dOTM: A mechanism for distributing centralized multi-party video conferencing in the cloud

Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa, Javier Cerviño

Abstract—**One of the key factors for a given application to take advantage of cloud computing is the ability to scale in an efficient, fast and reliable way. In centralized multi-party video conferencing, dynamically scaling a running conversation is a complex problem. In this paper we propose a methodology to divide the Multipoint Control Unit (the video conferencing server) into more simple units, broadcasters. Each broadcaster receives the media from a participant, processes it and forwards it to the rest. These broadcasters can be distributed among a group of CPUs. By using this methodology, video conferencing systems can scale in a more granular way, improving the deployment.**

Keywords-**multimedia, videoconference, mcu, cloud**

I. Introduction

The surge of Cloud Computing systems has allowed developers to improve scalability on traditional services. They facilitate deployment of these services to companies that could not afford it before due to the high costs of other traditional infrastructures by charging only for the resources used.

Almost at the same time, video conferencing systems have rapidly evolved to be available as part of web pages instead of stand-alone native applications. Hence, users now can easily communicate with others who are viewing the same web pages from their web browsers. Different technologies, like Adobe Flash[1] and, more recently, HTML5[2] and WebRTC [1] have allowed this evolution towards the web.

The convergence of these developments has enabled new types of systems that offer videoconference as a Cloud service. Taking advantage of cloud technologies, they can adapt to large variations in the number of users. This is particularly important in videoconference between multiple participants. Here, communication usually occurs in virtual rooms, conceptual places in the Internet where users can, at least, see and listen to the others in the same room.

Traditionally videoconference systems have been developed around a central device, called MCU (Multipoint Control Unit). MCUs have been used for years in these systems to address signalling and real-time transport of user's video and audio contents. However, they can also support advanced functionality like recording, video and audio transcoding, video composition and audio mixing. Actually, MCUs commonly serve many virtual rooms with multiple users in each of them. Hence, Cloud videoconference services should focus on solving the problem of adapting the number of MCUs to a varying number of users.

In this paper we first establish a common ground by defining several concepts and topics related with MCUs, virtual videoconference rooms, and the different layers that take part during the communication. Then we introduce the main problem that many MCU developers face when implementing a scalable service. From there, we describe our solution, primarily based on dividing the MCU into components that run different tasks. This tasks are designed so they can run in a logically isolated way. Once we have this isolated tasks, we can distribute them among different virtual machines.

We have focused the solution in the most illustrative example, which is the scenario where the MCU forwards each video and audio stream from every user to the rest of participants in the same room. The main task in this case is to copy every packet and send it to all participants. And we have also defined the component that is responsible for running this task within the MCU: *OneToMany* processor. Our solution divides a MCU into multiple *OneToMany* components, one per audio and video stream that need to be sent to the users. Moreover, we define the details we need to take into account for different communication layers as well as general considerations to manage all the components in this system.

We also describe the limits and challenges that we identify in this solution. We provide solutions for these challenges, based on reducing the server's boot time, increasing the scalability of MCUs for an unlimited number of users and commenting how we can apply similar solutions to other kind of tasks performed by the MCU such as recording, or transcoding.

The paper is structured as follows: in Section II we introduce similar solutions and the limits they presented that prevented us from using them to scale MCUs in the Cloud. In Section III we define the scenario and problem described previously, and then in Section IV we propose a novel solution based on dividing the MCU into tasks that can be isolated. Finally, in Section V we comment the main conclusions that we obtained from this work and our proposal for future works.

[1]http://get.adobe.com/es/flashplayer/

[2]http://dev.w3.org/html5/spec/

II. Related Work

Since the term Cloud Computing was coined and its characteristics enunciated, there has been plenty of studies on how to take different kind of applications *to the Cloud*. In [2] the authors provide a definition on Cloud computing and also identify the main obstacles and opportunities for growth, the ability to scale quickly is among them. Our objective is to apply the principles described in this work to the videoconference world.

The illusion of infinite computational process and bandwidth that the Cloud provides is very attractive for deploying complex video conferencing services. These services, usually rely on MCUs to interconnect clients avoiding peer-to-peer scenarios, especially when the number of participants starts to grow. The capabilities of deploying new MCUs on demand is highly sought after.

