Load and Video Performance Patterns of a Cloud Based WebRTC Architecture

Vamis Xhagjika*†‡, Òscar Divorra Escoda*, Leandro Navarro† and Vladimir Vlassov‡
*Tokbox Inc. - a Telefónica company, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
‡Royal Institute of Technology, Stockholm, Sweden
Email: {xhagjika, leandro}@ac.upc.edu, {vamis, oscar}@tokbox.com, vladv@kth.se

***Abstract*—Web Real-Time Communication or Realtime communication in the Web (WebRTC/RTCWeb) is a prolific new standard and technology stack, providing full audio/video agnostic communications for the Web. Service providers implementing such technology deal with various levels of complexity ranging anywhere from: high service distribution, multi-client integration, P2P and Cloud assisted communication backends, content delivery, real-time constraints and across clouds resource allocation. This work presents a study of the joint factors of: multi-cloud distribution, network performance, media parameters as well as back-end resource loads. To the best of our knowledge such study has not been explored in previously work. The monitored workload is sampled from real users and does not present a synthetically generated load, additionally the performance data is sampled both by passive user measurements as well as server side measurements. The patterns correlating such factors enable designing adaptive resource allocation algorithms and media Service Level Objectives (SLO) spanning over multiple data-centers or servers. Based on our analysis, we discover strong periodical load patterns even though the nature of user interaction with the system is mostly not predetermined with variable user churn.**

***Keywords*-load measurements, webrtc, rtp/rtcp, media, bitrate, stream allocation**

I. Introduction

WebRTC[1], [2] is the HTML5 extension for real-time communications, enabling live media communications between two or more parties using standardised web technologies. WebRTC/RTCWEB is currently specified through three main aspects:

- WebRTC W3C standard API specification for use in web browsers [1].
- RTCWEB IETF standard recommendation for the set of protocols necessary for media communications for every connection [2].
- WebRTC reference software media stack (open source component of Chrome browser), implementing previous specifications [3].

WebRTC/RTCWEB are a set of standard recommendations conceived for delay non-tolerant applications where *interactive* real-time communication is necessary. One application of WebRTC/RTCWEB is multiparty audio/video conferences. A conference is a session where each participant, publishes his audio/video sources while simultaneously receiving audio and video streams from other participants.

WebRTC clients are general purpose Web Browsers or devices that implement WebRTC/RTCWEB compatible standards. Common nomenclature for both is WebRTC Endpoint[1] (hence, both referred as such in the remainder of this work). The WebRTC/RTCWEB standard includes both P2P and cloud-relayed communications. This work focuses on real-time media transmission leveraging WebRTC/RTCWEB and cloud-relayed architectures. An analysis of quality of such media architecture operation is of utmost importance for user experience and the overall performance of the system.

The protocol in charge to deliver media is the Real-Time Transport Protocol (**RTP**) and uses Real-Time Transport Control Protocol (**RTCP**) for quality control. RTP/RTCP[4] is a general purpose transport protocol that provides support for multi-homing. It is standardized to run over both lower level UDP and TCP protocols (although UDP is usually the rule for timeliness performance). RTP is agnostic to specific codecs and can function as a transport for both video and audio stream formats. For example, in the framework of WebRTC/RTCWEB we can encounter VP8, H.264 and VP9 video codecs, while for audio OPUS, ISAC, G.722 or G.711 are common as well.

Live Audio/Video Conferencing in WebRTC/RTCWEB is implemented to use RTP to deliver media to endpoints and servers. In middlebox/server based topologies[5], each endpoint publishes one or more RTP streams for each media stream, and subscribes to each of the RTP streams of the other participants in the session. Other typical mechanisms are also implemented as well by means of a backend, like STUN and TURN for Nat-Trasversal. WebRTC is supported natively by major web browsers (e.g. Chrome, Firefox or Edge) and provides a free real-time communication medium.

