Dynamic media stream mobility with TURN

Álvaro Alonso, Pedro Rodríguez, Joaquín Salvachúa, Javier Cerviño

***Abstract*—Multi party videoconference systems use MCU (Multipoint Control Unit) devices to forward media streams. In this paper we describe a mechanism that allows the mobility of such streams between MCU devices. This mobility is especially useful when redistribution of streams is needed due to scalability requirements. These requirements are mandatory in Cloud scenarios to adapt the number of MCUs and their capabilities to variations in the user demand. Our mechanism is based on TURN (Traversal Using Relay around NAT) standard and adapts MICE (Mobility with ICE) specification to the requirements of this kind of scenarios. We conclude that this mechanism achieves the stream mobility in a transparent way for client nodes and without interruptions for the users.**

***Keywords*—cloud; MCU; mobility; videoconference; scalability**

I. Introduction

Over the last few years video conferencing systems have become more popular in Internet due to a series of new technologies. The first one is the Web, which allows users to communicate among them through their web browsers using applications that are hosted inside traditional web pages. Other technologies are the access and broadband networks, which have been dramatically improved during the recent years and, now, they provide connections of tens of megabytes to end users. And these users can be connected through wired or mobile networks, using PCs, smartphones, or even TV sets. Last, but not least, Cloud Computing systems promote scalability mechanisms to different kind of systems, including video conferencing platforms.

Commercial Cloud platforms that provide transparent scalability are mainly focused to web applications (Heroku[1], Google App Engine[2], Microsoft Azure[3]), while Cloud infrastructures allow more general applications (Amazon EC2[4], Google Compute[5], Rackspace[6]), but with more limited scalability features. This is the case of video conferencing systems that allow communication among multiple participants in the same virtual room. In these applications, video and audio streams are typically forwarded from every user to the others, and they all pass through a main component that is called MCU (Multipoint Control Unit).

[1]https://www.heroku.com/
[2]https://appengine.google.com/
[3]http://www.windowsazure.com/
[4]http://aws.amazon.com/ec2/
[5]https://cloud.google.com/products/compute-engine/
[6]http://www.rackspace.com/

These video conferencing systems could easily take advantage of Cloud infrastructures [1], especially when adapting to an increasing user demand. In that case these systems put more capacity in the Cloud by increasing the number of MCUs in the same virtual room. But they would also need to adapt to decreasing user demands by reducing the number of MCUs and reallocating the connections to the resulting MCUs. This second phase need the system to move existing connections from "deprecated" MCUs, which will be powered off, to other MCUs that the system will keep using.

Moreover, video conferencing systems put some requirements on the movement of connections between MCUs: video and audio streams should not be interrupted during the process, it should allow the system to consecutively move connections between deprecated MCUs and new MCUs, and it should be compatible with a wide set of heterogeneous videoconference clients.

In this paper we show a novel mechanism to tackle the mobility of connections between MCUs, which also resolves the strong requirements of video conferencing systems and works for the great majority of clients available on the Internet. This system is based on a draft specification, called MICE (Mobility with ICE), which extends TURN (Traversal Using Relay around NAT) and ICE (Interactive Connectivity Establishment) specifications for general mobility scenarios.

This is necessary because MICE standard does not completely resolve the case of scaling down the number of MCUs. Therefore, we show an adaptation of it including new attributes and parameters to the STUN messages. We also describe the mechanism of moving streams between MCUs in detail, avoiding interruptions in the communications and allowing the systems to decrease the MCUs with no delay. We also demonstrate why clients are not required to be adapted to this mechanism, making it possible to use in current video conferencing systems deployed in Cloud Computing platforms by only implementing changes in the MCUs.

We finally show the main conclusions of this work and how video conferencing systems can take advantage of this mechanism, and we also introduce future works based on the results of this paper.

In section II we identify the related work, introducing the advantages and problems we identified in similar systems and how we can adapt them to the new requirements. In section III we give details of the mobility problem and the different requirements of video conferencing systems. In section IV we explain how we adapt MICE with new attributes and messages, and we describe the message flow between the MCU and the

TURN device in detail. In section V we explain how this extension resolves the initial problem for different scenarios. And finally in section VI we show the main conclusions and we introduce the future work.

II. Related work

The authors have not found any related work that directly approach to the problem of media stream mobility between MCUs. However, there are several studies about peer mobility in real time communications scenarios. Most of them are mainly focused on device mobility in heterogeneous networks.

