Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DISEÑO DE UN SISTEMA DE COMUNICACIONES EN TIEMPO REAL EN LA WEB Y SU ESCALABILIDAD EN EL CLOUD

TRABAJO FIN DE MÁSTER

Álvaro Alonso González

2013

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISEÑO DE UN SISTEMA DE COMUNICACIONES EN TIEMPO REAL EN LA WEB Y SU ESCALABILIDAD EN EL CLOUD

Autor
Álvaro Alonso González

Director
Joaquín Salvachúa Rodríguez

Departamento de Ingeniería de Sistemas Telemáticos

2013

Resumen

Los sistemas de videoconferencia han sufrido un gran impulso en los últimos años. El desarrollo de las comunicaciones de datos inalámbricas así como la abundancia de dispositivos móviles de cada vez mayor potencia y capacidades multimedia ha propiciado que la popularidad de las comunicaciones en tiempo real a través de Internet aumente considerablemente.

Entre estos avances destaca, dentro del marco del estándar HTML5, la irrupción del estándar WebRTC, que permite realizar comunicaciones de este tipo directamente entre navegadores web. Entre sus posibles usos destaca la comunicación de voz y vídeo como streaming y videoconferencias y los juegos online.

En este Trabajo Fin de Máster se describe el proyecto Licode, una plataforma para proveer comunicaciones en tiempo real en la web. Licode está diseñado para poder instalarse en cualquier infraestructura y está puramente orientado al Cloud. De esta forma podrá ofrecerse un servicio escalable y que ahorrará costes gracias a un uso eficiente de los recursos.

La descripción del sistema incluye el diseño de su arquitectura y la descripción de la funcionalidad de cada uno de los módulos necesarios. Además se realiza un análisis exhaustivo de la conveniencia de desplegar el sistema en infraestructuras Cloud. Este análisis describe las principales ventajas que el Cloud provee al sistema así como los retos más importantes a los que hay que enfrentarse para conseguir un despliegue eficiente y efectivo que haga un uso adecuado de los recursos disponibles en cada caso y acorde a la demanda de los usuarios.

Abstract

Videoconference systems have experimented an important impulse in the last years. The improvement of the wireless data communication and the increase of the mobile devices of much more performance and multimedia capabilities have stimulated the popularity of the real time communications in Internet.

One of these progresses is, inside de HTML5 standard frame, the irruption of the WebRTC standard that allows us to make real time communications between web browsers. Its possible uses are voice and video communications, such as streaming or videonferencing and online video games.

In this document is described the Licode project, a platform to provide real time communications in the web. Licode is designed to be installed in any computer or infrastructure and it is totally oriented to the Cloud. This way it offer a scalable service that saves costs and makes an efficient use of the available resources in the systems.

The project description includes the design of its architecture and the description of the functionality of each module. It is also made an exhaustive analysis of the convenience of deploy the system in Cloud infrastructures. This analysis describes the main advantages that Cloud provides to the system. Also the more important challenges that must be faced in order to make an efficient and effective deployment by making an appropriate use of the available resources and accord to the user demand in the service.

Índice general

Resumen i

Abstract iii

Siglas x

Agradecimientos xii

1 Introducción 13

1.1 Objetivos 14

1.2 Estructura del documento 14

2 Estado del arte 16

2.1 Sistemas de videoconferencia en la Web 16

2.1.1 Adobe Flash Player 16

2.1.2 VaaS 16

2.1.3 Google Hangouts 17

2.1.4 Big Blue Button 17

2.1.5 OpenTok 18

2.2 WebRTC 18

2.3 Cloud Computing 19

3 Arquitectura 21

3.1 Nuve 23

3.1.1 MAuth 24

3.1.2 API de cliente 25

3.1.3 Servidor 29

3.1.4 La Base de Datos 33

3.2 Erizo 34

3.2.1 Wrapper Node 35

3.3 Erizo Controller 37

3.3.1 API de Cliente 37

3.3.2 Servidor ... 45
3.4 La configuración de Licode ... 50
4 Despliegue en el Cloud ... 51
4.1 Oportunidades ... 51
4.1.1 Escalabilidad a la demanda de usuarios ... 52
4.1.2 Escalabilidad a los requisitos de la sesión ... 53
4.1.3 Flexibilidad geográfica ... 54
4.2 Retos ... 55
4.2.1 Caracterización del sistema ... 55
4.2.2 Escalabilidad hacia arriba ... 59
4.2.3 Escalabilidad hacia abajo ... 60
4.2.4 Distribución geográfica ... 61
5 Resultados ... 63
5.1 El proyecto de código libre ... 63
5.2 La comunidad ... 64
5.3 Proyectos ... 64
5.3.1 FI-WARE ... 64
5.3.2 Telefónica España MSC5 ... 64
5.4 Investigación ... 65
6 Conclusiones ... 66
6.1 Trabajo futuro ... 66
Bibliografía ... 69

Índice de figuras

Figura 1. Arquitectura del sistema 21
Figura 2. Topologías de red 35
Figura 3. Ejemplo de intercambio de eventos 44
Figura 4. Varias MCUs 54
Figura 5. Distribución geográfica de MCUs 55
Figura 6. Uso de CPU en Streaming 56
Figura 7. Uso de ancho de banda en Streaming 57
Figura 8. Uso de CPU en Videoconferencia 58
Figura 9. Uso de ancho de banda en Videoconferencia 58
Figura 10. Proxy 61

Índice de tablas

Tabla 1. Interfaz Nuve API. Inicialización 26
Tabla 2. Interfaz Nuve API. Salas 27
Tabla 3. Interfaz Nuve API. Usuarios 27
Tabla 4. Interfaz Nuve API. Servicios 28
Tabla 5. Interfaz Nuve API. Tokens 29
Tabla 6. API REST Nuve 31
Tabla 7. Interfaz Erizo Controller API. Stream 39
Tabla 8. Interfaz Erizo Controller API. Room 41
Tabla 9. Interfaz Erizo Controller API. Event 43
Tabla 10. Configuración Licode 50

Siglas

API	Application Programming Interface
DOM	Document Object Model
DTLS	Datagram Transport Layer Security
GING	Grupo de Internet de Nueva Generación
HTML	Hiper Text Murkup Lenguage
HTTP	Hiper Text Transport Protocol
HMAC	Hash Message Authentication Code
IaaS	Infraestructure as a Service
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSON	JavaScript Object Notation
MCU	Multipoint Control Unit
NAT	Network Address Transversal
NoSQL	Not only SQL
PaaS	Platform as a Service
REST	Representactional State Transfer
RIA	Rich Internet Application
RPC	Remote Procedure Call
RTCP	Real-time Transport Control Protocol
RTMP	Real-time Transport Message Protocol
RTP	Real-time Trasport Protocol
RTT	Round Trip delay Time
SaaS	Service as a Service

SDP	Session Description Protocol
SHA1	Secure Hash Algorithm
SRTP	Secure Real-time Transport Protocol
URL	Uniform Resource Locator
WebRCT	Web Real Time Communications
W3C	World Wide Web Consortium

Agradecimientos

El proyecto Licode ha sido diseñado y desarrollado junto a mis compañeros Javier Cerviño y Pedro Rodríguez. Se ha realizado en el marco de desarrollo de proyectos de investigación del Grupo de Intenet de Nueva Generación (GING) bajo la dirección del Juan Quemada y Joaquín Salvachúa.

1 Introducción

En el Grupo de Internet de Nueva Generación de la Universidad Politécnica de Madrid, se ha trabajado en sistemas y aplicaciones de videoconferencia entre múltiples usuarios desde hace más de 20 años.

En un principio, estos programas de videoconferencia eran complejos programas difíciles de instalar y gestionar, especialmente para usuarios sin perfil técnico. Esto suponía una barrera de entrada que se sumaba a la disponibilidad de ancho de banda y la potencia de los terminales, en aquel momento, casi exclusivamente ordenadores de sobremesa.