Several works have tried to take advantage of the capabilities the Cloud provides. In [3] the authors provide a general overview on video conferencing services in the Cloud. By designing the conference like as service oriented architecture, the final consumer application is separated from the video conference provider. Conferencing services are offered by request in a similar way infrastructure is offered in the Cloud. We build on top of that idea. In [4] the authors present an implementation that adheres to some of the principles described in the previous work. While in these works the general idea on how to deploy videoconferencing systems in the Cloud is clear, it is not specified how to react to the variations in demand. We provide a solution for those variations.

In [5], the authors present a video conferencing system using a protocol that takes advantage of intra-Cloud networks to forward traffic among peers. However, they do not rely on a distributed MCU, so traditional MCU operations like recording or transcoding are not supported at this point. In the architecture we present, recording and transcoding are contemplated.

The authors of [6] use Cloud deployed surrogates to assist in mobile video conferencing by unloading some of the more CPU intensive tasks such as transcoding, being able to exchange media streams among them. The focus of the work is not to replicate the capabilities of a MCU but to improve the experience of multi-party mobile users, so it does not provide a solution to the scalability of MCUs as such.

III. Scenario and problem description

Video conferencing with multiple participants has traditionally been facilitated by MCUs, as opposed to peer-to-peer solutions. The lack of availability of IP Multicast [7] for end-users impairs the use of p2p for multi-party real time communications as the amount of participants dictates the amount of upload bandwidth needed. In such cases, MCU-based application level multicast is used. The centralized architecture means there is a bottleneck in the server side but, on the other hand, the clients can save upload bandwidth as they do not have to send one stream to each participant.

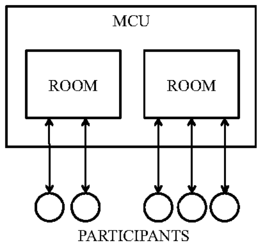

Fig. 1. MCU terms

New technologies provide video conferencing through web browsers. This enables a new, more dynamic way of communicating but also imposes a great task to MCU deployments as the traditional dedicated infrastructures are not tailored to meet the fast changes in demand associated with web applications. However, the architecture presented in this paper is general and does not only apply to web-based conferencing,.

The possibilities of Cloud Computing enable video conference providers to deploy as many MCUs as needed for a particular demand. However, the design of those MCUs can be optimized to avoid wasting Cloud resources.

First, we will define the general scenario. The final objective is to establish a multi-party video conference where an undefined number of users send and receive real time audio and video. These clients will not connect directly, all the traffic will go through a MCU.

A. Context

The following terms will be used throughout this paper. These are illustrated in Fig.1:

- *Participant*: A device that takes part in a video conference. It can receive and send multimedia streams. In general it will represent the final users of the system.
- *Room*: A conceptual environment where participants go to make their multimedia streams available and receive data from others. In general, we will say participants *publish* and *subscribe* to streams. The scope where these publications are visible is the room.
- *MCU*: A software device that hosts one or more rooms. It physically receives the media streams from the participants and is able to forward those streams to the others.

While there are many ways to design and implement and MCU, such as the ones seen in [8], in this paper we will focus on a pure forwarding MCU. That is, the MCU receives packets from each participants and forwards them to the rest, without performing any advanced operations in the streams themselves like composing or mixing. Only the essential operations for transmitting the media will be performed.

Furthermore, in the scope of this paper we well assume all the participants in a room subscribe to all the available media streams.

In the described scenario, the MCU acts in three different levels:

- *Signaling*: The MCU implements the required protocols that enable the negotiation required to establish the direct communication with the participants.
- *Media*: The MCU is able to send and receive media streams using the video and audio codecs and protocols used by the participants. These protocols are negotiated during the signaling phase.
- *Control*: The room concept is implemented in the MCU, allowing participants to publish and subscribe to media streams in that scope. This implies keeping a list of the participants in each room and being able to notify them when this list changes.