Common middlebox/server topologies include using Multipoint Control Units (MCU) or Selective Forwarding Unit (SFU). MCUs typically implement both software assisted multicast as well as media translation as needed, while SFUs selectively forward to each participant media (and control) packets in more or less sophisticate ways without transcoding operations. Fig. 1 shows the high level design of such an architecture.

[1] Standardized in: https://tools.ietf.org/html/draft-ietf-rtcweb-overview-12

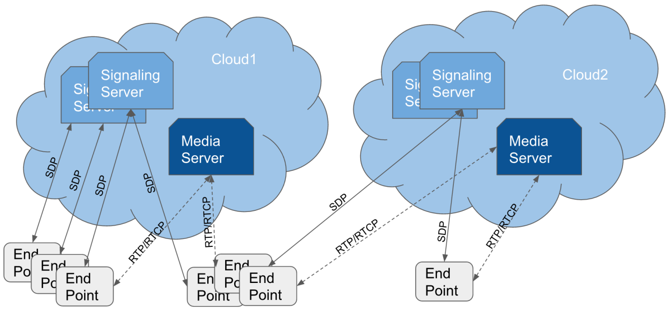

Figure 1. System Overview

In this work we will focus on examining media parameters and load profiles in the scenario where media operations use a Selective Forwarding Unit (SFU) as media relay (Fig. 1). Media relaying presents a better alternative to P2P communications as it permits to lower upstream bandwidth needs of the clients implementing WebRTC compatible communication software. Without a selective forwarding unit clients would need upstream bandwidth proportional to the the number of endpoints participating in the same session.

By having a Cloud SFU, each sending endpoint needs to send at most one stream for each published stream while the Cloud backend takes care of replication and delivery to all of the receiving endpoint. Upstream bandwidth is a limited resource for some network deployments (as an example we consider both traditional asymmetric ADSL as well as mobile devices) and thus multi-party sessions would be limited to the upstream bandwidth of the participants. In a scenario not supporting Cloud SFUs, and using a p2p mesh among endpoints, as these join a session and the stream count increases, the performance would degrade rapidly, and the use of resources increase rapidly, rendering reliability unpredictable as it depends on the upstream bandwidth of each participant.

Compared to an MCU including transcoding, an SFU provides a much more flexible streaming architecture that is a better fit to the requirements of modern use cases, at the same time that it refrains from using and excess of resources in the cloud. A key benefit of a selective forwarding unit is the ability to forward to each receiving endpoint a quality that is as close as possible to the network or cpu capabilities of that endpoint without having to use expensive transcoding. E.g. When the sending endpoint implements simulcast (sending multiple video qualities at the same time) the SFU can pick and send exactly the closest quality that matches the network requirements of each receiving endpoints.

Network quality of service is of utmost importance for these communication architectures. Also, one of the most important properties directly related to video quality is video bitrate. For resilient real-time communications, bitrate needs to adapt at every moment to the available resources in clients and network, and avoid dropping communication. End-to-end rate-control in combination with RTP/RTCP takes care of it. In this work, we will focus on studying the impact of machine load in terms of streams/server towards rate-control and client received bitrate.

The motivation of this work is thus the study and definition of stream load patterns (per SFU), bitrate and other system parameters. With these, one can devise automatic algorithms to predict resource needs and enforce Service Level Objective (SLO) limits. This can lead to improved user experience and service cost management. The observed patterns concern both: periodic repetitive server loads and the influence they have on media quality (such as bitrate). The SFU as a stream relaying, selection and duplication component directly impacts the quality of the media in a WebRTC session. Processing delay introduced by the SFU is directly propagated to the endpoints and as such could produce various media quality problems such as communication delays, stream artifacts (poor quality indicators glitches etc...), and media de-synchronization (inside the same stream between audio and video, or between different streams of the same session).

The remainder of this paper is organized as follows: in Sec. II we provide a study and characterization of server stream loads for different servers over an extended time interval. Sec. III provides measurements of video bitrates in representation of media quality for different clients and their correlation with load periodicity. Finally, Sec. V draws conclusions from results and comments future work toward multi-cloud resource allocation in WebRTC/RTCWEB backends.