La videoconferencia ha sufrido un gran impulso en los últimos años. El desarrollo de las comunicaciones de datos inalámbricas así como la abundancia de dispositivos móviles de cada vez mayor potencia y capacidades multimedia ha propiciado que la popularidad de las comunicaciones en tiempo real a través de internet aumente considerablemente.

Entre estos avances destaca, dentro del marco del estándar HTML5, la irrupción de WebRTC. WebRTC es un estándar que está siendo definido en colaboración entre el W3C (World Wide Web Consortium) y el IETF (Internet Engineering Task Force) para permitir la comunicación en tiempo real directamente entre navegadores. Entre sus posibles usos destaca la comunicación de voz y vídeo y los juegos. WebRTC es una tecnología inherentemente peer to peer donde únicamente se requiere un servidor Web para establecer la comunicación inicialmente mediante el intercambio de los mensajes de señalización.

No obstante, para casos avanzados de videoconferencia tales como: conversaciones con muchos participantes, clases distribuidas o eventos que necesiten grabación, es necesaria la existencia de un nodo central que reciba toda la comunicación procedente de los usuarios, la trate de la manera adecuada y la reenvíe si es necesario. Este componente central, además de ser un desarrollo nuevo, requiere que el sistema tenga cierta inteligencia, sobre todo a la hora de reaccionar frente a cambios dinámicos en la demanda.

1.1 Objetivos

El objetivo planteado en este Trabajo Fin de Máster es el de diseñar la arquitectura y funcionalidad de Licode. Licode, diseñado e implementado en el marco de desarrollo de proyectos del Grupo de Investigación, es una plataforma de comunicaciones en tiempo real en la web. Estas comunicaciones incluyen servicios de streaming, videoconferencia entre muchos usuarios y transmisión de datos en tiempo real. Están enteramente basadas en el nuevo estándar de la web HTML5 y WebRTC (Web Real-Time Communications). Además gracias a Licode podrán realizarse funcionalidades avanzadas como grabación de las sesiones o transcodificación para la adaptación a terminales móviles y tablets.

De esta forma cualquier desarrollador web que quiera incluir comunicaciones de este tipo en sus servicios y aplicaciones web podrá hacerlo de forma rápida y sencilla utilizando las APIs que Licode ofrece. Pero además podrá controlar totalmente todos los aspectos relacionados con el servicio ya que tiene la posibilidad de desplegar el servicio en sus propias infraestructuras.

Por último Licode estará orientado a su despliegue en infraestructuras Cloud de manera que se pueda aprovechar las grandes ventajas que ofrecen este tipo de despliegues. Estas ventajas están relacionadas con la escalabilidad del sistema frente a cambios dinámicos en la demanda por parte de los usuarios y frente a los distintos tipos de escenarios que pueden ser necesarios para las diferentes aplicaciones realizadas con Licode.

Para su diseño y desarrollo deberá estudiarse las características de cada módulo y establecer las tecnologías que mejor permiten su implementación.

1.2 Estructura del documento

Este documento está dividido en cinco capítulos:

1. Introducción: es el capítulo en el que nos encontramos. En él se hace breve descripción del problema que quiere resolverse y por lo tanto de los objetivos de este Trabajo Fin de Máster. Además se realiza una visión general de la estructura del documento y sus diferentes capítulos.

2. Estado del arte: en este capítulo se realiza una descripción del estado actual de algunas tecnologías, estándares y productos de relevancia o estrechamente relacionados con el tema que se trata en este Trabajo Fin de Máster.

3. Arquitectura: es el capítulo central de este documento. En él se describe a arquitectura general del sistema y se realiza una descripción exhaustiva de cada uno de sus módulos explicando su diseño, funcionalidad y las relaciones con el resto.

4. Despliegue en el Cloud: en este capítulo se hace una descripción de la conveniencia de desplegar un sistema como Licode en una infraestructura Cloud. Además se analizan los retos que este despliegue plantea y que se abordarán como trabajos futuros.

5. Resultados: en este capítulo se comentan los resultados que se han obtenido o piensan obtenerse a partir del desarrollo de Licode. Esto incluye los proyectos del departamento en que se ha utilizado, las líneas de investigación surgidas y la comunidad de software libre creada a su alrededor.

6. Conclusiones: último capítulo del documento en el que se explican las conclusiones del trabajo así como los trabajos futuros a realizar a partir de este momento.

2 Estado del arte

En este capítulo se hace una descripción de algunas de las tecnologías que tienen relación con el trabajo realizado y por lo tanto con el proyecto de comunicaciones en tiempo real en la web Licode.

2.1 Sistemas de videoconferencia en la Web

En esta sección se analizan algunas tecnologías o servicios existentes para ofrecer comunicaciones de tiempo real en aplicaciones web.

2.1.1 Adobe Flash Player

Adobe Flash es una plataforma RIA propietaria para añadir funcionalidades interactivas a las aplicaciones web tales como animaciones, vídeo, audio, etc. Flash Player tiene soporte para un lenguaje de programación interpretado conocido como ActionScript basado en el estándar ECMAScript. Desde su origen ActionScript ha pasado de ser un lenguaje muy básico a un lenguaje avanzado con soporte de programación orientada a objetos, comparable en funciones y uso al lenguaje JavaScript.

Originalmente creado para mostrar animaciones vectoriales en 2 dimensiones, ha pasado a utilizarsepara crear aplicaciones Web que incluyen flujo de audio y video e interactividad. La utilización de gráficos vectoriales le permite disminuir el ancho de banda necesario para la transmisión y, por ende, el tiempo de carga de la aplicación.

Actualmente Flash Player está disponible para las versiones más recientes de los navegadores más populares (Internet Explorer,Mozilla Firefox, Safari, Opera, etc.). El navegador Google Chrome no lo necesita porque viene incluido dentro de él.

2.1.2 VaaS

Vaas es un servicio de videoconferencia desarrollado por este mismo departamento. Se trata de un servicio multipunto en la nube basado en Isabelv5 que se apoya en un

servidor que de acceso a clientes tanto desde PC fijo/portátil, como desde terminales móviles.

El servidor de multivideoconferencia es capaz de interconectar PCs con terminales móviles, con acceso en ambos casos a través de navegador Web y sin necesidad de instalar aplicaciones. El sistema tiene las siguientes características:

1. El servidor se realiza adaptando Isabelv5, el gateway flash y otras tecnologías asociadas para soportar los siguientes terminales en la videoconferencia:

A. PC con navegador con soporte de video Flash. B. Terminal móvil capaz de emitir y recibir video en HTML5. C. Terminal móvil tipo iPad capaz de emitir y recibir video en HTML5.

2. Capacidad para soportar varias multiconferencias simultáneas e independientes. El número máximo de participantes se determinará en función de las características del servidor.

3. El sistema incluye una gestión mínima de multiconferencias.

4. El sistema incluye la gestión de estadísticas de códecs, uso de CPU, uso de BW, resolución, etc.

2.1.3 Google Hangouts

Google Hangouts un servicio de multiconferencia de Google que opera en web. No obstante también tiene disponible aplicaciones móviles aunque en este caso no soporta multiconferencia.

La versión web soporta hasta diez participantes de manera simultánea en la misma conversación. Además permite compartir vídeos de Youtube en las conversaciones.

2.1.4 Big Blue Button

BigBlueButton es una aplicación web open source para videoconferencia y e-learning o educación a distancia. Es un programa distribuido bajo licencia GNU y que ha surgido de la reutilización de proyectos varios como Asterisk, Flex SDK, Red5, MySQL y otros.

Pero además, BigBlueButton ofrece algunas características adicionales como la posibilidad de que sus usuarios suban archivos PDF o documentos de texto y hojas de cálculo. Ofrece tres roles en sus videoconferencias

- Presentador, que puede subir presentaciones y compartir su escritorio.
- Espectador, que no tiene autoridad en la videoconferencia y solo puede ver o chatear.
- Moderador, que puede subir presentaciones, compartir su escritorio y aceptar o expulsar usuarios.