B. Scalability

In the server side, the MCU is the minimum complete piece of software that can be deployed and run on a node. Thus, it acts as the deployment unit when taken to a data center or a Cloud provider. That is, we will get more video conferencing capacity by adding more MCU nodes to the system.

In a public Cloud or a private data centre scenario we can set up architectures such as the one described in [9] to balance the load of rooms among the different running MCUs, or event start and shutdown MCUs on demand. In software terms, it is a process that is run on every server we want to have. As such, the amount of participants that can be handled by an MCU is limited by the hardware it is installed on and the bandwidth available to it. When we limit the amount of participants in one room we can estimate the resources needed for each room and plan accordingly.

However, if rooms get bigger, it can be deduced how being limited by the MCU as a unit will be a problem. For each participant in a room size N, the MCU receives 1 stream and forwards it $N-1$ times. In total, there are N^2 connections to the MCU at the same time. When the next participant comes, it publishes its stream, that is received by all the others $(1+N)$, and it subscribes to the available streams (N connections) so we have $2N+1$ new connections, $(N+1)^2$ in total. That is, the connections grow quadratically. The amount of processing power, memory and bandwidth needed is directly proportional to the number of active connections an MCU is maintaining. Thus, the amount of participants that an MCU can handle is much lower when they are arranged in big rooms.

To show the implications of the number of connections in the overall load in the system, we have performed tests in our open source WebRTC MCU [3]. This experiment was performed in a small instance in Amazon EC2 [4], roughly equivalent to 1GHz Intel Xeon family processor.

We reproduce the scenario detailed above. One conference rooms where all participants subscribe to all the streams and optimize the quality of the streams by maximizing the used bandwidth.

The clients provided 3 Mbps of traffic combining audio and video streams, for a very high quality video conference. The result can be seen in Figure 2

[3]www.lynckia.com/licode

[4]http://aws.amazon.com/ec2/

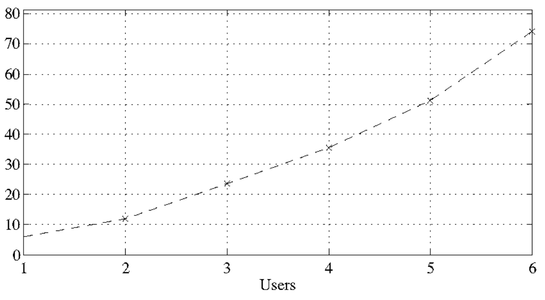

Fig. 2. CPU Usage per Participant

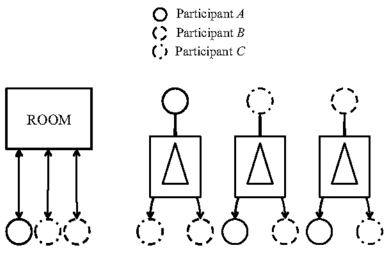

Fig. 3. Room as broadcasters

As we can see, the increase in CPU is very significant for each participant that joins the session. While the MCU is only copying and forwarding packets, the high CPU load is explained by the use of SRTP [10]. Each packet has to be decrypted and encrypted again to be transmitted to each of the participants as each one of the connection has its own encryption keys. This, combined with the very high bit-rate of the media, is very CPU intensive. A bigger Amazon instance would, obviously, allow for more users in the same conference, however, the problem persists.

If the size of the conference is not known beforehand we can be limited by the CPU when a participant arrives making it impossible to further increase the number. Besides, allocating lowly populated MCUs in powerful machines can be a waste of resources.

In the following subsections, we propose a mechanism that allows to distribute a single room into several MCUs, allowing us a more granular control of the scalability in the system. We will do this by separating the media layer of the MCU from the control layer and dividing the media processing into more simple units.

IV. DOTM

To be able to divide the MCU into smaller components, we go back to the basics of the tasks the MCU has to perform.

A room is a scope where participants can publish and subscribe to real-time streams. Each of these rooms can be seen as described in Figure 3, a set of broadcasters that take the input from one participant and forward it to the rest of the room. Thus, in a room with three participants, there will be three broadcasters with two subscribers each. These broadcasters are fully independent and do not have to know of the existence of the others. However, the total delay from and to the participants is in the same rage of hundreds of milliseconds to make the communication viable.

