CORE
Metadata, citation and similar papers at core.ac.uk
Provided by UPCommons. Portal del coneixement obert de la UPC

Media Streams Allocation and Load Patterns for a WebRTC Cloud 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 Web Real-Time Communication (WebRTC) is seeing a rapid rise in adoption footprint. This standard provides an audio/video platform-agnostic communications framework for the Web build-in right in the browser. The complex technology stack of a full implementation of the standard is vast and includes elements of various computational disciplines like: content delivery, audio/video processing, media transport and quality of experience control, for both P2P and Cloud relayed communications. To the best of our knowledge, no previous study examines the impact of Cloud back-end load and media quality at production scale for a media stream processing application, as well as load mitigation for Cloud media Selective Forwarding Units. The contribution of this work is the analysis and exploitation of *server workload* (predictable session size, strong periodical load patterns) and *media bit rate* patterns that are derived from real user traffic (toward our test environment), over an extended period of time. Additionally, a simple and effective load balancing scheme is discussed to fairly distribute big sessions over multiple servers by exploiting the discovered patterns of stable session sizes and server load predictability. A Cloud simulation environment was built to compare the performance of the algorithm with other load allocation policies. This work is a basis for more advanced resource allocation algorithms and media Service Level Objectives (SLO) spanning multiple Cloud entities.**

***Keywords*-load measurements, webrtc, rtp/rtcp, media, bit rate, stream allocation, simulcast, load balancing, simulcast**

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. The API nature of WebRTC in web browsers makes it possible to easily go beyond basic conferencing use cases and allow applications to blend with media communications in ways that had not been possible before.

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). To coordinate among them and/or with the cloud, WebRTC/RTCWEB endpoits require a messaging infrastructure as well. WebRTC/RTCWEB, though, specifically leaves messaging out of its definition, allowing freedom of choice, and focuses on the range of communication protocols and technology stacks that take care of real time communications for media (audio and video) and data signaling. WebRTC/RTCWEB stack intended for 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/RTCP, among other, adds support for media source identification, media mixers, media track synchronization facilities, quality of service feedback or media bundling and multiplexing. 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 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

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

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes,creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI 10.1109/NOF.2017.8251214

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/RTCWEB is supported natively by major web browsers (e.g. Chrome, Firefox and Edge) and provides a free real-time communication medium. A pure P2P implementation of the standard has many limitations. Some of these limitations are: i) The upload bandwidth (and CPU usage) a client needs when sending the stream grows proportionally with the number of clients receiving such stream. ii) users behind firewalls or NATs may be subject to severe network restrictions and as such P2P direct communication may be not possible. iii) the rate of connectivity failure grows with the number of clients joining the session. iv) Additional operations like archiving (saving to a permanent storage) a session may be a challenge. These and other issues constitute a big problem for applications that need reliability, quality and cost effectiveness in common fixed or mobile networks with higher download and far lower upload bandwidths. Common middlebox/server

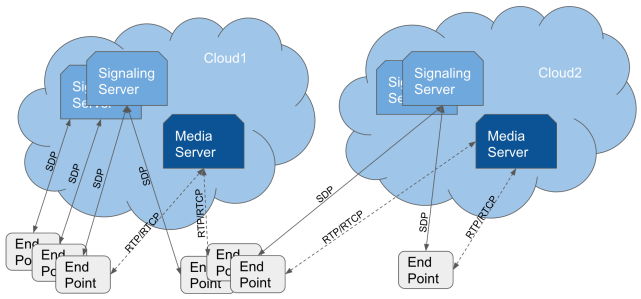

Fig. 1. System Overview

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. One of the most evolved forms of selective forwarding is the capacity to adapt media quality individually for every endpoint without conducting media translation when scalable/simulcast media encodings are used [6], [7]. In such a case, a sending endpoint (publisher) produce media streams composed of multiple qualities that can then be intelligently selected and forwarded by SFUs for each receiving endpoint (subscriber). Fig.1 shows the high level design of such an architecture.

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).

Network Quality of Service (QoS) is of utmost importance for these communication architectures. Also, one of the most important properties directly related to video quality is video bit-rate. For resilient real-time communications, bit-rate needs to adapt at every moment to the available resources in clients and network, and avoid dropping the 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 bit-rate received at the clients, as well as on the impact of rate-control with single layer encoded video if compared to simulcast[7].