2.1.5 OpenTok

OpenTok provee un API libre que permite a cualquier desarrollador web introducir características de chat y vídeo en grupo a sus sitios web. La plataforma está basada en la tecnología Adobe Flash y permite a los usuarios realizar sesiones de videoconferencia con chat y vídeo en vivo de hasta 20 participantes. Permite realizar invitaciones para usar el servicio a través de cuentas en las redes sociales Twitter, Facebook o MySpace.

2.2 WebRTC

WebRTC [A. Bergkvist et al. 2012] [S. Loreto et al. 2012] es un estándar en desarrollo por el Internet Engineering Task Force (IETF) y el World Wide Web Consortium (W3C) que pretende definir un framework, protocolos e interfaces de programación que proveerán comunicaciones interactivas y en tiempo real de audio, video y datos a las aplicaciones en los navegadores web.

Como se explica en [C. Jennings et al. 2013], la arquitectura en las aplicaciones WebRTC son peer-to-peer de tal forma que únicamente es necesario un servidor externo a los navegadores de los clientes para establecer la comunicación intercambiando los mensajes de señalización al comienzo de la misma. Los APIs de los que consta el protocolo son los siguientes:

- GetUserMedia: se utiliza para acceder a los recursos multimedia del usuario como su cámara o su micrófono. El resultado será un MediaStream.

- MediaStream: es un conjunto de pistas de flujos de audio o vídeo con los recursos multimedia del usuario.

- RTCPeerConnection: representa un canal de conexión con otro cliente. Un cliente añade a un RTCPeerConnection uno o más MediaStreams para compartirlos con el otro cliente.

Al añadir uno de estos MediaStreams a la conexión comienza la negociación con el otro cliente mediante la cual se negocian ciertos parámetros de la sesión como el formato de los datos (códecs de audio y vídeo), la resolución o el ancho de banda. Esta negociación debe realizarse mediante medios externos al protocolo y consiste en el intercambio de unos mensajes SDP del tipo offer/answer.

El protocolo implementa ICE para poder realizar la conexión en escenarios en los que los clientes se encuentren tras un NAT. Una vez establecida la conexión el transporte de datos se realiza sobre SRTP [M. Baugher et al. 2004], la versión segura de RTP y gracias a RTCP los clientes pueden obtener feedback sobre el estado de la conexión, las posibles pérdidas, etc. Además para la gestión de claves de SRTP se utiliza el protocolo DTLS-SRTP, una extensión de DTLS.

En la fecha en que se escribe este documento los navegadores en los que WebRTC está completamente soportado son Google Chrome y Mozilla Firefox en sus versiones de escritorio y Google Chrome en su versión para el sistema operativo Android.

2.3 Cloud Computing

Cloud Computing [Mell et al. 2011] es un modelo para proveer acceso bajo demanda a recursos de computación como redes, servidores, almacenamiento, aplicaciones y servicios de forma configurable y a través de la red.

Las características del modelo de Cloud Computing son las siguientes:

. *On-demand self-service:* Los usuarios pueden disponer de capacidades de computación (CPU, ancho de banda, almacenamiento, etc.) según lo necesiten.

. *Broad network access:* Estas capacidades están disponibles en la red en diferentes posiciones y son servidas a través de mecanismos estándar.

. *Resource pooling:* El Cloud sigue un modelo *multi-tenant* por el cual pueden asignarse recursos a diferentes usuarios.

. *Rapid elasticity:* Las capacidades pueden ser provistas y liberadas de forma automática para adaptarse a la demanda de los usuarios.

. *Measured service:* Los recursos son controlados de forma automática y monitorizados y reportados por mecanismos de medida.

Algunos de los proveedores de Infraestructuras Cloud (IaaS) existentes en la fecha en que se escribe este documento son los siguientes:

- Amazon AWS
- RackSpace
- CloudSigma
- Windows Azure
- GoGrid
- Terremark vCloud
- Joyent

3 Arquitectura

En esta sección se explica la arquitectura general de este sistema, Licode. Los diferentes módulos que son necesarios para montar un servicio Licode, la función y características de cada uno de ellos y sus relaciones.

En la Figura 1 puede observarse la arquitectura del sistema.

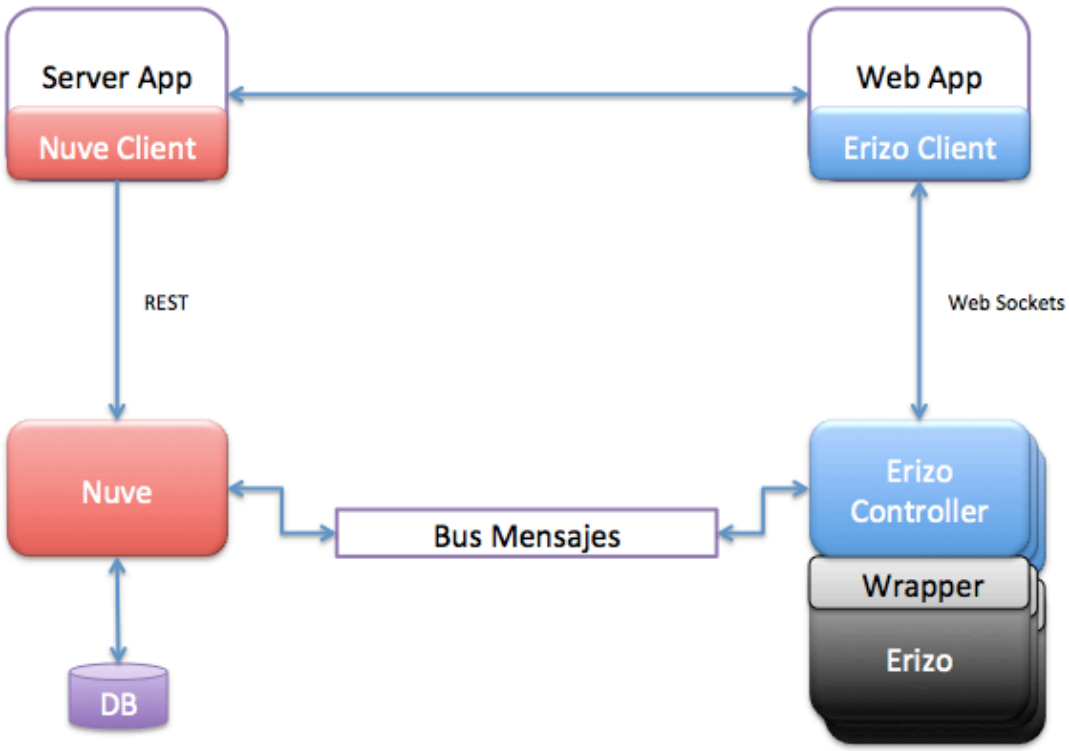

Figura 1. Arquitectura del sistema

Los módulos que conforman esta arquitectura son los siguientes:

- Nuve
  Es el módulo que se encarga de la gestión de los recursos que son necesarios en una sesión multimedia como salas, usuarios, etc. Consta de una parte servidor que se arrancará como un proceso Node y que recibirá peticiones REST de la parte cliente que se incorporará como parte de los servidores web de las

aplicaciones que utilicen Licode. Para la persistencia de los datos utiliza una base de datos MongoDB.

- Erizo
  Es la MCU del sistema. Se encarga de redireccionar los flujos multimedia de unos clientes a otros.

- Erizo Controller: Es el controlador de Erizo. Realiza este control a través de un wrapper que adapta los diferentes lenguajes de programación utilizados. utilizados. Además dispone de una parte de cliente con la que se comunica a través de websockets. Esta parte cliente será la que los navegadores web cargen en sus aplicaciones para realizar acciones sobre una sala multimedia controlada por Erizo Controller.