For instance, [2] proposes a new mobility service for IP Multimedia Subsystem (IMS) platforms. The service also analyse network parameters such us QoS (Quality of Service). In [3] is discussed a pre-negotiation mechanism for SIP (Session Initiation protocol) focused in cellular phones that move between networks. [4] approaches to mobility between different device network interfaces even using more than one at the same time. On the other hand, [5] defines a socket abstraction to the application layer in order to be able to add or remove sockets without disturbing the running applications.

These mechanisms can be applied to our scenario in which, definitively, the MCU acts as a peer in the communication implementing connectivity protocols such us SIP or ICE. However, a renegotiation may imply latency in communication and interruptions in the media received by the users. Furthermore, all these proposals imply specific implementations in all the peers and this can be a problem in scenarios with heterogeneous devices. For instance, in a web based videoconference system using the WebRTC (Web Real Time Communications) standard [6] it is not easy to modify the browsers implementation in order to support new protocols. The best solution is one in which the mobility is transparent to the peers and the only node that have to implement the new mechanism is the MCU.

MICE specification [7] proposes two mechanisms to achieve mobility. The first one is based on ICE specification [8]. They add STUN (Simple Transversal Utilities for NAT) attributes MOBILITY-EVENT and MOBILITY-SUPPORT in order to support mobility. So it suppose that all peers supports MICE, we have the same problem discussed above. On the other hand, the second mechanism uses TURN to achieve considering that only a peer supports the new specification. So it seems that MICE fits very well with our requirements. However, as we explain in the next section, this solution has some problems that we have to solve proposing an extension of the specification.

III. Media stream mobility scenario

As seen in the previous section, the media stream mobility solution that best fits to our scenario is MICE specification. However, we need to adapt it to this specific environment adding some characteristics and modifying some parts of its behaviour. In this section we describe in detail the architecture of our scenario and we explain why we need to extend MICE.

In a typical videoconference scenario there are one or more peers publishing its media streams in virtual spaces frequently called rooms. Another peers connected to the same rooms subscribe to these media streams in order to receive them. These peers are usually people that are sharing its video or audio streams but can also be other types of sources and sinks such as recorders or external media processors.

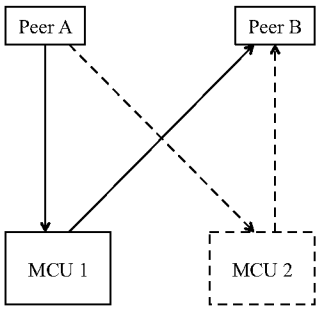

Fig. 1: Stream mobility architecture.

Definitely, from the point of view of an MCU this scenario implies that the MCU device receives a media stream from a peer, processes it and sends it to another peer. The process is a stream redirection in the simplest case but can be also an advanced task such us transcoding the stream or multiplexing it to a variable number of peers.

Fig. 1 shows an scenario where Peer A and Peer B are connected to MCU1 and ready to exchange data packets with it. Both peers may be connected directly to a public network or in a private network behind a NAT. In order to establish a connection with the MCU device they have used ICE protocol. Peer A is sending a media stream to MCU1 and MCU1 is redirecting it to Peer B. And our goal is to move the management of that media stream from MCU1 to MCU2. As explained in the introduction, this mobility is very useful in scalable scenarios, where we are going to power off MCU1 and MCU2 has available resources. Thus, once the mobility process is finished Peer A will send the stream to MCU2 and MCU2 will redirect it to Peer B.

This is a basic scenario where a Peer A is only sending and not receiving and Peer B is only receiving and not sending. However, the problem is the same but duplicated if both peers send and receive data. And it is also valid for more complex configuration with more than two peers participating in the videoconference session. Each peer publishes its stream and the MCU forwards the stream to multiple subscribers. Thus, it is easy to perform on streaming and multi-videoconference scenarios. So here we are going to illustrate the simplest case and the solutions for other configurations will be extrapolations of this one.

As introduced in Section II, MICE specification provides two mechanisms to perform endpoint real time mobility between networks. The second one applies to scenarios where only one of the endpoints supports MICE. Our scenario is very similar to it because we want to modify the IP address of one of the endpoints of the communication, the MCU.

However, it is very important to remember that we are talking about real time scenarios in which the user experience must be the best possible. That means that when we move the media stream from MCU1 to MCU2, Peer 2 must not

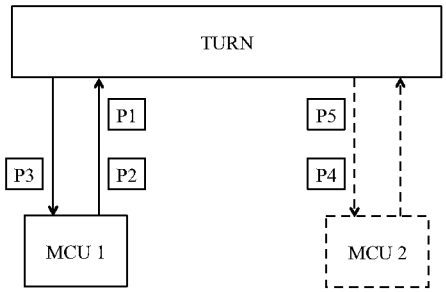

Fig. 2: Packet loss example.

experiment any interruption in the media reception. And it makes MICE not valid itself.

