can be used with any algorithm the SFU might use to change its forwarding policy, as long as the SFU keeps track of whether a stream is being forwarded to at least one endpoint or not. With Last N and no selected endpoints, it is easy to see which streams are being forwarded[3], because they are listed in the order they last spoke (L). All streams in L after stream $N + 1$ are not forwarded and can be paused. Hence, there are always $K - N - 1$ streams paused. The performance evaluation and results are discussed in section 5.

Key-frames: When a paused stream is resumed, the endpoints in order to start decoding a stream require a key-frame (an I-frame). Typically, when endpoints do not have a valid key-frame, they generate an RTCP Full-Intra frame Request (FIR) message; senders generate a key-frame in response to such messages. Instead of waiting for the receivers to generate a FIR request, the SFU preemptively sends a FIR message when it instructs an endpoint to resume the video stream. Additionally, when an endpoint is elected dominant speaker and was not in the Last N set, its stream starts to be forwarded to all receivers. This allows for a single FIR message to be sent after a dominant speaker change, even though there usually is more than one receiver and it also effectively reduces the time to correctly decode and render a stream by one RTT (i.e., instead of waiting for the receivers to send FIRs).

Further Improvements: The Last N scheme proposed here can be easily extended with additional policies, providing more complex solutions suitable for some use-cases, while preserving many of the features. One example, which might be useful for remote presentations, is to have a global set of streams which are always forwarded (which is not the same as the endpoint-selected streams, which are per-endpoint). This can be implemented simply by reordering L and keeping the desired streams at the top. Another example would be to allow N to vary per-endpoint, possibly dynamically, according to the device and/or network conditions of the endpoint.

[3]The case with selected streams is also not complicated. We do not describe it for lack of space.

4. DOMINANT SPEAKER IDENTIFICATION

An important feature for a conferencing system from a user experience perspective is being able to dynamically switch the focus to the currently speaking participant of the conference; further as an optimisation, the receiving endpoint may render not just the current dominant speaker but also a list of Last N speakers and/or set of selected speakers. SFUs enabling the Last N scheme perform Dominant Speaker Identification (DSI). Traditionally DSI is performed using raw audio streams [21], but an SFU (e.g., *Jitsi Videobridge*) forwards audio streams without decoding. Our algorithm for DSI is adapted to work with the audio levels indicated by the endpoints in each RTP packet as a header extension [7].

The desired behavior of a dominant speaker identification algorithm is as follows:

- No false switching should occur during a dominant speech burst. Both transient noise occurrences and single words that are said in response to or in agreement with the dominant speaker are considered transient occurrences. These should not cause a speaker switch.
- A speaker switch event cannot occur during a break in speech between two dominant speakers. It has to be triggered by a beginning of a speech burst.
- A tolerable delay in transition from one speaker to another, in a speaker switch event, is up to one second.
- When simultaneous speech occurs on more than one channel, the dominant speaker is the one who began speaking first.
- The relative loudness of the voice of a speaker should not influence his chance to be identified as the dominant speaker.

A speech burst is defined as a speech event composed of three sequential phases: *initiation*, *steady state* and *termination*. In the initiation phase, speech activity builds up. During the steady state, speech activity is mostly high, but it may include breaks in activity due to pauses between speech. Finally, in the termination phase speech activity declines and then stops. Typically, a dominant speech activity is composed of one or more consequent speech bursts. The point where a change in dominant speaker occurs is referred to as a speaker switch event.

The implemented algorithm for dominant speaker identification uses speech activity information from time intervals of different lengths. It consists of two stages, a local processing and a global decision. In the first stage, the audio signal of each participant is processed independently and speech activity scores are evaluated for the immediate, medium, and long time-intervals. The lengths of the time intervals correspond to and allow capturing basic speech events such as a few phonemes, a word or two, and a short sentence. In the second stage, the dominant speaker is identified based on the speech activity scores obtained in the first stage and speaker switch events are detected. It is assumed that a speaker switch event can be inferred from a rise in the three speech activity scores on a certain channel, relative to scores of the dominant channel.

Sequences and combinations of basic speech events may indicate the presence or the absence of dominant speech activity. The method distinguishes between transient audio occurrences that are isolated and those that are located within a speech burst.

Long term information is used in order to determine whether speech is present in a currently observed time-frame since dominant speech activity in a given time-frame would be better inferred from a preceding time interval than from any instantaneous signal property.