Como puede observarse en la figura en un servicio Licode puede haber más de una MCU (Erizo + Wrapper + Erizo Controller). De esta forma el sistema será escalable dependiendo de el número de usuarios que lo utilicen. La gestión de todos los Erizos arrancados la llevará Nuve ya que se comunica con los Erizo Controller a través de un bus de mensajes RabbitMQ mediante llamadas a procedimiento remoto (RPC). En cada uno de los apartados de esta sección se explican las interfaces de estas llamadas a procedimiento remoto.

Un ejemplo común de uso de Licode es el de un usuario conectándose a una sla de videoconferencia y publicando su video y audio en ella. Para ello cuando desea conectarse a la sala desde una aplicación web (Web App) la aplicación pedirá a su servidor web un token de autenticación. El servidor web (Server App) utilizando Nuve Client se lo pedirá a Nuve. Nuve lo generará y almacenará en la base de datos. Con este token de autenticación y utilizando Erizo Client el navegador web se conectará a Erizo Controller que validará el token contra Nuve a través del bus de mensajes (con una llamada a procedimiento remoto). Si el token es correcto Erizo Controller configurará Erizo y comenzará a publicar los datos usando la MCU.

Este sería un ejemplo resumido y simplificado de funcionalidad básica en Licode. No obstante las posibilidades son mucho más amplias y complejas. En las secciones siguientes se explica de manera detallada cada uno de los módulos de esta arquitectura y las comunicaciones entre ellos.

3.1 Nuve

Nuve es el módulo que se encarga de gestionar todos los componentes que intervienen en las sesiones multimedia de Licode. Estos componentes son:

- Salas
  Una sala de videoconferencia es un lugar abstracto donde los clientes publican sus flujos multimedia o donde acuden a suscribirse a los que otros clientes han publicado en esa sala. Lo que ocurre en una sala es privado y solo pueden participar en una sala los clientes con permiso explícito para ello.

- Usuarios
  Un usuario es un cliente que se conecta a una sala y publica o se suscribe a los flujos multimedia que otros usuarios han publicado.

- Servicios
  Un servicio es una aplicación que utiliza Nuve para la gestión de los recursos multimedia necesarios en su funcionamiento. Hay un tipo de servicio llamado *super servicio* que tiene una serie de privilegios respecto al resto de servicios como la capacidad de realizar acciones sobre los recursos de tipo servicio como crearlos o borrarlos.

- Tokens
  Un token en una ficha proporcionada por nuve a un cliente para que éste pueda acceder de manera autenticada a una sala de videoconferencia. Es de un solo uso y además únicamente válido durante un periodo de tiempo determinado.

Las aplicaciones web que utilicen Licode utilizarán por lo tanto Nuve para la gestión de estos recursos. Dispondrán de un servicio en el que podrán crear salas multimedia a las que darán acceso a los usuarios mediante un token de sesión.

Para ello utilizarán en sus servidores web un API de cliente de Nuve que accederá a los recursos mediante un API REST. Las peticiones http deberán estar autenticadas utilizando un protocolo diseñado en este departamento específicamente para Nuve. Se trata de MAuth. Además para proveer de persistencia al sistema, Nuve utilizará una base de datos.

A continuación se explica con detalle cada uno de estos componentes, el protocolo MAuth, el API de cliente de Nuve, el servidor REST, y la estructura de la base de datos. También la comunicación del servidor Nuve con el componente Erizo Controller.

3.1.1 MAuth

Como se ha explicado anteriormente MAuth es un protocolo de autenticación para los mensajes http que atacarán el API REST de Nuve. De cara a la autenticación http se sigue un esquema parecido otros protocolos como OAuth [OAuth] , es decir, una extensión de la cabecera estándar de autenticación de http. De una manera parecida a este protocolo, la cabecera se usa tanto para proporcionar la autenticación como para pasar parámetros en los casos que sea necesario.

Así un ejemplo de cabecera de autorización de http que los servicios deben utilizar de cara a poder realizar peticiones a Nuve es la siguiente:

```
Authorization: MAuth realm="http://xxxxx.es",
mauth_signature_method="HMAC_SHA1",
mauth_serviceid="ServiceName",
mauth_signature="jlsa731232=",
mauth_timestamp="1231321321",
mauth_cnonce="123123aadf",
mauth_version="3.1",
mauth_username="user",
mauth_role="participant"
```

A continuación se explican los parámetros uno a uno:

- *mauth_signature_method*: Indica el método usado para firmar la petición. Actualmente el único soportado es HMAC- SHA1. Para la generación de la firma se utiliza una clave simétrica que conocen tanto el servicio como Nuve y que se intercambia a través de otro canal.

- *mauth_serviceid*: Es un identificador único de cada servicio, utilizado por Nuve para conocer la clave con la que debe calcular la firma y, además, para asociar con él tanto las salas creadas como los usuarios añadidos.

- *mauth_signature*: La firma generada mediante el método señalado más arriba.

- *mauth_timestamp*: Por defecto, el número de segundos desde el uno de enero de 1970. Es obligatorio que sea un entero positivo y debe ser igual o mayor que el de anteriores peticiones.

- *mauth_cnonce*: Un número entero positivo aleatorio que hay que asegurar que cambie entre peticiones con un mismo timestamp. Así se protege frente ataques de *replay*.

- *mauth_version*: Número de versión actual de la autorización. Actualmente se utiliza 3.1.

Hasta aquí los parámetros obligatorios en cada petición, los dos siguientes son especiales ya que únicamente se utilizan en aquellos casos en los que se pretende solicitar el acceso a una sala para un usuario.

- *mauth_username*: Nombre del usuario que se desea que entre en la sala de conferencia.

- *mauth_role*: Rol que ejercerá dicho usuario dentro de la conferencia. Se utilizarán para diferencian la capacidad de control que tienen sobre el desarrollo de la conferencia.

La cadena con la que se obtiene la firma variará dependiendo de los parámetros que se incluyen en la cabecera. Así esta cadena será:

```
(mauth_serviceid, mauth_timestamp, mauth_cnonce,
[mauth_username, mauth_role])
```

Por lo tanto cualquier petición REST que se realice al servidor de Nuve deberá ir autenticado con los parámetros explicados. Como Licode provee a los servicios web un API de cliente de Nuve la implementación de este protocolo ya está incluida en el API y el desarrollador que lo utilice únicamente deberá preocuparse de configurar antes de realizar cualquier petición las credenciales del servicio que esté utilizando como se explica en el siguiente apartado.

3.1.2 API de cliente

En la fecha en que se escribe este documento, el API de Nuve está escrito en los lenguajes de programación Node.js, Python y Ruby. Por lo tanto puede utilizarse en

cualquier servidor web que utilice estos lenguajes en su lógica de servicio. Todos ellos tienen la misma interfaz de atributos y métodos, que se pasa a explicar a continuación.

Inicialización

Como se ha explicado anteriormente antes de empezar a utilizar cualquier recurso de Nuve, es decir, antes de realizar cualquier petición mediante los métodos proporcionados en el API, es necesario inicializar con una serie de parámetros que se utilizarán para la autenticación de los mensajes.

Inicialización
init (serviceId, serviceKey, nuve_host)

Tabla 1. Interfaz Nuve API. Inicialización

Los parámetros que utiliza esta función son:

- serviceID: identificador del servicio para el que se utilizarán los recursos de Nuve para esta instancia del API.
- serviceKey: clave de acceso a Nuve para este servicio. Se utilizará junto con el campo anterior para la autenticación en las cabeceras del protocolo MAuth según se ha explicado anteriormente.
- nuve-host: dirección donde se encuentra corriendo la instancia de Nuve a la que se tiene que acceder para utilizar el API REST.

Salas

En la Tabla 2 pueden observarse las propiedades y las funciones o métodos públicos que podrán utilizarse sobre un recurso del tipo sala en el API de Nuve y que se explican a continuación.