An MCU is a set of rooms, thus, we can build a full MCU with these smaller units. From that simple idea we divide the MCU into this broadcasters, we are going to call *OneToManys* (OTMs). Conceptually, these OTMs are subdivisions of the MCU and can be run anywhere, even on different infrastructures. The main difference is their smaller footprint, they only forward packets from one client.

This atomic unit provides the name for the solution. The full architecture is named distributed OTMs (dOTM) as it is built by an array of distributed units.

In the following subsections we will detail this separation by going through the three layers of the MCU: media, signaling and control.

A. System Description

1) Media layer: The media layer is the most intuitive. We will implement the logic for forwarding packets into separate processes. The processes can be started individually and will be in charge of receiving a stream from a single participant, copying the packets and forwarding them to the subscribed participants.

In the example seen in Section III, a room with N participants can be understood as N OTMs each with $N-1$ subscribers. If a new participant joins, a new OTM is created for this new entry and all the rest of the participants will subscribe to it, while the rest of the OTMs will have one new subscriber. So, while in the global picture the amount of connections still grows in a quadratic progression, each OTM's connections grow linearly.

Being OTMs independent processes that can be deployed individually on demand, we can distribute a full MCU across a set of servers. These servers can be located in any infrastructure and allocated dynamically.

These nodes do not have to be equal so we can adequate the number of OTMs deployed in each machine depending on its capabilities achieving a more efficient use of the available resources. As opposed to a fully centralized MCU, where a single node has to be able to process a full room.

2) Signaling layer: In the fully centralized scenario there is a single access point for every participant in a room. Now, potentially, each participant would have to negotiate the connection details to a different server node in order to publish and subscribe to the media streams. We will assign the problem of directing the participants to the OTMs to the Control Layer. A full signalling stack has to be implemented in each OTM so the media negotiation can take place.

For instance, when a client wants to subscribe to a stream, the control layer will know which deployed OTM is able to provide that stream. The client will be informed of the IP of that OTM so it can begin the actual signalling to establish the media session directly.

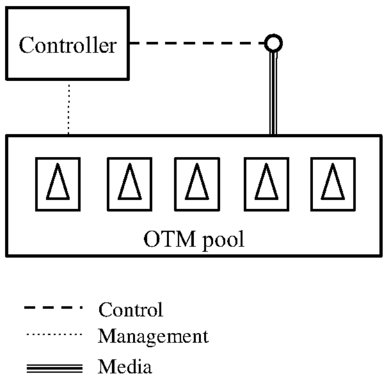

Fig. 4. Control, Media and signaling

3) Control Layer: In order to establish a conference, the OTMs have to be deployed intelligently according to the resources available, the logical rooms have to be distributed in this OTMs and participants have to be able to find the OTMs that are supposed to broadcast their streams and provide them with the other participants media.

To solve this problems, we introduce a new component in the system that will emulate the control layer of the centralized MCU, the Controller. This new unit takes over three main tasks:

- Room Management. All the logic concerning rooms will be located here. A room will be a set of participants and their associated OTMs.
- OTM Management. This includes starting and stopping OTMs remotely as well as monitoring their states. In case of any error in any of the OTMs, this layer would also react by starting another one and notifying all the involved participants.
- Resource monitoring. This component also implements the monitoring of the available hardware and network resources in order to be able to efficiently allocate OTMs.

In Figure 4 we can see the communication in the architecture. The clients enter the system via the Controller. The Controller manages the OTM pool, starting and stopping OTMs as needed and redirects the clients to the right OTM to publish or subscribe to streams.

This component is the new centralized entry point of the video conference system. Here we provide the behavior of the Controller when a new participant (Bob) joins a running room with N participants already communicating:

1) Controller receives the petition to join the selected room from Bob. Controller adds bob as a participant in the room.
2) A new OTM is launched by OTM in an available server. This new OTM will be in charge of forwarding Bob's stream to the other participants.
3) Controller provides Bob with the access point to the new OTM, once the signaling is successful, Bob starts sending media to the distributed MCU.