II. Load Characterization

Load characterization is a critical factor in devising resource allocation strategies for a distributed service. The scenario we are investigating is composed of a distributed software multicast for a real-time, delay non-tolerant and subscribers-publishers cloud delivery system. The SFU is the multicast backend while the subscribers and publishers are the consumers and producers of media respectively. Table. I shows the distribution of load measurements taken on our test cloud as number of streams per server over $2min$ intervals. Publisher Streams have a mean of 37.31 streams/server while having a $25\% - 75\%$ percentile range of $32-51$ streams/server. The number of Subscriber streams, on the other hand, are centered at 85.52 streams/server and have a $25\% - 75\%$ percentile range of $24-125$. The number of streams/server without discriminating subscribers and publishers is centered around 122.83 and has a $25\% - 75\%$ percentile range of $44 - 177$.

A very interesting aspect of such distribution is that the maximum number of streams per machine is orders of magnitude higher than the respective mean and 75% percentile range. This aspect is very important as once the first stream

Table I
DATA DISTRIBUTION LOAD TEST CLOUD 2MIN INTERVAL

Property	Publisher Streams	Subscriber Streams	Streams
Count	190279	190279	190279
Mean	37.31	85.52	122.83
25%	32	24	44
75%	51	125	177
Max	159	868	980

is allocated to a server, the following streams associated to the same session need to be allocated on the same server. We conduct this study with the assumption that a session does not span multiple servers, while different sessions can be allocated in different servers or Clouds. Additionally session can't migrate to a different servers as the real-time nature of the service would be severely impacted. Migrating sessions would require additional setup time for resources, and brief service disruption to handle hand-over, this challenges make resource scheduling a critical task for a Cloud back-end SFU. If multiple big sessions would be allocated on the same machine, that could cause such sessions to hit the machine stream capacity limit. Once such limit is reached, it would cause problems for streams joining the session as the SFU would not be able to handle the load.

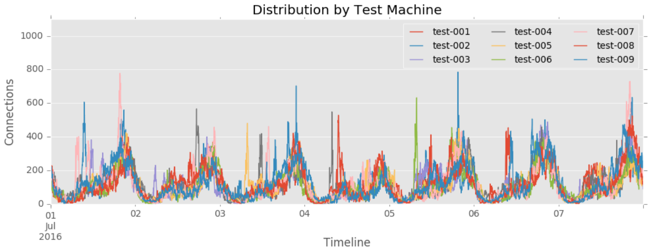

Figure 2. Test Datacenter/Server Load Distribution 7days period

We visualize in Fig. 2 the average count of streams/server as measured for one week worth of data (a subset of the dataset used for this resource allocation part). Visualization of the entire dataset would be tedious for longer periods. The load presented in Fig. 2 is sampled in 2min intervals as measured from server logs over our test data-center. In general, we observe from data that the there is a strong periodic load pattern. Further exploring such pattern, we examine the load distribution in the form of a lag plot (Fig. 3) where each lag unit represents a duration of $2min$.

The lag plot shows a clustering of values around the diagonal, which represents a strong positive auto-correlated sequence. Data-center centric load as such can be well approximated by an auto-regressive (or running-averages) model. The linear regression equation covering the lag observations is written in Eq. 1 with parameters $slope = 0.9974$ and $intercept = 2.8282$.