The mechanism proposed by MICE extends TURN specification [9]. It consists in refreshing a TURN allocation with a new parameter (MOBILITY-TICKET) that indicates a variation in the client IP address/port. Thus, TURN server modifies the 5-tuple associated to the allocation replacing it with a new one that contains the new IP address/port. Then, all data packets coming from the relayed address/port corresponding to that allocation will be sent to the new client IP address/port. On the other hand, all application data coming from the new client IP address/port will be sent through that relayed address to the peers. But at the moment of updating the 5-tuple some packets may have been sent to the old client and they will not be processed by the TURN server when returning because then the old client's 5-tuple will not exist in the server.

Fig. 2 shows an example that illustrates this problem. TURN server is sending data packets to MCU1. Just before sending packet 3 (P3 in the figure) the 5-tuple of the allocation is changed from MCU1 to MCU2. So packet 4 is sent to MCU2. But packets 1, 2 and 3 are still being processed by the MCU and they have not arrived yet to TURN server. When they arrive to TURN server the 5-tuple of MCU1 will not exist and they will be ignored. This results in a packet loss and consequently in an interruption in the media receipt by the peers.

IV. MICE EXTENSION

In this section we propose a MICE extension to solve the problem illustrated in Section III.

Coming back to the example shown in Fig. 2, our proposal is to keep both 5-tuples (of MCU1 and MCU2) for the same allocation until all packets sent from TURN to MCU1 return to TURN server. But the fact that a single allocation has two 5-tuples must only imply that TURN server can receive application data from two different clients and send it through the same relayed IP address/port. However, in the other direction, when it receives new data through the relayed IP address/port, it only sends it to the client corresponding to the new 5-tuple.

Furthermore, it may happen that the system requires a new mobility from MCU2 to a third MCU before MCU1 has finished sending application data to TURN server. Thus, the new MCU (MCU3) will refresh the allocation adding its 5-tuple and keeping the 5-tuples of both MCU1 and MCU2. This behaviour allows quick and dynamic media stream mobility. Therefore, a single allocation may have associated one traditional 5-tuple and multiple special 5-tuples that only work in one direction, receiving application data from clients and sending it to peers through the same relayed IP address/port. We have called that special 5-tuples *deprecated-5-tuples*.

They are *deprecated* because their life time must be limited. Due to security reasons it is not recommended to have that type of 5-tuple alive indefinitely. So TURN server will delete a *deprecated-5-tuple* from an allocation in a limited time period. This period is specified in a new configuration parameter of the TURN server, *deprecated-5-tuple-LIFETIME*. Anyway, the client who owns a *deprecated-5-tuple* can delete it from an allocation at any time by sending a *Refresh Request* with LIFETIME attribute with a value of 0. If TURN server does not want to support *deprecated-5-tuples* it has to set *deprecated-5-tuple-LIFETIME* parameter to 0.

As introduced above, MICE defines a new STUN attribute called MOBILITY-TICKET. A client who wants to support mobility has to include this attribute when creating an allocation. Then, TURN server generates a ticket and includes it in the same attribute of the *Allocate Success* response sent to the client. When the client wants to refresh an allocation retaining the same relayed IP address/port but with a new client IP address/port, it has to include the ticket in the *Refresh Request*. TURN server validates the ticket and if everything success it retains the relayed IP address/port for the new client IP address/port.

We define the new STUN attribute SHARED-MOBILITY-TICKET that, if included in the *Refresh Request*, indicates the TURN server to keep *deprecated-5-tuples* when refreshing an allocation. In our example, MCU1 obtains a SHARED-MOBILITY-TICKET when allocating and sends it to MCU2 before making the media stream mobility. MCU2 will include it in the *Refresh Request* to ensure the mobility process. Therefore, SHARED-MOBILITY-TICKET is designed for be shared between clients in order to support mobility between them. Moreover, both MOBILITY-TICKET and SHARED-MOBILITY-TICKET are compatible. A client can request both tickets when creating an allocation. With the first one it will be able to move between networks and with the second one it will move streams to another client.

How the exchange of the SHARED-MOBILITY-TICKET is made between clients is out of the scope of this paper. However, it is recommended doing it in a secure way in order to avoid man in the middle attacks.

To perform the mobility successfully it is also recommended the new client sends a notification to the old client when it receives the *Refresh Success* response. Then, the old client can delete its *deprecated-5-tuple* by sending a LIFETIME of value 0 in a *Refresh Request*). Otherwise, packets could be lost because TURN server stops receiving packets from the old client before establishing the connection with the new one.