In order to satisfy resource needs, enough SFU units need to be provisioned for all sessions in a communications cloud platform. This work assumes the definition of server load to be the number of streams managed by each SFU, and since the servers are cloud machines within defined categories we can benchmark maximum allowed streams per server for each category. This metric appears to be more reliable than other resource metrics and can be easily sampled for each server. The allocation of streams directly impacts load factors as well as it can translate into media quality (such as bit-rate).

The first contribution of this work, is the study at ”production scale” of stream load patterns (per SFU), and other system parameters (topology). With these, one can devise automatic algorithms to predict resource needs and enforce Service Level Objective (SLO) limits. As a second contribution, we provide a sever selection policy that fairly distributes sessions among SFUs, minimizing the chance for server overload and thus not lowering media quality. The third contribution is a comparative study of media bit-rates(quality) for different session sizes and the use of simulcast (senders producing multiple qualities) or single stream adaptive approach where senders adapt bit-rate to the worst bandwidth available to a receiver (all using Google’s Congestion Control [8][9]). The last contribution of this work is the implementation of a simulation framework to simulate session allocation algorithms for WebRTC streams.

Our study differs from previous work in the field, by examining the behaviour of a Cloud SFU at production scale with real user traffic. We tackle the problem of how hundreds of thousands streams impact the overall quality of a generic media stream processing application over time, with hundreds of streams concurrently being handled by the same SFU.

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. Based on such workload analysis we provide in Sec. III a session based minimum load allocation algorithm for WebRTC/RTCWEB SFUs. We continue in Sec. IV by making a comparison of video quality for both simulcast and single stream rate-control over an increasing size of subscribers per publisher and how that can be impacted by session topology or size. Then following, in Sec. V, we present a comparison of load balancing algorithms for stream allocation (using minimal user information). The comparison is conducted through experiments ran on a cloud simulation environment based on CloudSim [10][11] and using real session traces from our testing environment. At last, we conclude in Sec. VI with final remarks on this work, as well as a description of the future work toward multi-cloud resource allocation for WebRTC/RTCWEB backends.

II. Load Characterization

Load characterization is an impacting factor in devising resource allocation strategies for a distributed service. The scenario we are investigating is composed of a distributed software media multicast, real-time, delay non-tolerant in a subscribers/publishers cloud delivery system. The SFU is the multicast backend while the subscribers and publishers are the consumer and producer clients. Table. I shows the distribution of the 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 range of $44 - 177$.

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

A very interesting trait of such distribution is that the maximum number of streams reached per machine is 5x-10x times higher than the respective means. This aspect is very important as once the first stream is allocated to a server, the following streams associated to the same session need to be allocated on the same server. This study is built with the assumption that a session does not span multiple servers in order to gain simplicity, while different sessions can be allocated in different server or clouds. In turn, big sessions are not allowed to be migrated to different servers, which makes resource scheduling a critical task for a cloud backend. 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.

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 make the observation from the 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 exposes a strongly auto-correlated sequence. Clustering of values around the diagonal represents a strong positive auto-correlation. Data Center centric load as such