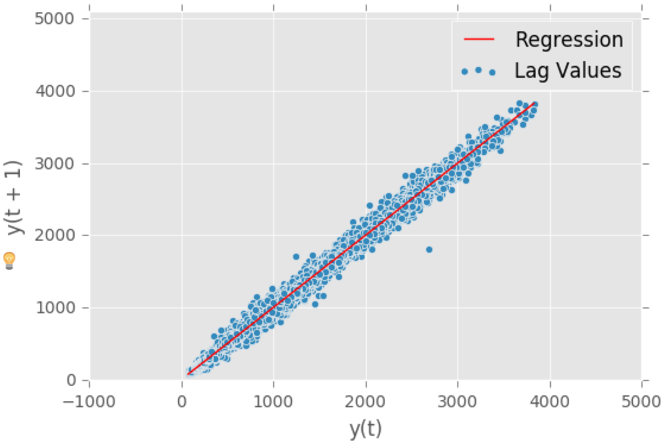

Figure 3. Datacenter total load lag plot 1month period

$$Y_{t+1} = 0.9974 * Y_t + 2.8282 \qquad (1)$$

Even though we are able to predict the total load going to a defined data-center, a resource allocation algorithm can still get into problems under the restrictions that once the first stream of a session is assigned to one server all other streams of that session will be assigned to the same server. As such, extra care needs to be taken in allocating sessions to servers, so that the load is spread fairly.

III. MEDIA BITRATE ANALYSIS

A. Bitrate Distribution

Given a media codec, bitrate is a key quality metric as it is directly related to resolution, quality and frequency of video frames being encoded at the endpoints. As such, we separate our analysis into the following video resolutions for ($width$ x $height$) QVGA(320x240), VGA(640x480) and HD(1280x720), all at 30 $frames/second$. The codec used for the analysis is VP8 (MtI[2] in WebRTC/RTCWEB endpoints). From previous measurements and also domain knowledge, we can consider that for each of these resolutions, and a wanted frame rate of 30$frames/second$, a good enough average video bitrate for single layer streaming (only one video resolution encoded at the source) is respectively: QVGA (300kbs), VGA (500kbs) and HD (1.2Mbps).

The first thing we examine is the distribution of bitrate values for both publishers and subscribers. As shown in Fig. 4, **VGA** and **QVGA** comply with the expected behaviour and exhibit normal distributions centered around $317.9Kbps$ and $492Kbps$ respectively. These two distribution are well centered and have an acceptable standard deviation of 90.7 and 130.8 respectively. In the case of the **HD** resolution, a different behaviour is exhibited in which the bitrate profile is distributed over a longer range of values and

[2] Mandatory to Implement

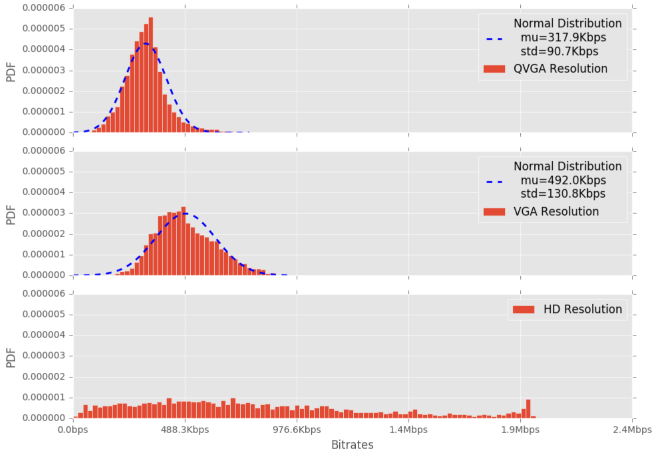

Figure 4. Subscribers Probability Distribution Functions

only a portion of the values is within the upper acceptable range of bitrates. Based on the data gathered, we conclude that such behaviour is mostly due to still common limited upload bandwidths of client access networks (i.e. ADSL), together with its effect on WebRTC software rate-control behavior[3]. That clearly restricts subscribers' statistically received bitrate and quality of experience.

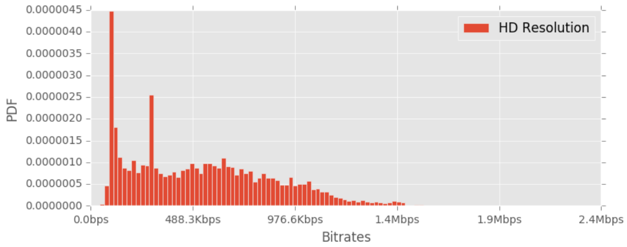

Figure 5. HD Publisher Probability Distribution Functions

The distribution of outgoing bitrates for Publishers in the HD resolution shows the same distribution seen for Subscribers. If we compare the distributions for HD Subscribers in Fig. 4 and Publishers in Fig. 5, it is clear that the limiting factor that causes low level bitrates is actually upload bandwidth. The publishers cannot keep up with the needed bitrate to stream HD video, congestion is generated limited by the underlying network infrastructure, and rate-control algorithms keep bitrate lower than desired in order to avoid congestion.