With all these considerations MICE and TURN specifications result as follows:

A. Creating an Allocation

1) Sending an Allocate Request: In addition to the process described in Section 5.1.1 of [7], the client can also include the SHARED-MOBILITY-TICKET attribute with length 0. This indicates the client is a node that requires media stream mobility between clients and wants a ticket.

2) Receiving an Allocate Request: In addition to the process described in Section 5.1.1 of [7], server checks if the SHARED-MOBILITY-TICKET attribute is included. If its length is 0 and TURN mobility between clients is forbidden by local policy (by setting TURN configuration parameter *deprecated-5-tuple-LIFETIME* to 0) the server must reject the request and send back the new *Shared Mobility Forbidden* error code. If the server can not understand the SHARED-MOBILITY-TICKET, it ignores the attribute.

If the server can successfully process the request proceeds as explained in MICE, creating a ticket and including a STUN SHARED-MOBILITY-TICKET attribute with the encrypted ticket in the success response.

3) Receiving an Allocate Response: The process is the same as the one described in Sections 5.1.3 and 5.1.4 of [7] for *Allocate Success* response and for *Allocate Error* response. But in this case the client stores the SHARED-MOBILITY-TICKET (not the MOBILITY-TICKET) and receives the error code Shared Mobility Forbidden instead of error code Mobility Forbidden.

B. Refreshing an Allocation

1) Sending a Refresh Request: If a client wants to refresh an existing allocation with a new time-to-expire or wants to delete an existing allocation it will proceed as described in Section 7.1 of [9]. If a client wants to make media stream mobility between clients it will include the SHARED-MOBILITY-TICKET obtained by the client who owns the allocation when creating it. This SHARED-MOBILITY-TICKET is valid only one time so in each refresh transaction the client has to use the ticket of the last one.

2) Receiving a Refresh Request: When the TURN server receives a *Refresh Request* with a SHARED-MOBILITY-TICKET attribute it checks the validity of the ticket as explained in Section 5.2.2 of [7] for MOBILITY-TICKET and sends the corresponding error code when something is wrong in the validation.

If the *Refresh Request* contains a desired life-time value not equal to 0 and the 5-tuple retrieved from the packet corresponds to a *deprecated-5-tuple* of an allocation, the client is attempting to refresh a *deprecated-5-tuple* and TURN server must reject the request with an error.

If all the checks pass, TURN server understands that the client wants to perform mobility between clients and it gets the 5-tuple from the SHARED-MOBILITY-TICKET instead of from the request. It search for the allocation associated to that 5-tuple and adds that 5-tuple to the *deprecated-5-tuples* list of the allocation. Then the 5-tuple that identifies the allocation is the 5-tuple of the new client, extracted from the request. TURN server calculates a new ticket with the new 5-tuple and sends it in the STUN SHARED-MOBILITY-TICKET as part of *Refresh Success* response.

From this moment all the data received through the relayed IP address/port of the allocation will be sent to the client of the new 5-tuple. However, TURN server will process application data from all clients whose 5-tuples are *deprecated-5-tuples* of the allocation sending it to peers through the same relayed IP address/port. So when receiving application data TURN server has to check not only the 5-tuples it its database, also the *deprecated-5-tuples* of the allocations.

If the *Refresh Request* contains a desired life-time value equal to 0 and the 5-tuple retrieved from the packet corresponds to a *deprecated-5-tuple* of an allocation, TURN server deletes that tuple from the allocation.

If the *Refresh Request* contains a desired life-time value equal to 0 and the 5-tuple retrieved from the packet identifies an allocation that has one or more *deprecated-5-tuples*, TURN server deletes the allocation including all the *deprecated-5-tuples*.

3) Receiving a Refresh Response: The process is the same that the described in Section 5.2.3 of [7] but in this case the client stores the SHARED-MOBILITY-TICKET.

C. Allocations

In addition to the state data described for TURN allocations in Section 5 of [9], we describe a new field that contains a list of data structures with the information about the *deprecated-5-tuples* associated to the allocation. Each data structure consists of the following state data:

- the deprecated-5-tuple: (client's IP address, client's port, server IP address, server port, transport protocol)
- the time-to-expiry

The time-to-expiry is the time in seconds left until the *deprecated-5-tuple* is deleted from the allocation. When stored, the time-to-expiry is set to the time configured in the parameter *deprecated-5-tuple-LIFETIME*. As explained above, a client can delete a *deprecated-5-tuple* from the allocation by setting this parameter to 0 with a *Refresh Request*.

The list of permissions and channels are not included because they are the same of the 5-tuple to which the *deprecated-5-tuple* is associated.