4) The rest of the participants are notified of the new OTM. Each of them join the OTM as a new subscribers. Once the signaling is over, all the participants receive Bob's media.
5) Controller provides Bob with the list of all the OTMs running in the room. Bob joins each of them a subscriber.

This new centralized unit presents a new bottleneck in our architecture as it is a central component that will receive requests from all the client. However, as opposed to the real-time communication oriented MCU, the controller is request-response oriented. Techniques valid for request-response servers (such as http servers) can be applied here and should be studied.

B. Deployment considerations

dOTM is designed to be deployed within a private or public Cloud infrastructure. The Controller acts as the single entry point and is aware of the resources available in the system. It is also able to start or stop VMs in order to allocate more OTMs.

According to [11], Cloud virtual machines' startup time ranges from the 44 seconds to up to 800 seconds. This greatly impairs the ability of the Controller to start new OTMs on new machines on the fly as new participants want to join the video conferencing rooms. A waiting time of up to hundreds of seconds is not acceptable in any case.

One possible solution to this scenario is to use predictions similar to those proposed for Cloud providers [12] to optimize the preallocation of VMs and OTMs while trying not to waste resources. However, the added granularity of dOTM provides easier allocation of new OTMs in the wasted CPU power of already running instances, compared to allocating full rooms.

However, being OTMs single processes allow us to take advantage of systems like OSv [5] that aims to provide better performance for single applications on top of a hypervisor. This approach speeds up startup time enabling more dynamic OTMs distributions.

C. OTM trees

We have discussed the improvement is scalability from the centralized MCU to the distributed OTMs. However, the number of participants when using dOTM is still limited by the amount of connections a single OTM can handle, its fanout. Even though the growth in connections is linear (for each OTM), each new participant in the session implies a new subscriber to each running OTM. Enough participants will saturate the capacity of each single OTM separately.

Broadcast trees can be used to avoid this problem. Here, each tree of OTMs would act as the distributor of one participants' stream to the others in a similar way as peer-to-peer tree broadcast algorithms, except here, all the peers are in our infrastructure.

In this case the limitation of the fanout of the OTMs would be set by each individual node running a OTM instance while

[5]www.osv.io

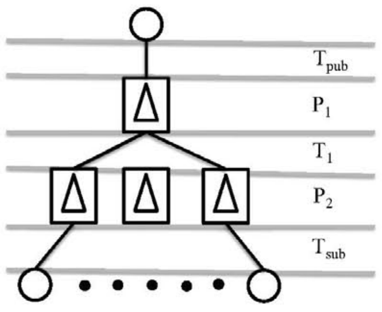

Fig. 5. Delay in OTM trees

TABLE I
DELAY IMPACT IN USER SATISFACTION

Delay (ms)	User Satisfaction
0 - 200	Very satisfied
200 - 280	Satisfied
280 - 390	Some users dissatisfied
390 - 520	Many users dissatisfied
> 520	Nearly all users dissatisfied

the depth of the node would be given by the amount of delay that participants can tolerate.

In [13], the International Telecommunication Union provides a recommendation for the total delay in voice communications over a network. It divides the delay in five levels as seen in Table I.

Accordingly, we would be able to add layers to the tree as long as we make sure the delay is under the 500 milliseconds threshold.

A brief study follows on how does this affect the depth of the OTMs trees. In this scenario, the delay of level n for a given subscriber participant is:

$$D_n = T_{pub} + T_{sub} + \sum_{i=0}^{n} P_i + \sum_{i=0}^{n-1} T_i$$

Where:

- D is the total delay at level n.
- T_{pub} is the transmission time from the publisher participant to the first OTM.
- T_{sub} is the transmission time from the last OTM to the subscriber participant.
- T_i is the delay from transmitting from OTM in level i to the OTM in level $i + 1$.
- P_i is the processing time for OTM in level i.

In Figure 5 we can see a tree topology formed by OTMs and the delays associated with this distribution.

This is common to all tree-based distribution architectures. However we can make some adjustments to optimize the delay in a OTM tree.

The processing time in each OTM depends on the operations performed over the streams. In the case exposed in Section III, the packets needed to be decrypted and encrypted again for each of the participants. In a tree, the decrypting has to be done by the level 0 OTM and the encrypting is only needed in the last level just before sending the data to the participants.