Propiedades	Funciones
Room._id	createRoom (name)
Room.name	getRooms ()
	getRoom (roomId)
	deleteRoom (roomId)

Tabla 2. Interfaz Nuve API. Salas

- _id: es el identificador único de sala en la base de datos.
- name: nombre de la sala
- createRoom: crea una sala con el nombre especificado en el servicio. Devuelve un objeto sala con el identificador de la sala en la base de datos.
- getRooms: devuelve una lista de las salas disponibles en el servicio.
- getRoom: devuelve una sala especificando su identificador en la base de datos.
- deleteRoom: borra una sala especificando su identificador único en la base de datos.

Usuarios

En la Tabla 3 pueden observarse las propiedades y las funciones o métodos públicos que podrán utilizarse sobre un recurso del tipo usuario en el API de Nuve y que se explican a continuación.

Propiedades	Funciones
User.name	getUsers (roomId)
User.role	

Tabla 3. Interfaz Nuve API. Usuarios

- name: nombre del usuario en la sala a la que se ha conectado.

- role: rol con el que el usuario se ha conectado a una sala.

- getUsers: devuelve una lista de los usuarios que están conectados a una sala especificada mediante su identificador en la base de datos.

Servicios

En la Tabla 4 pueden observarse las propiedades y las funciones o métodos públicos que podrán utilizarse sobre un recurso del tipo servicio en el API de Nuve y que se explican a continuación. Cabe destacar que solamente un servicio del tipo *super servicio* puede realizar acciones sobre otro servicio.

Propiedades	Funciones
Service_id	createService (name)
Service.name	getServices ()
Service.key	getService (servideId)
Service.rooms	deleteService (serviceId)

Tabla 4. Interfaz Nuve API. Servicios

- _id: identificador único del servicio en la base de datos.

- name: nombre del servicio.

- key: clave del servicio.

- rooms: lista de las salas que se han creado en este servicio.

- createService: crea un servicio sin salas especificando su nombre. Devuelve un objeto servicio con su identificador en la base de datos.

- getServices: devuelve una lista con los servicios disponibles.

- getService: devuelve un objeto servicio especificando su identificador en la base de datos.

- deleteService: borra un servicio especificando su identificador de servicio en la base de datos.

Tokens

En la Tabla 5 pueden observarse las propiedades y las funciones o métodos públicos que podrán utilizarse sobre un recurso del tipo token en el API de Nuve y que se explican a continuación.

Propiedades	Funciones
	createToken (roomId, name, role)

Tabla 5. Interfaz Nuve API. Tokens

- createToken: crea un token para un usuario especificando la sala a la que va a conectarse mediante su identificador en la base de datos, el nombre que tendrá el usuario en la sala y el rol que desempeñará en la sala.

3.1.3 Servidor

Se trata del servidor REST que recoge las peticiones enviadas utilizando el API de Nuve desde los servidores de las aplicaciones web que usan Licode. Su lógica está escrita utilizando el lenguaje de programación Node.js. Antes de explicar la interfaz REST con la descripción de las acciones posibles sobre los recursos de Nuve es necesario hablar de un módulo fundamental en la gestión de la escalabilidad en Licode. Este módulo se llama Cloud Handler debido a su futura relación con el despliegue en el Cloud que se explicará en otro capítulo de este documento.

Cloud Handler

Como se ha explicado en la introducción de la arquitectura del sistema cuando el número de usuarios o salas de videoconferencia es muy elevado puede ser necesario

arrancar más de una MCU o módulo Erizo. Cuando un nuevo usuario se conecte a una sesión y quiera publicar sus datos multimedia en una sala o suscribirse a otros existentes en ella pedirá a Nuve (a través del servidor de la aplicación) un token de conexión. En ese momento Nuve necesitará saber la ubicación física de la MCU que está gestionando esa sala, es decir, la dirección IP de la máquina donde la MCU está corriendo. Para ello se utiliza el módulo Cloud Handler y la IP servirá para que el cliente sepa dónde se encuentra la sala, es decir, con qué máquina debe establecer una conexión de websockets como se explicará en la sección dedicada a Erizo.

Así Cloud Handler debe llevar un control de los Erizos (que estarán gestionados por un Erizo Controller cada uno) disponibles y de las salas que gestiona cada una de ellos. Para ello cuando se arranque un nuevo Erizo en el sistema éste se lo comunicará a Cloud Handler a través de una llamada a procedimiento remoto.

Además cada Erizo Controller enviará a Cloud Handler un mensaje periódico de Keep Alive para confirmar que ese Erizo sigue corriendo y disponible para gestionar salas de videoconferencia.

Por último Cloud Handler dispone de una cola ordenada con los Erizos arrancados en el sistema según el número de salas que gestiona cada uno. Así cuando se cree una nueva sala sabrá a qué Erizo debe asignarla. Cada Erizo Controller enviará a Cloud Handler un mensaje con información acerca de su estado. Este estado incluye parámetros como el número de salas o de usuarios que están activas en la sesión. Los mensajes de este tipo se enviarán cuando haya un cambio en el estado y servirán para que Cloud Handler actualice esta cola según la prioridad de asignación de salas. La información de este estado se explicará como más detalle en el la sección dedicada a Erizo Controller.

Interfaz REST

En la Tabla 6 puede observarse una relación de las acciones REST que pueden realizarse sobre los recursos de Nuve y su correspondencia con las funciones explicadas anteriormente para el API de cliente.

Recurso	Método	Función API Cliente
Sala	POST	createRoom
	GET	getRooms
	GET: room	getRoom
	DELETE :room	deleteRoom
Usuario	GET	getUsers
	GET :user	No soportado
	DELETE :user	No soportado
Servicio	POST	createService
	GET	getServices
	GET :service	getService
	DELETE :service	deleteService
Token	POST	createToken

Tabla 6. API REST Nuve

Al recibir cualquier petición en primer lugar el servidor comprueba que el mensaje http esté correctamente autenticado según MAuth. En caso de que no sea correcto contestará indicando que es necesaria una autenticación. En caso correcto pasará a procesar la petición.

La mayoría de estos procesos de petición únicamente realizarán la petición indicada actualizando la base de datos (cuya estructura se explicará en el siguiente apartado) u obteniendo datos sobre ella. No obstante hay dos peticiones que deben procesarse haciendo algunas tareas adicionales. Estas peticiones son la de consultar el número de usuarios en una sala (GET al recurso users) y la de crear un token (POST al recurso token).

Cuando se realiza una petición de consulta de usuarios en una sala Nuve debe consultar al Erizo Controller que está gestionando esa sala cuántos usuarios hay conectados en ese momento a esa sala. Puesto que, como se ha explicado, puede haber varios Erizo Controllers en primer lugar debe consultarse el registro de los Erizo Controllers en Cloud Handler para, una vez conocido el que está gestionando esa sala, realizar una llamada a procedimiento remoto al Erizo Controller consultando el número de usuarios.

Cuando se realiza una petición de creación de un token para la conexión de un usuario a una sala Nuve genera un string codificado en base 64 que tiene la siguiente estructura:

{tokenId: id, host: erizoController host, signature: signature of the token};

- tokenId: es el identificador del token en la base de datos de Nuve.

- host: la dirección IP de la máquina en la que se encuentra arrancado el Erizo que gestiona la sala y al que deberá conectarse el cliente. Para saberla Nuve preguntará a Cloud Handler, que lleva un registro de los Erizos arrancados y las salas que gestionan. En caso de ser una sala nueva decidirá en qué Erizo debe crearse.

- signature: es la concatenación de los dos campos anteriores cifrada utilizando el algoritmo de cifrado hash SHA1 con la clave del servicio en Nuve.

El string con estos tres campos se devuelve en la respuesta REST y el token, con el resto de parámetros que se explicarán en la subsección dedicada a la base de datos se almacena.

Interfaz de llamadas a procedimiento remoto

Como se ha explicado en la introducción tanto Nuve como Erizo Controller disponen de una interfaz de llamadas a procedimiento remoto RPC para comunicarse entre ellos a través de un bus de mensajes sobre el que se ha implementado esta funcionalidad.