Therefore, the state data of a TURN allocation results as follows:

- the relayed transport address
- the 5-tuple: (client's IP address, client's port, server IP address, server port, transport protocol)
- the authentication information
- the time-to-expiry
- a list of permissions
- a list of channel to peer bindings
- a list of deprecated-5-tuples data structures

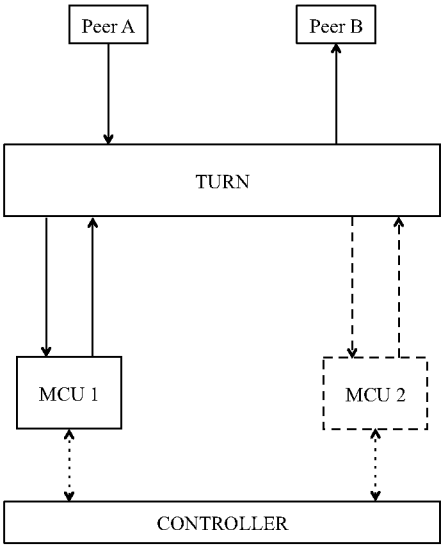

Fig. 3: Single allocation configuration.

D. New TURN configuration parameter

TURN server has a configuration parameter *deprecated-5-tuple-LIFETIME* that indicates the life time of the *deprecated-5-tuples*. When a 5-tuple is added to the list of *deprecated-5-tuples* of an allocation, the 5-tuple will be alive until reaches the time specified in this parameter or until the client deletes it by refreshing with desired life-time equal to 0.

E. Deleting expired deprecated-5-tuples

TURN server periodically checks if exists any *deprecated-5-tuple* with a life time higher than the set in TURN *deprecated-5-tuple-LIFETIME* parameter. If exists, it means that *deprecated-5-tuple* has expired and TURN server deletes it from the corresponding allocation.

F. New STUN Attribute SHARED-MOBILITY-TICKET

In addition to STUN attributes defined in [10], this attribute is used to perform media stream mobility between clients in TURN servers. It is exchanged between TURN server and clients to aid mobility. It has the same encryption and security properties that MOBILITY-TICKET described in Section 5.3 of [7].

G. New STUN Error Response Code

In addition to STUN error codes defined in [10], the new error code *Shared Mobility Forbidden* indicates that a mobility between clients request is valid but it is not allowed due to policy restrictions.

V. Solution

With the solution proposed in the previous section we are able to solve our initial problem, the media stream mobility between MCUs. In this section we explain how this MICE extension applies to a videoconference scenario. And we do that in two configurations. The first one is a basic configuration in which the MCU uses a single TURN allocation for all the streams and the second one is a more complex scenario with more than one allocation per MCU.

A. Single allocation configuration

Fig. 3 shows a simple videoconference scenario in which Peer A is publishing a media stream to MCU1 and Peer B is subscribing to it. The behaviour is the same that the one explained in Section III. TURN server is used as a relay in order to allow media stream mobility with MICE. To force the communication through the TURN server Peers must ignore the ICE candidates that are not the relayed candidates of the TURN server. We are not going to modify peers implementation so in order to do that we will delete these candidates from the SDPs (Session Description Protocol) when making the negotiation between MCUs and peers.

In a determined moment we want to move the media stream from MCU1 to MCU2 in such a way that Peer A will publish its stream to MCU2 and Peer B receives it from MCU2. It happens because, for instance, we want to scale down the system turning off MCU1 machine, so we have to move the streams that it is managing to another MCU, in this case, to MCU2. In the architecture is included a module Controller that communicates MCU1 with MCU2 in order to control and coordinate the mobility.

Fig. 4 illustrates the message exchange between all the nodes of the architecture from the beginning of the communication and before establish any connection until the moment in which the mobility to MCU2 is completed. As introduced above, we suppose this is a basic configuration in which both MCUs use the same TURN allocation to send and receive data from peers.

In order to start the communication with the Peers, MCU1 requests an allocation to TURN server. It wants to support mobility between clients so it includes SHARED-MOBILITY-TICKET parameter with length equal to 0 in the *Allocate Request*.

If mobility between clients is allowed and TURN server successfully processes the request, it creates an allocation (*Allocation 1*) with the 5-tuple of MCU1 (*5-tuple 1*) and assigns a relayed IP address/port to it (*RelayedAddress 1*). It generates a SHARED-MOBILITY-TICKET for this allocation (*Ticket 1*) and sends an *Allocate Success* response to MCU1 including the ticket. MCU1 stores *Ticket 1* associated to the published stream.