The rest of the levels could perform minimal operations on the packets so the processing time would be greatly reduced.

The transmission time among OTMs is given by the infrastructure where they are deployed. In a data center where all the nodes are connected in the same local network, we can assume the delay remains constant through the levels, so T_i is a constant T_{otm} and usually in the microseconds.

The OTMs can also be deployed in different geographical locations affecting the allowed depth of the tree. However, as explained in [14], the intra-Cloud networks perform better than the regular Internet so, when there are trans-continental communications this approach can provide benefits in terms of quality of the communication.

D. Advanced operations in the MCU

Until now, we have focused this study to an MCU that only forwards packets without performing any operations to the streams themselves. However, the division into smaller units can also be applied to MCUs that perform more specific tasks. Here we will apply the dOTM architecture to three of the most important.

1) Recording: Many video conferencing solutions offer the possibility to record the media streams of a given communication session. As the MCU is always receiving the published streams from all the participants it is often the device in charge of storing this streams.

Even though dOTM is a distributed MCU, the Controller has information of all the streams that are going through the system. In order to record the video and audio of a participant we propose to implement a modified "participant" , *recorder*, that is able to subscribe to a OTM and store the contents of the stream. As a result, each of these recorders produce a media file containing the publication of one participant.

However, instead of having all the recordings centralized as in a traditional MCU, the recorders can also be distributed in the available infrastructure. We propose using remotely mounted storage or Cloud storage solutions to save all the contents.

The Controller unit is responsible for starting these recorders and attaching them to the OTMs that need to be recorded depending on the configuration of the room. Also, as part of the monitoring task of the Controller, it has to keep track of the available disk space for storage in case there is a limit in that area.

2) Transcoding: MCUs are often used to change the characteristics of the streams in a conference in order to better adapt to the network and client's capabilities. The most common scenario is when one or more of the participants have a higher quality connection to the conference than the others. In this case, the bit-rate of the stream sent by these participants can choke the connection of the lower-bandwidth ones. This causes dropped packets that degrade the quality of the experience and in some cases can even completely break the conference. To avoid this, transcoding-able MCUs operate on the high-quality streams to lower their bit-rate to sacrifice individual steam quality while improving the conference as a whole.

With dOTM such mechanisms can also be implemented by modifying OTMs so that they can transcode the individual streams they are receiving before forwarding them to the other participants.

The Controller unit has to be aware of the characteristics of the network connections of the participants and decide when to start a new transcoding OTM and then command the low bandwidth participants to subscribe to this new stream. Thus, we would not replicate the transcoding work.

This does greatly increase the processing time for each stream as it has to be decoded and encoded again, that are expensive operations in terms of CPU. However, the ability to deploy it in individual instances in the Cloud helps us accommodate all this needed CPU power, at a cost.

3) Gateways to other technologies: Conferencing gateways such as the one in [15] are quite common in the video conferencing world. They allow the interconnection of two different video conferencing technologies with varying degrees of transparency.

While the implementation details can change depending on the two technologies that are to be interconnected, we will address the most common concerns in this translation scenarios and apply them to dOTM when interconnecting two different technologies, A and B.

- *Control:* The Controller has to be aware of the characteristics of technologies A and B. Furthermore, the participants have to be identified by the technologies they are using. With this information the Controller can use the modifications listed above to interconnect participants.
- *Media:* Usually different video conference technologies use different codecs. If that is the case, transcoding OTMs have to be used. The Controller has to set up the two type of transcoding OTMs, from the codecs in technology A to B and back.
- *Signaling:* OTMs have to be adapted to understand signalling from technologies A and B, once the do, they can act transparently in both enviroments.

This can be expanded to interconnect as many technologies as needed. The concept of room is in the Controller and is technology agnostic, as long as the technology supports multi-party communications. The flexibility offered by the OTMs also translates here as the core of the implementation does not change with each technology, only the initial signalization and media negotiation.