Los métodos públicos de los que dispone Nuve están divididos en los que están directamente relacionados con Nuve y los que guardan relación con el módulo Cloud Handler.

- deleteToken: cuando un cliente accede a una sala utilizando un token Erizo Controller debe validarlo en Nuve. Para ello hace una llamada a esta función que comprueba si el token es válido y lo elimina de la base de datos para que no pueda volver a utilizarse. El token será válido por lo tanto si no ha sido utilizado anteriormente y si no ha caducado. En cuanto a la comprobación

del cifrado con la clave de Nuve, se realiza en el propio Erizo Controller como se explicará más adelante.

- addNewErizoController: cuando un nuevo Erizo Controller es arrancado ejecuta esta función para ser añadido en el registro de Cloud Handler.

- keepAlive: mensaje de keep alive para indicar que el Erizo Controller sigue arrancando y ejecutándose sin ningún problema. Si tras un periodo de tiempo no se reciben mensajes de este tipo de un Erizo Controller se eliminará del registro en Cloud Handler.

- setInfo: función para actualizar el estado de un Erizo Controller con parámetos como el número de usuarios o salas activas.

- killMe: en caso de que se reciba un mensaje de keep alive de un Erizo Controller que no exista en el registro (puede ser debido a que se eliminó porque no respondía y al paso de un tiempo ha vuelto a responder) deberá procederse a parar su proceso en la máquina en la que estuviera corriendo.

3.1.4 La Base de Datos

Para la persistencia de los datos en Nuve se utiliza una base de datos MongoDB. Se trata de una base de datos NoSQL orientada a documentos JSON que está escrita en C++. Utilizando una librería de Node.js puede accederse a la base de datos que guarda tablas para los recursos sala, servicio y token. Así las entradas de la base de datos de cada uno de los tipos tienen la siguiente estructura:

- room

  {name: string, _id: ObjectId}

- service

  {name: string, key: string, rooms: Array[room], _id: ObjectId}

- token

  {host: string, userName: string'', room: roomId, role: string, service: serviceId, creationDate: Date(), _id: ObjectId}

3.2 Erizo

Erizo es el módulo que se encarga de la gestión multimedia de las comunicaciones en Licode. Es la MCU (Multipoint Control Unit) del sistema. Una MCU [Willebeek-LeMair et al. 1994] es un componente software utilizado en sistemas multimedia y de comunicaciones en tiempo real para actuar como módulo central entre los clientes o usuarios. De esta manera, como se observa en la Figura 2, puede convertirse una topología en malla en la que todos los usuarios se comunican con todos, en una topología en estrella donde la MCU actúa de nodo central e interconecta a todos los clientes. Gracias a esta topología la MCU realiza las siguientes funciones con sus consecuentes ventajas para el sistema:

1. Redirección

   En una topología en malla en la que todos los usuarios se comunican con el resto, cada usuario debe enviar sus flujos multimedia a todos y cada uno de los clientes y recibir un flujo de cada uno de ellos. En cambio en una topología en estrella la MCU actuando como nodo central recibe los flujos de cada cliente y los distribuye al resto. Esto se traduce en un ahorro considerable de ancho de banda ya que cada cliente tiene que enviar una única vez su flujo multimedia a la MCU.

2. Composición

   La situación descrita anteriormente mejora aún más si la MCU no envía a cada cliente un flujo de datos por cada uno del resto de clientes sino que realiza una composición resultando en el envío de un único flujo resultado de la suma del resto.

3. Transcodificación

   En determinadas ocasiones los clientes que participan en una sesión multimedia son de naturalezas muy distintas. Por ello resulta interesante que el nodo central MCU pueda realizar una transformación de los flujos multimedia para adaptarlos a las necesidades de cada cliente. Estas necesidades pueden ser de varios tipos. En ocasiones el formato de los datos con los que es compatible cada cliente puede variar (códecs de audio y vídeo por ejemplo). También puede ser necesario adaptar las calidades a las necesidades de cada cliente. Esto puede ser debido a que operen en redes con diferentes capacidades de ancho de banda (redes 3G, WiFi, wired) o simplemente que por las características del terminal

las calidades necesarias sean diferentes (por ejemplo por el tamaño de la pantalla).

4. Grabación

   En las aplicaciones multimedia es muy frecuente que sea necesario realizar una grabación de las sesiones para almacenarlas y poder reproducirlas posteriormente. Para esto es muy útil la presencia de una MCU ya que al recibir los flujos multimedia de todos los participantes podrá realizar la grabación de la sesión completa (o de una parte de ella) y almacenarla para su posterior uso.

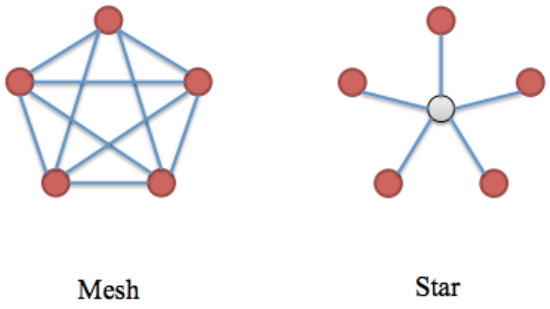

Figura 2. Topologías de red

Erizo por lo tanto es la MCU [P. Rodríguez et al. 2012] del sistema y desde el punto de vista de WebRTC se comporta como un cliente más. Por lo tanto Erizo debe implementar los protocolos necesarios para establecer una comunicación WebRTC. Como se explicó a principio del documento estos protocolos son ICE y SRTP (con la extensión DTLS-SRTP para la gestión de claves).

Erizo está escrito en los lenguajes de programación C y C++ y puesto que el módulo superior (Erizo Controller) que se encarga de controlar a gestión de la MCU está desarrollado utilizando Node.js, es necesario un wrapper que recubra las clases que serán accesibles desde fuera (y que son las que se explicarán en este documento).

3.2.1 Wrapper Node

Se trata de un módulo de recubrimiento del núcleo de la MCU Erizo y su nombre es Erizo API. Gracias a él las clases de Erizo necesarias serán accesibles desde el módulo

de control Erizo Controller. Está desarrollado utilizando las librerías V8 de C++ que a partir de clases C++ crean interfaces accesibles desde JavaScript.

Las clases que recubre son las siguientes:

- WebRtcConnection: permite establecer conexiones con los clientes WebRTC. Por lo tanto se encarga de procesar los SDPs de los clientes web y de generar los SDPs locales de la MCU. Además una vez establecida la conexión recibe y envía los datos multimedia.

- ExternalInput: permite establecer conexiones con otro tipo de fuentes multimedia como cámaras IP, flujos RTMP, etc.

- OneToManyProcessor: actúa como multiplexador multimedia recibiendo un fujo de datos y redirigiéndolo a otros clientes. Su entrada (publisher) por lo tanto es un objeto WebRtcConnection o ExternalInput que publica un flujo de audio y/o vídeo y sus salidas (subscribers) son uno o más objetos WebRtcConnection que se suscriben a los flujos multimedia.

- OneToManyTranscoder: su función es la misma que la del OneToManyProcessor con la diferencia de que es capaz de transcodificar los flujos de audio y vídeo según ciertos parámetros como códec, ancho de banda, frame rate, etc.

3.3 Erizo Controller

Erizo Controller es el módulo que se encarga de controlar Erizo de acuerdo a las necesidades de los clientes. Maneja tres conceptos fundamentales:

- Stream: identifica un flujo local o remoto compuesto de vídeo, audio, datos o varios de ellos.

- Sala: representa una sala de Nuve. Es decir, un espacio virtual donde los clientes pueden publicar sus streams o suscribirse a otros streams.

- Token: representa un token de Nuve. Utilizado por un cliente web para autenticarse en una sala y poder acceder a los streams publicados en ella así como publicar los propios.