Then, Peer A starts publishing its stream (*Stream A*) sending data to the relayed IP address/port of *Allocation 1*. TURN server gets the 5-tuple associated to this allocation, (*5-tuple 1*), that corresponds to MCU1, so it sends the data to MCU1. MCU1 processes the published data and multiplexes it to the subscribers, in this case Peer B. It sends the application data to TURN server encapsulating it in a STUN message. The best way to do this is using a TURN Channel Binding as described in Section 11 of [9]. TURN server processes the data from MCU1 (ie. *5-tuple 1*), searches the corresponding allocation (*Allocation 1*) and sends the data to Peer B through de relayed address/port associated to that allocation (*RelayedAddress 1*).

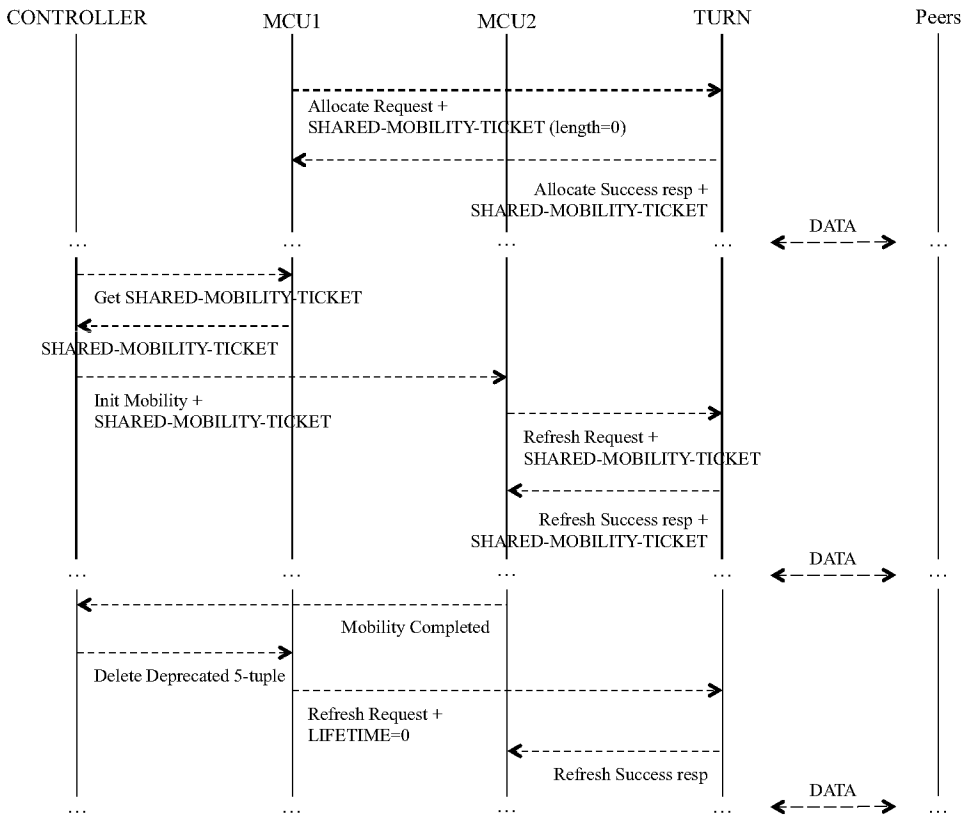

Fig. 4: Media stream mobility call flow.

In a determined moment of the communication, due to system requirements, Controller node decides to move *Stream A* from MCU1 to MCU2. It sends a message to MCU1 to get the SHARED-MOBILITY-TICKET of this stream. MCU1 searches the SHARED-MOBLITY-TICKET of *Stream A* and returns it to Controller. It is the ticket of *Allocation 1*, *Ticket 1*.

Then, Controller sends a message to MCU2 asking it to start the mobility of the allocation identified by *Ticket 1* (included in the message). As explained before, the protocol used to make the message exchange between Controller and MCUs is irrelevant in the scope of this paper.

Once received the ticket, MCU2 sends to TURN server a *Refresh Request* request including the SHARED-MOBILITY-TICKET attribute with that ticket, *Ticket 1*.

TURN server checks if the *Refresh Request* is valid following the steps described in previous section. If success, it extracts the 5-tuple from *Ticket 1*, (*5-tuple 1*) and searches it in its data base. It adds *5-tuple 1* to the *deprecated-5-tuple* list of the associated allocation (*Allocation 1*) and set as 5-tuple of the allocation the 5-tuple extracted from the request. This new 5-tuple (*5-tuple 2*) corresponds to MCU2. TURN server generates a new SHARED-MOBILITY-TICKET with the new 5-tuple (*Ticket 2*) and includes it in the *Refresh Success* response sent to MCU2.