Local Processing: In this stage, the signal in each channel is processed separately with the objective of placing each signal frame into a broader context than its instantaneous audio activity. This is accomplished by processing the currently observed frame by itself in addition to a medium- length preceding time interval and in addition to a long time interval that precedes it. Thus each time we move up to a longer time interval, the speech activity obtained in the previous step is analyzed again in a broader context.

The algorithm relates to each time interval as composed of smaller sub-units. The speech activity in each time interval is determined according to the number of active sub-units by attributing a speech activity score to this number. The score is obtained from the likelihood ratio between hypothesis of speech presence and hypothesis of speech absence. The speech activity evaluation process consists of three sequential steps, referred to as immediate, medium and long. The input into each step is a sequence of the number of active sub-units acquired in the previous step. A thresholding approach allows measuring the amount of speech activity while suppressing isolated high-energy noise spikes.

For the step of immediate speech activity evaluation we use the client-to-mixer audio level indication of the frame [7] rather than a frequency representation of the frame. This is made possible by the thresholding as long as the replacement input presents a comparatively similar speech activity estimation to the original frequency sub-band test.

Audio levels are expressed in dBov (with values from -127 to 0), which is the level, in decibels, relative to the overload point of the system, i.e., the highest-intensity signal. In order to reuse the heuristically derived parameters of the original algorithm such as the thresholds, we brake the audio level range into the same number of sub-bands as the number of frequency sub-bands analyzed by the original algorithm [21].

Additionally, Jitsi Videobridge accounts for differences in the relative loudness of the background noise present in the audio signal of a participant. We apply a separate, individual initial thresholding of the input based on a history of the minimum audio level indication reported by the participant. The history maintains the minimum by considering its recency for the purposes of reflecting medium- and long-term changes in the level of the background noise.

Global Decision: The objective of this stage is to identify the dominant speaker. This stage is activated in time steps of a certain interval, which is referred to as the decision-interval. It utilizes the scores that are obtained in the local processing stage for dominant speaker identification. The approach in this stage is detecting speaker switch events, rather than selecting a dominant speaker in every decision-interval. Once a dominant speaker is identified, he remains dominant until the speech activity on one of the other channels justifies a speaker switch.The non-dominant channels are referred to as competing (for dominance).

5. PERFORMANCE EVALUATION

This section describes the testing environment, the experimental setup the and results from all performed tests.

5.1 Testbed

The SFU is running on a machine with a quad-core Intel Xeon E5-1620 v2 @ 3.70 GHz processor. All tests comprise of a single conference with a varying number of participants. One endpoint in the conference is a Chrome instance running *Jitsi Meet*, with the microphone and camera disabled. The rest of the endpoints come from a Jitsi Hammer instance. This is an application developed specifically for load-testing the SFU. It creates multiple endpoints, each streaming one audio and one video stream from pre-recorded files. The two streams have a combined average bitrate of about 515 Kbps. These endpoints do not implement any congestion control mechanism and their RTCP support is limited. They only emit Sender Reports and do not react to REMB, NACK, or FIR messages. The streamed video includes keyframes every 1.3 seconds on average, which allows decoders to relatively quickly regain decoding state, thus the state of the conference can be monitored without the need to handle FIR or NACK requests.

In order to trigger changes to the dominant speaker in the conference, the participating endpoints (from Jitsi Hammer) add audio-levels in the format used by the DSI algorithm [7]. At any point in time one endpoint sends levels associated with the human speech, while the others send silence. The endpoint which sends speech levels changes every 4 seconds, with the new one being chosen randomly. Note that we change only the audio-level field and not the RTP payload of the audio streams. The "start" and "stop" messages are implemented with RTCP packets.

We denote the number of load-generating endpoints K, so, including the Chrome instance, there are always $K + 1$ endpoints in the conference. We denote the Last N value N, so each endpoint receives (at most) N video streams. With $N = -1$ we denote that Last N is not used, and all streams are forwarded to everyone. The number of video streams sent by the SFU is either $(K + 1) \times N$ (SFU), or $(K + 1) \times (K - 1)$ (full-star, $N = -1$). The number of streams received is either K or $N + 1$, depending on whether video pausing is used.

All data is gathered in intervals of one second. On all graphs, the values are means over an interval of at least 120 seconds, and the error bars indicate the standard deviation of the sample. The measured CPU usage constitutes the fraction of the last 1-second interval which the CPU spent in either `User`, `Nice`, `System` or `IOWait` state (what the `top` command shows on the "CPU(s)" line), and 100% would indicate that all 8 logical cores were in use.