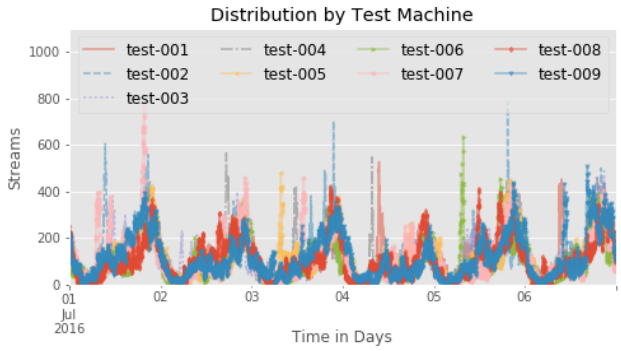

Fig. 2. Data Center/Server Load Distribution 7days period

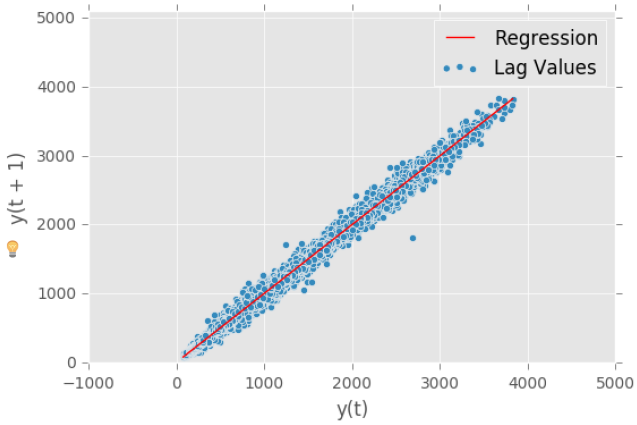

Fig. 3. Data Center total load lag plot 1month period

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 $intercept = 2.8282$. 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. The predictability of patterns in server loads for the cloud SFUs that was observed in this section can be exploited to provide better session allocation algorithms. This is exactly the purpose of Alg. 1 which tries to balance the load on the SFUs by allocating new sessions to underutilized servers.

$$Y_{i+1} = 0.9974 * Y_{i-1} + 2.8282 \quad (1)$$

III. Load Balancing Algorithm

In this section we present a stream allocation algorithm that distributes as much as possible peak sessions without previous knowledge of session sizes. In a Platform as a Service deployment architecture, knowledge of individual users connecting to the platform is restricted and user anonymity must be guaranteed. Such restrictions in information handling

introduce limitations in the stream allocation algorithms. In practice each stream gets a unique identification string chosen at random on creation, and as such no historical data can be aggregated for a defined end user of the platform. Predicting session size may be difficult given that minimal individual user information is available. The allocation policy for each stream should try to offload peaks to different servers as much as possible with only server load-centric historical information.

Algorithm 1 Min Load Server Allocation Policy

1: **procedure** SERVER_SELECTION(S_{Loads}, $radius$, max)
   // ***S_{Loads}[i]***, *array of structures* $[(srv\_id, load)...]$
   // ***radius***, *Minimal set selection radius*
   // ***max***, *Maximal allowed streams per server*
2: $S_{Loads} \leftarrow sort\_ascending(S_{Loads})$
3: $S_{MinSet} \leftarrow [\,]$
4: $srv\_selected \leftarrow Nil$
5: **for** $(srv\_id,\ load)$ **in** S_{Loads} **do**
6: **if** $load < max$ **&&**
7: $((load - S_{Loads}[0].load) < radius)$ **then**
8: $S_{MinSet}.add(S_{Loads}[i])$
9: **else**
10: break;
11: **end if**
12: **end for**
13: **if** $len(S_{MinSet}) == 0$ **then**
14: $srv\_selected \leftarrow allocate\_new\_server()$
15: **else**
16: $index \leftarrow rand\_int(0,\ len(S_{MinSet}) - 1)$
17: $srv\_selected \leftarrow S_{MinSet}[index].srv\_id$
18: **end if**
19: **return** $srv\_selected$
20: **end procedure**

The server selection policy, part of the Minimal Load selection algorithms, presented in Alg. 1 allocates incoming streams to the machines with current minimal load in a determined time window. Load statistics normally lay within a time window that can be deduced from the data in order to take into account subsequent big sessions arrival time. Lines $2-4$ initialize the state of the selection algorithm, then in the following lines $(5-12)$ a subset of the currently allocated servers with minimal load are selected. The minimal set entries do not differ more than ***radius*** stream utilization from the absolute minimal server load in the system. To conclude, lines $13-17$ select either one of the target subset servers at random with min load cardinality, or allocates a new server to expand the system. We will observe in Sec. V that the family of Minimal Server load algorithms performs better in terms of maximal server load during operations, through a series of simulation based on real usage trace data from our test data center.

Fig. 4 shows the distribution of the peak sessions for the servers of the test data center over a period of 2 days, we can

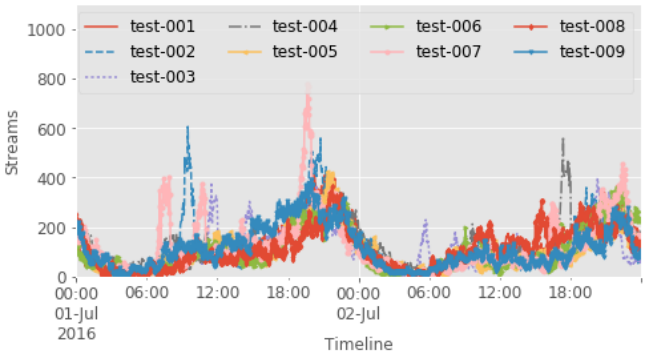

Fig. 4. Max Servers Loads for 2min Intervals

observe that by applying the min load policy, we manage to distribute the big sessions over the different servers and thus protect the system from allocating consecutive big sessions on the same machines. This property is seen as a result of big sessions being rare and their inter-arrival time being sporadic.