Now *Allocation 1* has *5-tuple 2* as main 5-tuple and *5-tuple 1* as a *deprecated-5-tuple*. That means that data received in *RelayedAddress 1* is sent to *5-tuple 2* IP address/port (MCU 2). Data received from *5-tuple 1* (MCU1) and from *5-tuple 2* (MCU 2) IP addresses/ports is sent to peers through *RelayedAddress 1*.

When MCU2 receives the *Refresh Success* response from TURN server, it stores *Ticket 2* for a future mobility and sends a message to Controller indicating that the mobility is completed.

Controller waits a time in order to ensure that all packets sent before mobility to MCU1 has arrived to TURN server and then sends a message to MCU1 asking it to delete its 5-tuple from TURN server. So MCU1 sends an *Allocate Request* with LIFETIME parameter equal to 0 to TURN server.

TURN server extracts the 5-tuple from the request, *5-tuple 1* and searches it in its data base. It finds that 5-tuple as a *deprecated-5-tuple* of *Allocation 1* so it deletes it from the allocation. If the life time of that *deprecated-5-tuple* has previously expired due to *deprecated-5-tuple-LIFETIME* parameter of TURN server it has been already deleted from *Allocation 1*. TURN server sends a *Refresh Success* response to MCU1.

Now *Allocation 1* has not any *deprecated-5-tuple* so works sending data from Peer A in *RelayedAddress 1* to *5-tuple 2*

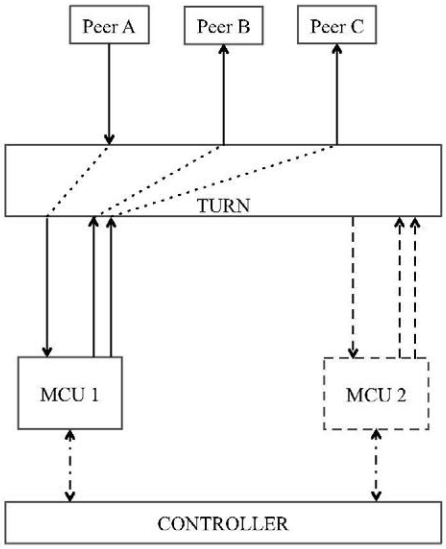

Fig. 5: Multiple allocation configuration.

IP address/port (MCU2) and receiving data from *5-tuple 2* IP address/port (MCU2) to Peer B through *RelayedAddress 1*.

When finished this process Peer A is publishing to MCU2 and Peer B is subscribing from MCU 2. The mobility is completed. As explained above, in this configuration we have multiple streams in the same TURN allocation, sending and receiving data through the same relayed IP address/port. But it is not always so, sometimes we need to move streams associated with more than one allocation and in this case we have to take into account some considerations.

B. Multiple allocation configuration

When performing stream mobility between MCUs it may happen that we have to move more that one allocation. In Fig. 5 is shown a configuration in which Peer A is publishing a stream and Peers B and C are subscribed to that stream. However, in this case the streams use different relayed IP addresses/ports, so different allocations (dotted lines in the figure). It implies that if we want to move that stream from MCU1 to MCU2 we have to move the three allocations.

The order in which the mobility of the allocations is made is very important. Calling *Allocation A*, *Allocation B* and *Allocation C* the allocations corresponding to Peers A, B and C respectively, if *Allocation A* is moved before moving *Allocation B* and *Allocation C* we will lose some packets. Packets received through relayed IP address/port of *Allocation A* from Peer A will be sent to MCU2 and when returning to TURN server *Allocation B* and *Allocation C* are sill waiting packets from MCU1. So will lose those packets.

The conclusion is that when we have to move more than one allocation between MCUs we must move first the allocations corresponding to the subscriber peers. Thus, supposing

- MCU1 has allocated Allocations A, B and C with relayed IP addresses/ports for communicating with Peers A, B and C respectively,
- allocations A, B and C are identified by *5-tuple 1A*, *5-tuple 1B* and *5-tuple 1C* respectively,
- those 5-tuples have the IP address of MCU1 and a different port of MCU1 each one,
- MCU1 has the SHARED-MOBILITY-TICKETS of Allocations A, B and C,
- those tickets are *Ticket A*, *Ticket B* and *Ticket C* respectively,

we have to made the following steps for achieve mobility in this configuration. They are simplified because it is supposed that the details of the messages are already known.

First, Controller asks MCU1 for the SHARED-MOBILITY-TICKETS of the subscribers, getting *Ticket B* and *Ticket C* and sends theme to MCU2.