5.2 CPU usage in a full-star conference

In this test we measured the CPU and bandwidth used in a conference with a full-star topology ($N = -1$). We varied the number of endpoints K (10, 15, 20, 25, 29, and 33). Figure 2 shows the variation in CPU usage compared to the aggregate bitrate. We observe linear growth, which is to be expected because the most computationally intensive part of the SFU's operation is encryption and decryption. Based on these values, we conclude that in reasonable practical scenarios both the computing and the network resources might constitute a bottleneck. An average machine can serve a 100 Mbps link, while a less powerful machine might not be able to cope with 1 Gbps of bitrate. Assuming that about 500 Kbps is a typical bitrate for a video stream in a conference, we observe that, as expected, the required resources increase rapidly: a conference with 33 participants requires on the order of 500 Mbps capacity for the SFU.

5.3 Full-star vs Last N

In this test we measured the output bitrate of the SFU with $K = 10, 15, 20, 25, 30$, with four configurations: full-star, and Last N with

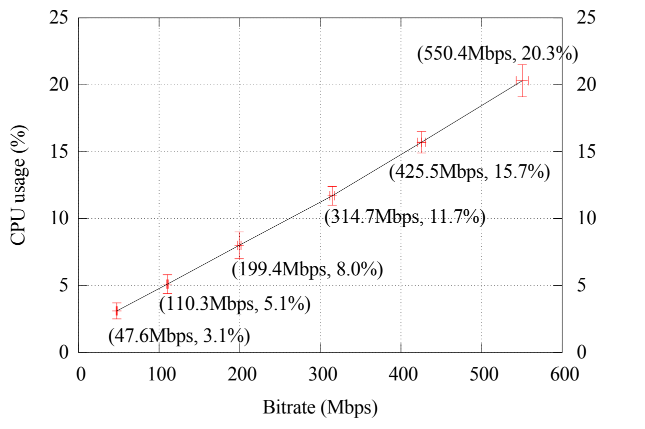

Figure 2: The CPU usage for different aggregate throughputs in a full-star conference. The CPU usage is mainly contributed by the decrypting and re-encrypting of the RTP packets. The error bars represent the standard deviation.

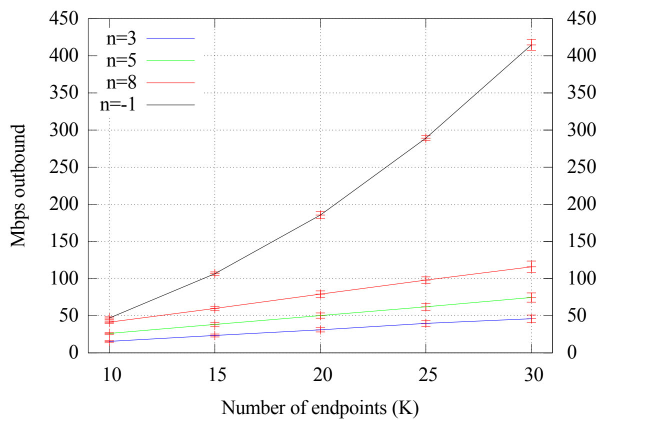

Figure 3: The outbound bitrate at the SFU, as the number of endpoints grows. Shown are four Last N configurations. The error bars represent the standard deviation.

$N = 3, 5, 8$. Video pausing was enabled. Figure 3 shows that when Last N is used, the bitrate grows linearly with K, with a coefficient that depends on N, while with no Last N it grows quadratically. This constitutes a significant improvement when Last N is used. For $K = 30$, the gain is 72%, 82%, and 89% for N 8, 5, and 3, respectively. Counting the aggregate bitrate the gain is almost the same: 72%, 82%, and 88%. We also see that different compromises between bitrate and number of visible videos can be achieved by simply varying the value of N.

5.4 Video Pausing

In this test we observed the differences in the total bitrate and CPU usage between the two modes of Last N: with video pausing on and off. We fixed $K = 30$ and changed N to *25, 20, 15, 10, 8, 5, 3, 2, 1*. The graphs also show values for $N = 30$; for them Last N is disabled. We plot them as $N = 30$, because it is effectively the same (i.e., in both cases everything is being forwarded). Figure 4 shows variation in the bitrate, and figure 5 shows the variation in CPU usage. We observe that for high values of N there is little or no gain, but it increases when N is lower. This is expected because 1) the total bitrate is lower and 2) there are more endpoints with