The algorithm presented is a simple but efficient way to handle and distribute the load between servers in a fair way. Alas other parameters need to be taken into account before this algorithm is production ready. The decision to either start new servers when existing resources are not available could be changed to a mixed solution between admission control and resource allocation to keep resource cost at bay. Another limiting aspect of the algorithm is that if the rate of incoming subsequent big sessions is lower that the rate at which users join the session, multiple big sessions would end up on the same server. Those sessions on the same server would exhaust the resources available on the server and provide quality issues. The randomization part of Alg.1 (lines $16-17$) tries to tackle exactly this problem by allocating streams on different servers randomly within the subset of servers with minimal load. In practice, one can assume that the rate of incoming big sessions is far lower than the rate at which users join existing sessions. Under these assumptions, the algorithm even though simple, performs quite well. Later in this work (Sec. V) we will further examine other allocation algorithms and compare their performance with Alg.1 through a simulated environment based on real session traces.

IV. Non/Simulcast impact on quality

In case of non-simulcast video streams, given that only a single layer encoded stream is available, the publisher shall adapt bit rate to match an estimate of the worst subscriber bandwidth or some lower limit called **Minimal Bit rate**. On the other hand, simulcast publishers send all the available video qualities directly without limiting or adapting bit rates (as long as upload bandwidth allows). In the case of simulcast, the SFU chooses for every subscriber which quality it shall receive, matching at best, the available bandwidth. From such definitions, we can see that as more clients are subscribing to the same stream simulcast provides more flexibility, and permits to keep average bit rate more stable, avoiding the

worst subscriber effect of pure packet relaying topologies, as *bad* subscribers typically are few.

The results of this section are derived from client side measurement using our test SFU-based cloud over a period of 14Days. We sample data for different distributed test machines in order to avoid bias on data due to geolocation. Bit rate estimations should not be affected by physical allocation of cloud resources as they are sampled on all connections at the same time. We focus here on VGA quality for simplicity.

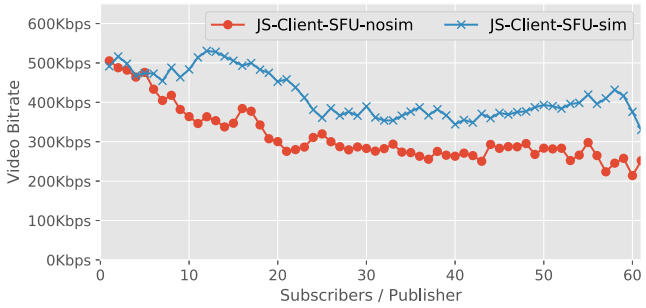

Fig. 5. Average Bit rate for #Subscribers/Publisher

Inspecting the impact of topology on the average bit rate, we see that with an increase of #subscribers/publisher non-simulcast adapts the bit rate to match either the minimum bit rate limit of the system or to the lowest performing subscriber; thus, all subscribers being penalized. This behaviour is not optimal for sessions with a high number of subscribers. In Fig. 5, we observe that in general bit rates for simulcast (on the same platform) are equal or better than non-simulcast. They are, at most, comparable just for low number of subscribers/streams (at around 2 subscribers). As commented, the more the number of subscribers, the more the bit rate profile falls for non-simulcast, giving rise to the the patterns shown in Section. IV. With the increase of load we have higher number of sessions per machine and as the number of #subscribers/#publishers increases on those sessions the average bit rate falls further to accommodate the lower performing subscribers up to a minimum bit rate threshold. As such it is not only the load in itself that leads to a lower bit rate but the topology of the sessions as the presence of sessions with more than 3 #subscribers/#publisher already start impacting bit rate in a non-simulcast session.