Por lo tanto Erizo Controller se encarga de gestionar tanto la conexión de los clientes web a las salas de videoconferencia autenticándolos mediante un token contra Nuve como el manejo de los streams de los clientes en las salas. Un cliente podrá publicar sus streams en una sala o suscribirse a los que previamente otros clientes han publicado.

Desde un punto de vista de la arquitectura Erizo Controller está colocado por encima del módulo Erizo y dispone de un API de cliente (Erizo Client) que los clientes web utilizarán para comunicarse con él a través de una conexión de websockets.

A continuación se explica con detalle cada uno de estos componentes, el API de cliente de Erizo Controller, el servidor y también la comunicación de éste con el componente Nuve.

3.3.1 API de Cliente

Se trata de un API escrito en el lenguaje de programación Javascript que las aplicaciones web que utilicen Licode deberán incluir en sus recursos. Con él se pueden controlar tres tipos de objetos. Objetos tipo Stream para el control de los fujos multimedia que se publiquen o a los que se suscriba el cliente. Objetos del tipo Room que gestionan la conexión a una sala multimedia y objetos de tipo Event para la gestión de eventos sobre Streams o Rooms.

Para la conexión del navegador del cliente con Erizo Controller se utiliza una conexión de websockets mediante la que se intercambiarán diferentes tipos de mensajes. Como se explicó al hablar de Nuve cuando el cliente quiera conectarse a una

sala en el token de conexión estará incluida la dirección IP de la máquina en la que está corriendo el Erizo Controller correspondiente. Así que será con esa máquina con la que se establecerá la conexión de websockets.

A continuación se explica con detalle cada uno de estos tipos de objetos así como la forma de utilizarlos mediante las propiedades y funciones de cada uno de ellos.

Stream

Un objeto de tipo Stream realiza un recubrimiento ampliado del API MediaStream de WebRTC que además incluye algunas características propias de Licode y Erizo Controller. Representa por lo tanto un flujo multimedia de audio, vídeo, datos o de una combinación de cualquiera de ellos. Este flujo multimedia puede ser local o remoto. En ambos casos si contiene audio o vídeo podrá reproducirse o mostrarse su contenido en un elemento del DOM html de la página web del cliente. Si es local este contenido se obtendrá directamente del hardware del cliente (micrófono y webcam) utilizando el API GetUserMedia de WebRTC.

Puesto que en cada navegador la implementación de WebRTC es diferente Erizo Controller posee un adaptador o wrapper que permite que usando siempre la misma interfaz del API se utilicen unas u otras implementaciones en función del navegador. Además la detección del navegador en la que está corriendo el cliente se realiza de forma automática.

En caso de ser de datos el cliente podrá enviar datos a utilizando un Stream local o leer los datos que hay en los Streams remotos. En la fecha en que se escribe este documento la implementación de datos en WebRTC no está finalizada por lo que en Licode la transmisión de datos se realiza a través de la propia conexión de websockets entre el navegador y Erizo Controller.

En la Tabla 7 puede observarse tanto la forma de inicializar un objeto de este tipo tanto las propiedades y funciones de que dispone y cuya función se explica a continuación.

Inicialización
Erizo.Stream ({audio: boolean, video: boolean, data: boolean, attributes: JSON})

Propiedades	Funciones
showing	hasAudio ()
room	hasVideo ()
local	hasData ()
	init ()
	getID ()
	close ()
	show (elementID, options)
	hide ()
	sendData (msg)
	getAttributes ()
	getVideoFrame ()
	getVideoFrameURL ()

Tabla 7. Interfaz Erizo Controller API. Stream

Al crear un objeto del tipo Stream debe especificarse si contendrá audio, datos y vídeo. Puede contener cualquier combinación de estos tres. Además puede incluirse una lista de atributos en forma de objeto JSON que se utilizará para lo que cada desarrollador necesite en sus aplicaciones.

- showing: propiedad que indica si el vídeo del Stream está mostrándose en algún elemento de la web.
- room: en caso de que el Stream esté siendo publicado en una sala, es el identificador de esa sala.
- local: booleano que indica si el Stream es local.
- hasAudio: devuelve un booleano indicando si el Stream tiene audio.
- hasVideo: devuelve un booleano indicando si el Stream tiene vídeo.
- hasData: devuelve un booleano indicando si el Stream tiene datos.

- init: accede a la información multimedia de cliente, cámara y/o micrófono, mediante el API GetUserMedia. En este momento se pedirá al usuario permiso para acceder a esa información. Solamente puede utilizarse sobre Streams locales. Si el Stream solamente contiene datos esta función no realiza ninguna acción.

- getID: obtiene un identificador único del Stream.

- close: detiene el acceso a la información multimedia del cliente. En caso de que el Stream estuviese siendo publicado deja de hacerse. En caso de que el Stream fuera de vídeo y estuviera mostrándose en un elemento del DOM deja de hacerse. Al igual que init solamente puede utilizarse sobre un Stream local.

- show: muestra el vídeo del Stream en el elemento del DOM especificado. Pueden especificarse algunas opciones como el hecho de mostrar o no el controlador de volumen en el tag de vídeo.

- hide: deja de mostrar un Stream previamente mostrado con show.

- sendData: envía un objeto JSON a través de un Stream de datos.

- getAttributes: obtiene la lista de atributos con la que se inicializó el Stream. Se trata de un objeto JSON.

- getVideoFrame: obtiene una captura de un Stream de vídeo en forma de mapa de bits.

- getVideoFrameURL: obtiene una captura de un Stream de vídeo y devuelve la URL donde se encuentra la imagen.

Room

Un objeto de tipo Room representa un sala multimedia de Licode. Gestiona la conexión del cliente web con Erizo Controller mediante websockets y la forma en que publican o se suscriben a Streams los usuarios. Para esta publicación o suscripción a Streams deben realizarse conexiones WebRTC entre el navegador y Erizo por lo que el

objeto Room utilizará el API PeerConnection del estándar. Al igual que en el caso del API GetUserMedia se utiliza un recubrimiento que permite la compatibilidad con distintos navegadores.

El intercambio de los mensajes de señalización se realizan a través de la conexión de websockets por lo que el objeto Room se encargará tanto del envío de los mensajes como de escuchar para la recepción de las respuestas. La conexión también se utilizará para el envío de objetos JSON a través de Streams de datos.

En la Tabla 8 puede observarse tanto la forma de inicializar un objeto de este tipo tanto las propiedades y funciones de que dispone y cuya función se explica a continuación.

Inicialización
Erizo.Room ({token: string})

Propiedades	**Funciones**
roomID	connect ()
state	publish (stream)
localStreams	subscribe (stream)
remoteStreams	unpublish (stream)
	unsubscribe (stream)
	disconnect ()
	getStreamsByAttribute (name, value)

Tabla 8. Interfaz Erizo Controller API. Room

Al crear un objeto de tipo Room deberá especificarse el token de conexión a la sala que se habrá obtenido a través del servidor de la aplicación web (y éste de Nuve utilizando el API de cliente). En este token está la información sobre el host donde se encuentra corriendo el Erizo Controller que gestiona la sala.

- roomID: identificador único de la sala.

- state: estado en el que se encuentra la sala. Puede ser DISCONNECTED (desconectado), CONECTING (conectando) o CONNECTED (conectado).

- localStreams: lista con los objetos de tipo Stream locales publicados en la sala.

- remoteStreams: lista con los objetos de tipo Stream remotos publicados en la sala.

- connect: realiza la conexión con la sala. Esto significa establecer la conexión de websockets y comenzar a escuchar mensajes.

- publish: publica un Stream en la sala. En el caso de Streams de vídeo y/o audio esto implica crear un objeto PeerConnection, realizar el intercambio de mensajes de señalización y comenzar a enviar el flujo multimedia a Erizo a través de la conexión SRTP del estándar.