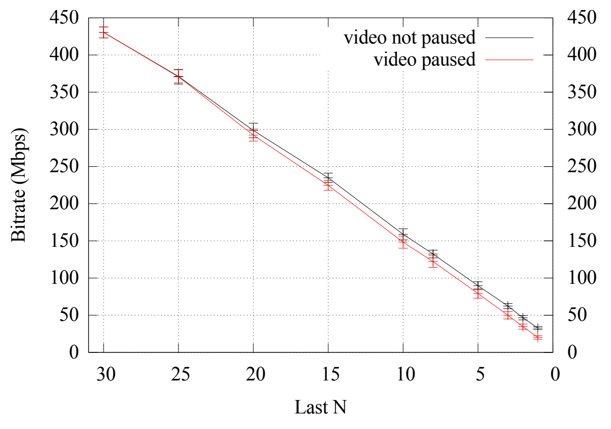

Figure 4: The aggregate bitrate at the SFU for a fixed number of endpoints (30) and varying value of N. Compared are the two modes of Last N: with and without video pausing.

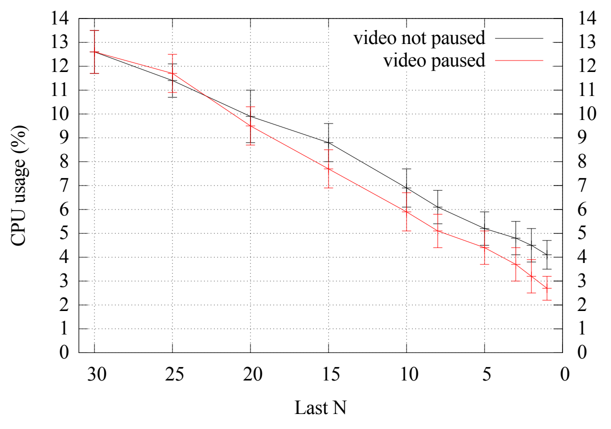

Figure 5: The CPU usage at the SFU for a fixed number of endpoints (30) and varying value of N. Compared are the two modes of Last N: with and without video pausing.

paused video. When N is set to 8,[4] we see an 8% decrease in bitrate and a 16% decrease in CPU usage. We expect that pausing video streams will be beneficial in practice, because it has the additional advantage of decreasing CPU usage for the endpoints with paused video by eliminating the need to encode video.

6. RELATED WORK

The Real-time Transport Protocol (RTP) [16] is designed to transfer multimedia data and is favoured over TCP due to the low-latency requirements [3]. Many congestion control algorithms have been proposed for peer-to-peer conversational real-time video, which attempt to match the media rate to the available end-to-end path capacity. These algorithms [18, 15, 19, 20, 11, 9, 12, 23, 6] rely on the congestion indicators reported in the RTCP Receiver Reports from the remote endpoint.

Systems providing multi-party call services typically structure the participating endpoints in a mesh or a star topology. In a full-mesh topology media from each endpoint is sent to every other endpoint. Currently the congestion control for each media stream pair

[4]We assume this to be a reasonable value for practical use, because of user interface considerations.

is maintained separately, hence, pairs with the same source compete with each other for available capacity, which effectively produces inconsistent quality at each recipient [17]. If the endpoints are able to detect shared bottlenecks between each pair of participants and merge the congestion control of those streams [4] it would be possible to fairly share the capacity. However, this would still require separately encoding the same media streams for each endpoint.

There are two types of server components often used in centralized architectures. Selective Forwarding Units (SFUs) route RTP packets without changing their payload [22], and without the need to introduce any delay at the application level[5]. Contrarily, traditional Multipoint Control Units (MCUs) process the payload in a way which requires decoding. This adds an inherent delay caused by synchronization of de-jitter buffers [2].

7. CONCLUSION AND FUTURE WORK

In this paper we propose a scheme (Last N) which an SFU uses to select which video streams to forward and which to drop. We present an implementation and evaluate it in terms of the used network and computing resources. We examine the effects of varying the number of forwarded video streams (N), and the possibility to temporarily turn-off unused streams. Our results show that the expected performance gains are achievable in practice. Notably, we observe a drop from a quadratic to a linear (with respect to the total number of endpoints) growth of the outbound bitrate used by the SFU. Additionally, for a fixed number of endpoints, tweaking the values of N results in gradual changes to the bitrate, making the system adaptable to different situations. Based on the results, we expect that Last N will be useful in practice in two scenarios: allowing for small-to-medium conferences to work more efficiently (thus allowing a single server to handle more conferences), and increasing the maximum number of endpoints in a conference, given the same resources.