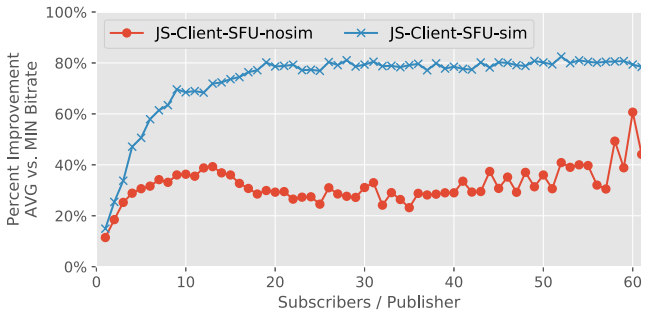

Fig. 6. Percent improvement AVG over MIN bit rate for #Subs/Pub

A good factor to further examine the impact of simulcast and non-simulcast publishers on the quality of the streams, is the relative difference of average set of bit rates to the minimum bit rate perceived by the worst of the subscribers from the same publisher. This parameter would present the relative perceptual improvement of the average bit rate over the minimum. In general simulcast outperforms non-simulcast by 2 times and also provide more stable bit rates as the number of subscribers increases. This behaviour is captured in Fig. 6, where we have the percent improvement over the number of #subscriber/publisher.

Simulcast performance can be tempting as a way to increase stability and scalability, but for low number of subscribers, like the case of One-to-One sessions, encoding and transmitting all video qualities may be a waste of client and network resources. In the case of mobile devices, simulcast may be beyond the possibilities of some hardware models, as it can easily take a 30% more CPU resources than the non-simulcast scenario for the same nominal resolution. This is only a small limitation at the moment of writing, as with the projected growth of mobile device CPUs, and with the inclusion of video encoder/decoder in the hardware architecture it will lead to more efficient encoding/decoding.

On a One-to-One user session the non-simulcast would not only need lower CPU usage but also would use the available bandwidth between the two users more efficiently as only one resolution layer is sent from the publisher. In WebRTC / RTCWEB, by means of Session Description Protocol (SDP) renegotiation, it is specified that it should be possible for endpoints to switch between simulcast and non-simulcast. However, some platforms may not support switching video transmission mode, thus knowing when to enable or disable simulcast becomes a very important decision or trade-off point. This further motivates our study of bit rate patterns and the results provided in this section can be used exactly to tackle this problem and drive trade-off decisions.

V. Load Balancing Evaluation and Experimentation

As a convenient way to experiment with session and stream allocation policies and have a coherent and repeatable way to compare them, we implemented a SFU workload simulation framework. Such simulation framework also lowers experiment costs and duration. Further experimentation was conducted by comparing the performance of various allocation algorithms, and in turn verify the efficiency of the proposed algorithm (Alg. 1). To drive the comparative analysis of such stream allocation policies, a dataset of traces of real WebRTC sessions was extracted from our test cloud. These traces[2] were used as simulation events for the resource allocation algorithm of a simulated cloud environment. The various load allocation techniques are built so that no user information outside of stream parameters provided in the WebRTC[4] standard are used. A simulation environment is used to compare the various

[2]Traces can be provided on-demand by the authors.

session/streams allocation algorithms as it permits faster and way less expensive means of experimentation with different allocation scenarios. A real deployment would have high resource costs for the both cloud SFUs and Clients as well as take longer to run.

A. Experiment Setup

Experiments on the performance of the various stream allocation policies were conducted in a simulated cloud environment, using task parameters to mimic the real *Test Cloud* environment. The simulation environment was implemented by extending the CloudSim [10][11] platform with a custom package to handle WebRTC streams. Normal operations of the CloudSim framework support batch processing tasks, we have coded in our extension package the ability to process WebRTC streams defined by the following parameters: ***Stream Identifier***, ***Session Identifier***, ***Stream Start Time***, ***Stream Duration***, ***Allocated Server Identifier***. The extended simulation environment supports the creation of multiple data centers, each one with an arbitrary number of Hosts, on which a single Virtual Machine (**VM**) is allocated. The package permits the configuration of the number of Hosts/VMs and also an upper limit on the number of streams that can be processed without SLO violations. During simulation run-time each VM can process in parallel as much streams as configured by the maximum streams parameter. Once the simulation is running if the number of streams surpasses the maximum configured number per VM, the additional streams will be executed in a time-share manner with the existing streams on the machine. In case that such streams are running in a time-shared fashion, the completion time of these streams would be delayed causing a violation of the SLOs. This aspect of the simulation permits to directly detect the performance of the stream allocation strategies, not only in terms of number of streams processed by the server, but also as reduced quality streams that would not meet real-time requirements.