V. Conclusions and future work

In this paper we have presented the problem of achieving multi-party video conferences with a large number of participants. We have seen how big conferences have high requisites in terms of CPU power. In order to address this, we have designed a mechanism to distribute MCUs so they can be deployed in data centers or in cloud providers to take advantage of more computing resources. The mechanism acts by distributing MCU servers, dividing them into more simple parts. These parts, called OneToManys (OTMs), act as broadcasters where each participant can publish a stream that will be received by the rest of the session. We have named the full solution distributed OneToManys or dOTM.

The control logic of a centralized MCU is still present in a centralized control unit (Controller). The usual mechanisms used for scaling web servers can be applied to this Controller. We have upgraded our proposal of distributed MCU by addressing more advanced problems such as recording and transcoding. Also, we have shown how, by forming trees, we can avoid the bottleneck imposed by the fanout of each individual OTM, being the total depth of the tree limited in the server side by the delay caused by the transmission delays among the different OTMs.

Regarding future work, an statistical evaluation of the implementation of the solution will finish the validation of the proposed architecture. Also, the work need to be completed by designing the deployment of even more advanced MCU features such as video composing and audio mixing using OTMs. On the other hand, we have discussed the problem of scaling up a conference by adding more OTMs. The problem of scaling down is also interesting. Given the granularity dOTM provides when deploying, if a conferencing room is divided among many different computing nodes, fragmentation may take place as resources are freed. A methodology to deploy and mechanisms to rearrange OTMs on the fly should be investigated.

References

[1] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, "Webrtc 1.0: Real-time communication between browsers," August 2012.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] J. Li, R. Guo, and X. Zhang, "Study on service-oriented cloud conferencing," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 6. IEEE, 2010, pp. 21–25.

[4] P. Rodriguez, D. Gallego, J. Cervino, F. Escribano, J. Quemada, and J. Salvachua, "Vaas: Videoconference as a service," in *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, nov. 2009.

[5] Y. Feng, B. Li, and B. Li, "Airlift: Video conferencing as a cloud service using inter-datacenter networks," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE, 2012, pp. 1–11.

[6] Y. Wu, C. Wu, B. Li, and F. Lau, "vskyconf: Cloud-assisted multi-party mobile video conferencing," *arXiv preprint arXiv:1303.6076*, 2013.

[7] S. E. Deering, "Host extensions for ip multicasting," Internet RFC 1112, August 1989.

[8] M. Willebeek-LeMair, D. D. Kandlur, and Z.-Y. Shae, "On multipoint control units for videoconferencing," in *Local Computer Networks, 1994. Proceedings., 19th Conference on*. IEEE, 1994, pp. 356–364.

[9] P. Rodriguez, D. Gallego, J. Cerviño, F. Escribano, J. Quemada, and J. Salvachúa, "Vaas: Videoconference as a service," in *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*. IEEE, 2009, pp. 1–11.

[10] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)," RFC 3711 (Proposed Standard), Internet Engineering Task Force, March 2004, updated by RFC 5506. [Online]. Available: http://www.ietf.org/rfc/rfc3711.txt

[11] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 423–430.

[12] Y. Jiang, C.-s. Perng, T. Li, and R. Chang, "Asap: A self-adaptive prediction system for instant cloud resource demand provisioning," in *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, 2011, pp. 1104–1109.

[13] T. S. S. of ITU., *ITU-T Recommendation G.114: Transmission Systems and Media : General Recommendations on the Transmission Quality for an Entire International Telephone Connection : One-Way Transmission Time*. International Telecommunication Union, 1994. [Online]. Available: http://books.google.es/books?id=eQ9RHwAACAAJ

[14] J. Cervino, P. Rodríguez, I. Trajkovska, A. Mozo, and J. Salvachúa, "Testing a cloud provider network for hybrid p2p and cloud streaming architectures," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 356–363.

[15] J.-M. Ho, J.-C. Hu, and P. Steenkiste, "A conference gateway supporting interoperability between sip and h.323," in *Proceedings of the Ninth ACM International Conference on Multimedia*, ser. MULTIMEDIA '01. New York, NY, USA: ACM, 2001, pp. 421–430. [Online]. Available: http://doi.acm.org/10.1145/500141.500204