B. *Bitrate and Load correlation*

From data, we can discern a clear pattern of decreasing bitrate with the increase of in number of streams allocated in a datacenter. Such pattern provides a good limiting factor for QoS specific optimizations. Knowing this behaviour in advance, for a given bandwidth and server cpu capacities, we can limit the number of streams per machine in order to guarantee a minimum bitrate range. This can effectively give rise to allocation algorithms that are QoS aware and try to optimize video quality on the platform.

We present in Fig. 6 temporal samples of average bitrates sampled over the machines of our test data-center for a period of 9 Days in 2min client samples, in order to try and find a correlation between load and impact on bitrate quality. We calculate the *Pearson* correlation coefficient and p-values in order to see if there is some linear correlation between load and bitrate quality. On these measurements, servers never hit their limit capacity, and as such, machine overload does not impact the results. Pearson coefficients in case of **QVGA** and **HD** are both 0 and show no correlation, although in the case of **VGA** we have a coefficient of -0.48 with p-values 0 which mean that there is a correlation between the two. Further work will be focused on understanding and exploiting this pattern for performance/bitrate trade-offs.

IV. Monitoring and System Architecture

The testbed used to gather the data used in this work, is built in a Cloud environment that was constructed from a micro deployment of a full Cloud relayed WebRTC infrastructure. We allocate a number of bare-metal machines instances in a Cloud provider on which we deploy Service API, Control Signaling Services and Media Selective Forwarding Units infrastructure. The data is sampled from both the server-side and the clients sides. WebRTC endpoints are software based components running a Javascript compatible client. All servers were allocated in the same data center and provided seamless access to the machines while each one of the sessions is allocated by a load balancer based on the actual load of servers. In our experimental setup, sessions are not permitted to span multiple servers in order to limit the spread of stream latency as well as increase system stability. The components of our monitoring systems and the testbed setup are shown in Fig. 1 where the relationship between the various components are presented. Our testbed incorporates the following components supporting a full WebRTC production level deployment:

1) **Service API** is the entry point to the infrastructure and is a HTTPS based application that functions as a resource selection service and authentication.
2) **Control Signaling Services** Is a software component implementing HTTP message communication and is used to exchange messages based on Session Description Protocol (SDP) protocol specification, which is used to negotiate stream parameters between WebRTC endpoints. Each server is configured to have one Signaling service instance.
3) **Media Selective Forwarding Units** This software component is present in all of the servers allocated and handles media stream forwarding for the sessions that are allocated in each of the machines.