For our experiments we have sampled one week of real system traces from our **Test Data Center**. We have selected a period with data concerning both normal operational loads and a particular day with a sudden spike in load behaviour. This data ensures that we have a general view of the implemented algorithms for both spiked data and normal operations. Our simulation environment is made up of: **1 Data Center**, **9 hosts**, **9 VMs** and **196 325 WebRTC Streams**. First we examine an algorithm that mimics the same stream allocation algorithm as the original data, which is also our baseline for testing the simulated environment. By comparing this algorithm to the original data we prove that the simulation is valid and calculates the same load profiles as the real **Test Data Center**. Other algorithms were implemented based on: **Static Threshold**, *Minimal Load*, and as well as *Rotational* algorithms such as Round Robin variants. Each one of the implemented algorithms is compared to the others and the baseline. As previously seen and observed in Fig. 7, the distribution of session sizes follows a tail distribution in which the majority of the sessions have a small compact size while there is a long

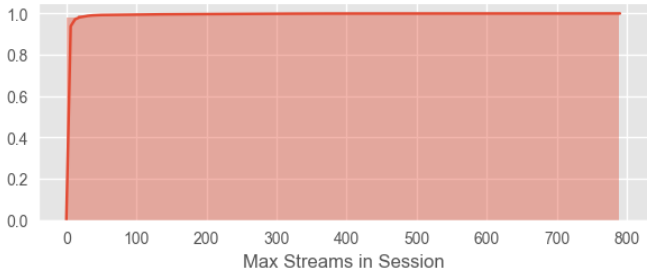

Fig. 7. CDF of Max Streams in Session

tail of big sessions which needs to be handled. The distribution shows that high spikes in session sizes or sessions with very high number of streams are rare and exceptional events. This makes it very unlikely for such sessions to have similar inter-arrival times and lower the impact of concurrent big sessions to the system.

B. Establishing a Baseline

Verification of the simulated environment was provided through the implementation of the **Preset Allocation** algorithm. The algorithm reads the input data and schedules the tasks to be processed at the same machine as the real system, with the same defined upper limit of **1 100** streams/machine and same **Stream Start Time**. The implemented baseline algorithm mimics the same configuration of the original system data sampled from the **Test Data Center**. We define in the context of this work the concept of SLO violation as streams which duration, or completion time is higher than the original duration of the stream.

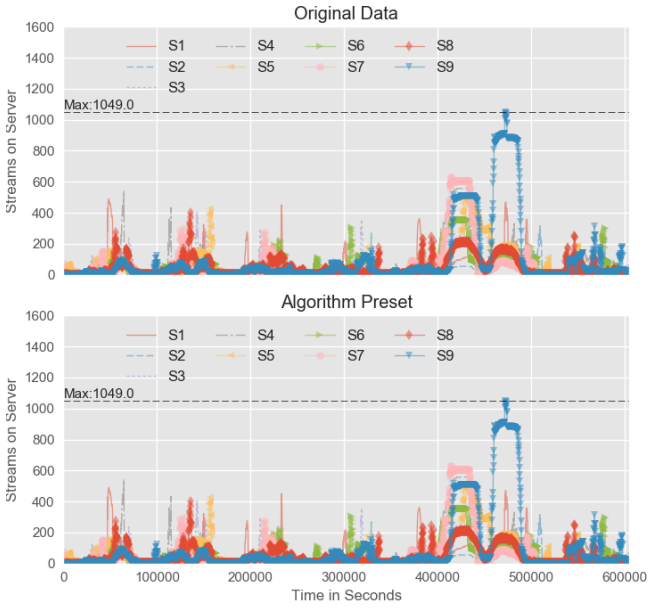

Fig. 8. Comparison of Original Data with Simulation

The simulation finished with no violations of SLO and both **Stream Start Time** as well as **Stream Duration** are identical

to that of the original data. The load profiles for both the original data and the simulation are presented in Fig.8, these profiles are totally identical. This data verifies our simulation environment and provides strong evidence of a stable and repeatable way for conducting further experiments.

C. The Minimum Load Algorithms

In order to try and minimize the maximum utilization in terms of streams in our servers, and lower the impact of peaks in form of big sessions, we experimented with minimum stream load algorithms. All the experiments were conducted by using the same parameters of our **Test Data Center** environment as the setup previously described when establishing our baseline (Sec. V-B).