Using these tickets, MCU2 sends two *Refresh Request* to TURN server and TURN server stores in *Allocation B* a 5-tuple of MCU2 (*5-tuple 2B*) setting *5-tuple 1B* as a *deprecated-5-tuple* of the allocation. The same process is done for *Allocation C* with *5-tuple 1C* and *5-tuple 2C*.

When MCU2 receives the success responses of these refresh requests TURN server is ready to receive data from MCU2 by the new 5-tuples and is also receiving data from MCU1 by the deprecated 5-tuples.

MCU2 notifies that to Controller and Controller proceed with the mobility of *Allocation A* getting *Ticket A* from MCU1 and sending it to MCU2.

When receiving the ticket, MCU2 ask TURN server for the refresh of *Allocation A* setting a 5-tuple of MCU2 (*5-tuple 2A*) as main 5-tuple of the allocation. So data received from Peer A is now sent to MCU2 and received by the previously enabled *5-tuple 2B* and *5-tuple 2C* in TURN server.

MCU2 receives the success response and notifies it to MCU1 through Controller. Finally, MCU1 ask TURN server to delete the deprecated-5-tuples of the three allocations.

Following these methodology we will achieve stream mobility between different clients in any scenario without packet loss.

VI. Conclusions and Future Work

In this paper we have proposed an advanced mechanism to perform media stream mobility between MCUs. This mechanism is based on MICE specification that allows mobility using TURN and ICE. However, we have discussed that in scenarios with strict real time requirements, MICE is not enough because using it may imply packet loss in the communications. Specially if the stream mobility is between different devices.

So we have extended MICE and TURN describing a new behaviour to achieve the stream mobility without packet losses. Moreover, our mechanism allows quickly and dynamic mobility, which is very useful in Cloud scenarios in order to scale videoconference systems. We have also applied the new mechanism to a videoconference scenario in two different configuration. The first one is a basic configuration in which the MCUs uses a single TURN allocation and the second one

is a more advanced configuration where there are more than one allocation. In this second scenario we have analyse some important factors to take into account in order to make the stream mobility successfully.

We conclude that we can taking advantage of this mechanism by performing dynamic media stream mobility between MCU devices for any videoconference scenario and without interruptions for the participants.

Regarding the future work, the next step is to characterise different videoconference scenarios using our MICE extension. It is very interesting to measure latency when moving streams between MUCs. And these measurements can be made using different configurations such us the explained in Section V. Thus, we can compare them and decide which is the better configuration to scale the system as fast as possible.

Going to an upper abstraction layer, a future line of research is to apply this mechanism to a real scalable videoconference system. Implementing the mechanism in the MCUs of our open source WebRTC project Licode [7] and deploying the system in Cloud Infrastructures we will analyse the improvement in cost, efficiency and performance.

REFERENCES

[1] A. Alonso, P. Rodriguez, J. Salvachua, and J. Cerviño, "Deploying a multipoint control unit in the cloud: Opportunities and challenges," in *CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization*, 2013, pp. 173–178.

[2] M.-I. Corici, A. Murarasu, S. Arbanowski, T. Magedanz, S. Lee, and X. Liu, "Multimedia mobility service solution," in *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, 2008, pp. 1–5.

[3] H. Mineno, M. Yoshida, and T. Mizuno, "Quick transfer of media stream in fmc environment," in *Innovations in NGN: Future Network and Services, 2008. K-INGN 2008. First ITU-T Kaleidoscope Academic Conference*, 2008, pp. 401–406.

[4] D. Maltz and P. Bhagwat, "Msocks: an architecture for transport layer mobility," in *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 1998, pp. 1037–1045 vol.3.

[5] J. Kristiansson and P. Parnes, "Application-layer mobility support for streaming real-time media," in *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, vol. 1, 2004, pp. 268–273 Vol.1.

[6] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, "Webrtc 1.0: Real-time communication between browsers," W3C," Working Draft WD, August 2012, http://www.w3.org/TR/webrtc/ [retrieved: March, 2013].

[7] D. Wing, P. Patil, and T. Reddy, "Mobility with ICE (MICE)," Working Draft, IETF Secretariat, Internet-Draft draft-wing-mmusic-ice-mobility-05.txt, Sept. 2013.

[8] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," Internet Requests for Comments, RFC Editor, RFC 5245, April 2010. [Online]. Available: http://tools.ietf.org/html/rfc5245

[9] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)," Internet Requests for Comments, RFC Editor, RFC 5766, April 2010. [Online]. Available: http://tools.ietf.org/html/rfc5766

[10] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," Internet Requests for Comments, RFC Editor, RFC 5389, October 2008. [Online]. Available:

[7] http://lynckia.com/licode