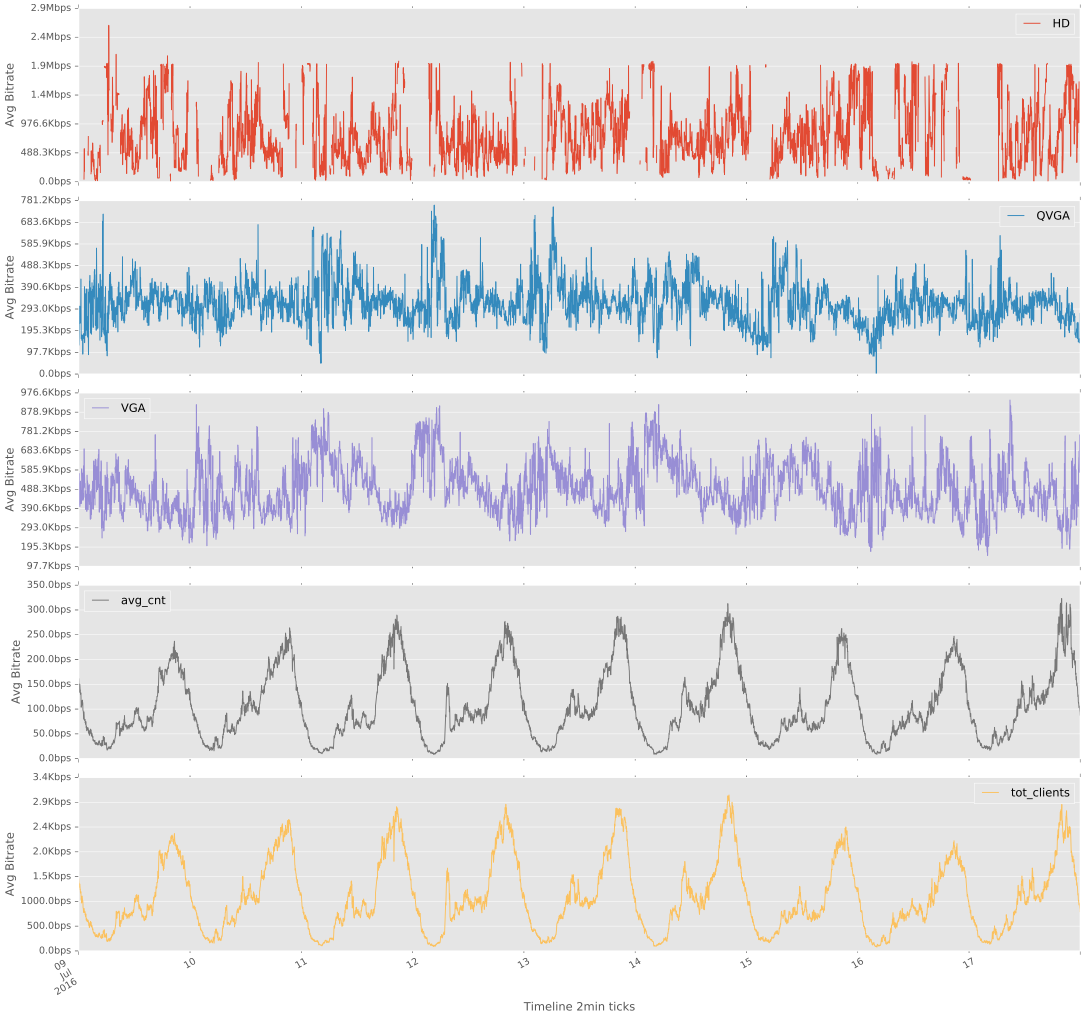

Figure 6. Bitrate and Respective Loads over time

The **Service API** is a HTTPS based server which is in charge of allocating sessions and streams based on machine loads. In our test bed, there is only one centralized instance of the allocator service api as this is only used to decide where to allocate streams the first time that the request for a stream is made. In having such one time only usage nature, it is guaranteed that this system component is not a bottleneck for the use-case analyzed in this work. Service API is only responsible to direct client to the Signaling and SFU servers, and does not transport media traffic, or is subject to high load in any case.

In order to setup endpoint capabilities a **Signaling Service** is needed that takes care of configuring endpoints with compatible features. In general in WebRTC the signaling component is not a media transport protocol, and as such is used only to setup the stream parameters and does not transport any kind of media. For this work, endpoints implement a websocket-based signaling protocol that is used to exchange SDP compliant messages and to setup the media protocols and capabilities for the actual audio/video communication. WebRTC as a standard does not define specific protocols for signaling on purpose. The choice for signaling is a non-functional API of the architecture which can depend on the application and/or use-case. The application nature of endpoints make a standard for signalling the less and less necessary now-a-days. Through using this signaling service, the endpoints negotiate a common set of features that are supported by them all and enables direct or relayed

Service API Units Service	x1
Signaling Units Service	x9
Selective Forwarding Units Service	x9
Total Servers	x9

Table II
TESTBED SERVICE COUNT

media inter-communications, and keep the status of the call updated.