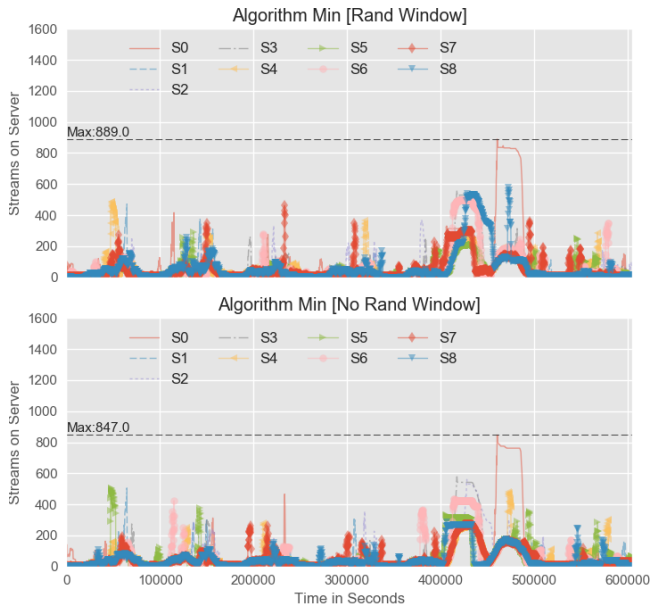

Fig. 9. Comparison with Minimum Load Algorithm

The first experiment provided is though Alg. 1 with a *threshold of 30* streams resource occupation difference for the minimal load set. Within the minimal load set, the selection of server where to allocate the next streams is done through randomly selecting one server candidate. We observe from our experiments that this algorithm already lowers the maximum observed utilization seen on all servers by 15%. This policy implements a minimal load algorithm with randomized selection windows and the results are presented in Fig. 9. Although this algorithm performs quite good, we can still improve over it by using a purely Minimal load selection algorithm. The algorithm plainly chooses for each new stream the server with minimal load without randomizing a set of minimal load servers. The results presented in Fig. 9 last subplot shows an additional improvement over the maximal load reached for all servers in the baseline, scored at 19% lower than the max utilization of the original data. Both algorithms provide a noticeable improvement over the other algorithms and thus were selected to tackle the problem of stream allocation.

D. Round Robin and Static Threshold

One other family of algorithms that were implemented and evaluated were rotation based algorithms implemented both as a per stream rotation and a threshold based rotation. The stream based rotational algorithm performs better than the original data and that the threshold rotation algorithms, but under-performs as compared to the minimal load algorithms. We observed that the maximal peak reached for all servers on the **Round Robin** stream allocation algorithm scores a maximal peak value of **925** and as such doesn't generate any SLO violations (Fig. 10). In these experiment results peak utilization do not reach the maximum number of streams per machine. Such numbers hold also due to the fact that in general a low number of #subs/#pubs dominates session sizes. The incoming rate of big sessions (more than **50** streams) is very rare but still impacts the overall performance. Clearly the impact of these rare big sessions is seen in the class of algorithms based on Static Threshold.

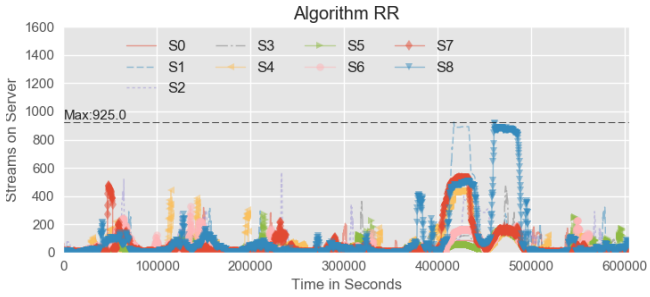

Fig. 10. Comparison with Round Robin Algorithm

The class of *Static Threshold* algorithms exhibits very big spikes in streams that do not only surpass the thresholds but as well the maximum number of streams that the machine can handle, creating SLO violations. We implemented a *Round Robin* variant, in which servers are not rotated until the resource occupation matches a given threshold. This technique has benefits as the servers needed to handle the load are in average less than the allocated number of machines. Even though we have a clear benefit in number of machines being used, the peaks reached during operations show an unacceptable side effect by producing a high number of SLO violations in terms of server overbooking. We show the result of such experiments in Fig. 11, where peak utilization of the servers reaches a values of **1 484**. The other implemented algorithm of the threshold family is the static threshold with minimal load utilization policy selection. After the threshold utilization for a server is reached the next stream is allocated to the server with minimal load. Both these algorithm are fragile and prospect to be impacted by big sessions assigned to a server that has reached near threshold utilization. As previously discussed in this case the peak utilization of such server reaches utilization factors of **134,9**%, which is unacceptable for real production scenarios.