Last N is quite general and allows for different modifications and improvements on top of it. In particular, one interesting modification, which we intend to study, is maintaining different values of N for each receiver, while keeping the list L global for the conference. This would allow a receiver's N to change dynamically according to the network conditions, the current bitrate of other endpoints' streams, and possibly other metrics.

8. REFERENCES

[1] H. Alvestrand. RTCP message for Receiver Estimated Maximum Bitrate, October 2013. IETF Internet Draft.

[2] Peter Amon, Madhurani Sapre, and Andreas Hutter. Compressed domain stitching of hevc streams for video conferencing applications. In *Packet Video Workshop (PV), 2012 19th International*, pages 36–40. IEEE, 2012.

[3] Eli Brosh, Salman Abdul Baset, Dan Rubenstein, and Henning Schulzrinne. The Delay-Friendliness of TCP. In *Proc. of ACM SIGMETRICS*, 2008.

[4] Safiqul Islam, Michael Welzl, Stein Gjessing, and Naeem Khademi. Coupled congestion control for rtp media. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14. ACM, 2014.

[5] Emil Ivov, Lyubomir Marinov, and Philipp Hancke. COnferences with LIghtweight BRIdging (COLIBRI), XMPP Standards Foundation XEP-0340, January 2014.

[5]Note that SFUs may still store RTP packets in buffers, sometimes referred to as (de-)jitter buffers, for other reasons (for example in order to respond to retransmission requests). In these cases the de-jitter buffers do not introduce any additional delay.

[6] Ingemar Johansson. Self-clocked rate adaptation for conversational video in lte. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14. ACM, 2014.

[7] J. Lennox, E. Ivov, and E. Marocco. A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication, IETF RFC 6464, December 2011.

[8] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan, and Joe Hildebrand. Jingle, XMPP Standards Foundation XEP-0166, December 2009.

[9] H Lundin, S Holmer, H Alvestrand, L De Cicco, and S Mascolo. A Google Congestion Control Algorithm for Real-Time Communication on the World Wide Web, 2014. IETF Internet Draft.

[10] J. Mattsson, D. McGrew, D. Wing, and F. Andreasen. Encrypted Key Transport for Secure RTP, October 2014. IETF Internet Draft.

[11] Marcin Nagy, Varun Singh, Jörg Ott, and Lars Eggert. Congestion control using fec for conversational multimedia communication. In *Proceedings of the 5th ACM Multimedia Systems Conference*, MMSys '14, pages 191–202. ACM, 2014.

[12] Piers O'Hanlon and Ken Carlberg. Dflow: Low latency congestion control. In *Proceedings of the 2013 IEEE ICNP Workshop on Capacity Sharing Workshop*, CSWS '13. IEEE, 2013.

[13] E. Rescorla. WebRTC Security Architecture, July 2014. IETF Internet Draft.

[14] J. Rosenberg, H. Schulzrinne, and O. Levin. A Session Initiation Protocol (SIP) Event Package for Conference State, August 2006. RFC 4575.

[15] A. Saurin. Congestion Control for Video-conferencing Applications. Master's thesis, University of Glasgow, December 2006.

[16] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications, IETF RFC 3550, July 2003.

[17] Varun Singh, Albert Abello Lozano, and Jörg Ott. Performance analysis of receive-side real-time congestion control for webrtc. In *Proc. of IEEE Workshop on Packet Video*, PV '13, 2013.

[18] Varun Singh, Stephen McQuistin, Martin Ellis, and Colin Perkins. Circuit breakers for multimedia congestion control. In *Packet Video Workshop (PV), 2013 20th International*, pages 1–8. IEEE, 2013.

[19] Varun Singh, Joerg Ott, and Igor Curcio. Rate adaptation for conversational 3G video. In *Proc. of INFOCOM Workshop*, Rio de Janeiro, BR, 2009.

[20] Varun Singh, Joerg Ott, and Igor Curcio. Rate-control for Conversational Video Communication in Heterogeneous Networks. In *in Proc. of IEEE WoWMoM Workshop*, SFO, CA, USA, 2012.

[21] Ilana Volfin and Israel Cohen. Dominant speaker identification for multipoint videoconferencing. *Computer Speech Language*, 27(4):895 – 910, 2013.

[22] M. Westerlund and S. Wenger. RTP Topologies, November 2014. IETF Internet Draft.

[23] X. Zhu and R. Pan. NADA: A Unified Congestion Control Scheme for Low-Latency Live Video. In *Proc. of IEEE Workshop on Packet Video*, PV '13, 2013.