At last the **Selective Forwarding Service** component is loaded in each of the servers of our setup and is responsible for implementing audio/video communication relay units. Media communications are implemented using the RTP/RTCP protocol and can accept streams of custom video resolutions based on VP8 video codec. Supported audio codec is Opus which is the Mandatory to Implement choice for production ready WebRTC environments.

The exact number of resources used in conducting this study are listed in Tab. II. We have a total number of 9 bare-metal machines hosted into a Cloud resource providers, on each one of the machines we have deployed both a **Signaling Service** component as well as a **Selective Forwarding** media component. On one of the allocated machines we have allocated one instance of the **Service API** responsible for allocating sessions to servers.

V. Conclusions

Previous work conducted in [6] deals with both performance aspects and bitrate estimation algorithms, but the study is limited to a small controlled environment. Work conducted in [7] [8] explore the behaviour of multiple rate-control algorithms and streaming properties, as well as introducing novel rate-control algorithms. Resource allocation for WebRTC in [9] leads to a Network Virtual Functions based cloud architecture to interconnect IP Media Subsystems and WebRTC.

Based on such previous research and the results presented in this work, we observe that load is predictable and the discovered patterns can be used in load mitigation and adaptive resource allocation for cloud based SFUs. This work provides evidence of strong load and video bit rate temporal patterns that can be further exploited in order to guarantee better user experience. Such patterns can be used as well in order to consider potential trade-offs between overall user experience and operational costs.

This study focuses on WebRTC sessions that do not span multiple servers, in order to have a more stable Cloud backend and lower potential latency and bandwidth bottlenecks. Future work will focus not only on the potential to exploit the discovered patterns in ensuring media quality for WebRTC clients, but also on providing novel SFU models with sessions spanning multiple servers. By spanning a session over multiple servers more advanced load allocation policies can be enforced, which are not possible otherwise. By breaking the assumption of sessions running within the confines of one machine, load can be mitigated as streams join the session thus lowering their impact on overall media quality.

Acknowledgment

This work was done in the framework of an Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) from the Education, Audiovisual and Culture Executive Agency (EACEA) of the European Commission under FPA 2012-0030, and Spanish government under TIN2013-47245-C2-1-R. A special mention is for Tokbox Inc. *a Telefónica company*, for funding this research work, and providing the testing environment and underlying technology to build on. We thank as well Michael Wilson and Jose Carlos Pujol from Tokbox Inc. for fruitful discussions.

References

[1] World Wide Web Consortium. (2016, Nov.) Webrtc 1.0: Real-time communication between browsers. [Online]. Available: https://www.w3.org/TR/webrtc/

[2] The Internet Engineering Task Force. (2016, Nov.) Real-time communication in web-browsers. [Online]. Available: https://datatracker.ietf.org/wg/rtcweb/documents/

[3] The webrtc project. [Online]. Available: https://webrtc.org/

[4] Network Working Group. (2003, Jul.) Rtp: A transport protocol for real-time applications. [Online]. Available: https://tools.ietf.org/html/rfc3550

[5] Internet Engineering Task Force (IETF). (2015, Nov.) Rtp topologies. [Online]. Available: https://tools.ietf.org/html/rfc7667

[6] V. Singh, A. A. Lozano, and J. Ott, "Performance analysis of receive-side real-time congestion control for webrtc," in *2013 20th International Packet Video Workshop*, Dec 2013, pp. 1–8.

[7] L. D. Cicco, G. Carlucci, and S. Mascolo, "Understanding the dynamic behaviour of the google congestion control for rtcweb," in *2013 20th International Packet Video Workshop*, Dec 2013, pp. 1–8.

[8] S. Islam, M. Welzl, D. Hayes, and S. Gjessing, "Managing real-time media flows through a flow state exchange," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 112–120.

[9] D. T. Nguyen, K. K. Nguyen, S. Khazri, and M. Cheriet, "Real-time optimized nfv architecture for internetworking webrtc and ims," in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, Sept 2016, pp. 81–88.