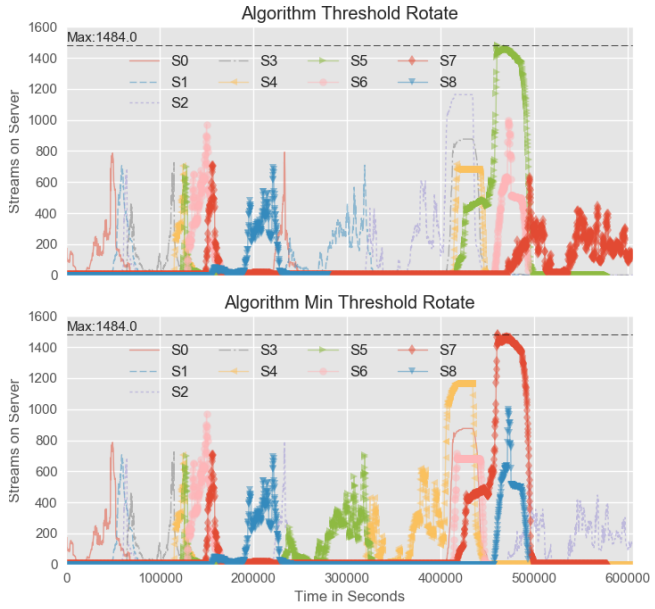

Fig. 11. Comparison with Static Threshold Algorithm Variants

VI. Conclusions

Previous work conducted in [9] deals with both performance aspects and bit-rate estimation algorithms, but the study is limited to a small controlled environment. Work conducted in [8] [12] explore the behaviour of multiple rate-control algorithms and streaming properties, as well as introducing novel rate-control algorithms. Resource allocation for WebRTC in [13] leads to a Network Virtual Functions based cloud architecture to interconnect IP Media Subsystems and WebRTC.

Building on these previous state-of-the-art, we provide insight into Cloud load patterns and load mitigation on a real, large scale WebRTC Cloud SFU system. Media performance is characterized through video bit-rates profiles for a SFU-based cloud supporting both simulcast and single source rate controlled video streams. We have shown from real client measurements that video simulcast provides benefits as big as 2 times higher than the minimum perceived bit-rate in the non-simulcast case, and that session topology size highly impacts media quality. Constructing on the discovered load and media patterns, we have provided a session allocation algorithm to fairly distribute big sessions among machines.

Different algorithms were evaluated in order to verify that the selected *Minimal Load* algorithm really behaves better in terms of peak utilization reached as compared to *Round Robin* and *Static Threshold* algorithms. All the algorithms that were discussed need minimal information on distributing the load and guarantee user anonymity toward the platform. This is an important aspect of running a Platform as a Service, and also in terms of insurances that can be given to both the users and the software implementers. We observe that the minimal load algorithms perform better than the rest of the observed algorithms and optimize peak utilization by exhibiting peaks of **19**% lower than the real allocation and up to **50**% lower than the static threshold algorithms.

Future work includes providing load allocation algorithms for multi-cloud deployment SFUs to optimize either cloud resources or user latencies. Further work can be conducted to investigate the alternate model in which sessions can span multiple SFUs and servers. This new system bypasses restrictions of single server session allocation treated here, but introduces new challenges on trunking and latency concerns.

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] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.

[7] The Internet Engineering Task Force. (2016, Oct.) Using simulcast in sdp and rtp sessions draft-ietf-mmusic-sdp-simulcast-06. [Online]. Available: https://tools.ietf.org/html/draft-ietf-mmusic-sdp-simulcast-06

[8] 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.

[9] 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.

[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. [Online]. Available: http://dx.doi.org/10.1002/spe.995

[11] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *2009 International Conference on High Performance Computing Simulation*, June 2009, pp. 1–11.

[12] 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.

[13] 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.