- subscribe: se suscribe a un Stream previamente publicado en la sala. En el caso de Streams de vídeo y/o audio esto implica crear un objeto PeerConnection, realizar el intercambio de mensajes de señalización y comenzar a reciir el flujo multimedia a Erizo a través de la conexión SRTP del estándar.

- unpublish: dejar de enviar los datos a la sala. En caso de audio y/o vídeo implica cerrar el PeerConnection.

- unsubscribe: dejar de recibir los datos de este Stream de la sala. En caso de audio y/o vídeo implica cerrar el PeerConnection.

- disconnect: desconectarse de la sala. Esto implica cerrar la conexión de websockets, dejar de publicar los Streams que se estuvieran publicando y dejar de recibir los datos de los Streams a los que se estuviera suscrito.

- getStreamsByAttribute: obtiene una lista de Streams remotos con la coincidencia indicada en los parámetros en función de los atributos de los Streams.

Event

Se trata de un objeto especial que representa eventos sobre otros objetos. La forma de inicializarlos no es relevante en esta descripción del API ya que se hace

internamente en los objetos de tipo Stream y Room. Por lo tanto lo relevante es la forma en que se añade un listener a un objeto y los tipos de eventos que pueden recibirse en ese objeto.

En la Tabla 9 puede observarse tanto la forma agregar un listener a un objeto de tipo Room o Stream.

Funciones
Room.addEventListener (type, function (evt) { ... });
Stream.addEventListener (type, function (evt) { ... });

Tabla 9. Interfaz Erizo Controller API. Event

Los tipos de eventos que pueden lanzarse en un objeto de tipo Room son los siguientes:

- room-connected: indica que la conexión con la sala se ha realizado de forma correcta. El evento incluye un lista de los streams remotos que actualmente están publicados en la sala.

- room-disconnected: indica que la desconexión con la sala se ha realizado de forma correcta.

- stream-added: indica que un nuevo Stream ha sido publicado en la sala. El evento incluye el objeto Stream.

- stream-removed: indica que un Stream se ha dejado de publicar en la sala. El evento incluye el objeto Stream.

Los tipos de eventos que pueden lanzarse en un objeto de tipo Stream son los siguientes:

- access-accepted: indica que el usuario ha aceptado el acceso a sus recursos multimedia (micrófono y/o cámara).

- acces-denied: indica que el usuario ha denegado el acceso a sus recursos multimedia.

- stream-data: indica que en este Stream de datos se han publicado nuevos datos. El evento incluye un campo con el objeto JSON que contiene los datos publicados.

En la Figura 3 puede observarse un ejemplo de intercambio de eventos entre dos clientes y Erizo Controller.

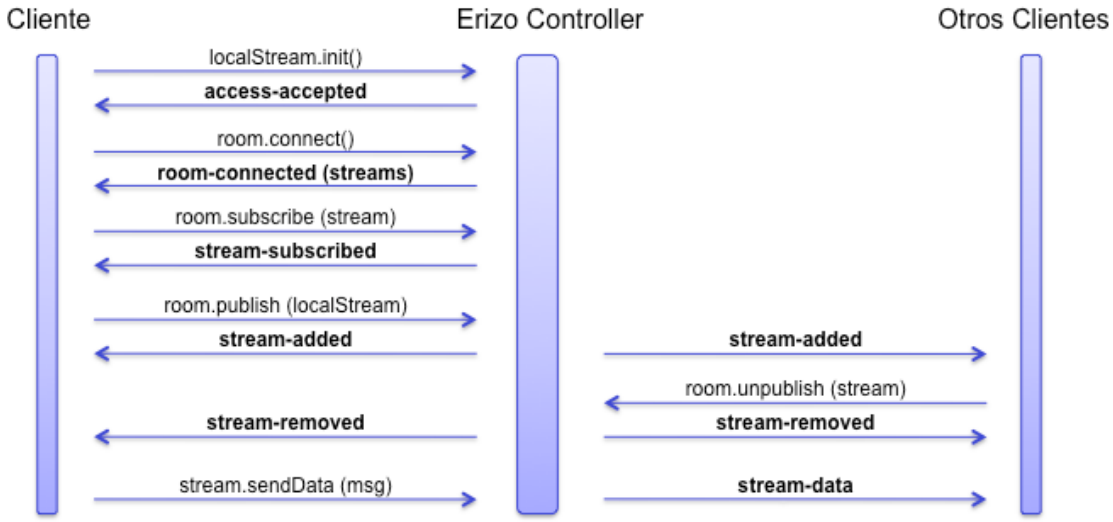

Figura 3. Ejemplo de intercambio de eventos

A continuación se explican las acciones que van realizándose por el cliente y los eventos que estas acciones lanzan, tanto para él mismo como para otros clientes conectados a la misma sala.

1. Sobre un Stream local se llama a la función *init* de tal forma que el navegador pedirá acceso a su cámara y /o micrófono. En caso de que el usuario acepte se recibirá sobre ese Stream un evento de tipo *access-accepted.*

2. Sobre un objeto de tipo Room se llama a la función *connect* con lo que se establece la conexión con Erizo Controller. En caso de que esta acción tenga éxito se recibirá sobre ese objeto un evento de tipo *room-connected* incluyendo la lista de Streams remotos que hay publicados en la sala.

3. Se realiza la acción de suscribirse a uno de estos Streams remotos que contiene por ejemplo auido y vídeo. Con ello se crea un PeerConnection y se realiza el intercambio de mensajes con Erizo Controller. Si la conexión se realiza correctamente se recibe un evento de tipo *stream-subscribed* y se comienzan a recibir los datos de tal forma que el cliente puede mostrar el vídeo recibido en un elemento de la página.

4. Se desea publicar un Stream local y se realizan las mismas acciones que en el caso de la suscripción. A todos los clientes conectados a la sala les llegará un evento del tipo *stream-added* y podrán suscribirse si lo desean.

5. En caso de que otro cliente deje de publicar un Stream a todos los conectados a la sala les llegará un evento de tipo *stream-removed.*

6. Por último se envían datos sobre un Stream de datos con lo que a todos los clientes que estén suscritos a ese Stream les llegará un evento del tipo *stream-data* conteniendo los datos que se han enviado.

Erizo Server Client

Como última consideración sobre el API de cliente de Erizo Controller cabe destacar que también está disponible para utilizarse como una librería de Node.js. De esta forma puede simularse un cliente que ejecutará en cualquier servidor Node y que puede conectarse a una sala y realizar las mismas acciones que un cliente web utilizando exactamente el mismo API.

Esto resulta útil por ejemplo para publicar flujos multimedia provenientes de otros medios que no sean la cámara de un cliente como cámaras IP o para publicar datos en una sala desde un servidor sin necesidad de arrancar un navegador web.

3.3.2 Servidor

Se trata del servidor que escucha los mensajes enviados por los clientes web que utilizan el API de cliente de Erizo Controller, los procesa y se comunica con Erizo para realizar las acciones correspondientes en la MCU. Está escrito en el lenguaje de programación Node.js por lo que para realizar esta comunicación con Erizo Controller es necesario el wrapper o addon de Node para C++ que se explicó anteriormente.

Además Erizo Controller se comunicará con Nuve a través de un bus de mensajes para realizar llamadas a procedimiento remoto.

En primer lugar y cuando se arranca un Erizo Controller éste se comunica con Nuve para notificarle que está preparado para comenzar a gestionar salas multimedia de Licode. Lo hace ejecutando la llamada a procedimiento remoto *addNewErizoController* que se explicó en la interfaz RPC de Nuve. En esta llamada le indica a Nuve su dirección IP y cuando Nuve contesta afirmativamente a la petición Erizo Controller comienza a enviar los mensajes de keepalive y a escuchar en el puerto de websockets mensajes de los clientes.

Es importante destacar que Erizo Controller está preparado para ser arrancado en máquinas de proveedores de cloud públicos. Actualmente es compatible con el proveedor de cloud Amazon EC2 [Amazon]. Al arrancar una máquina de esta forma Erizo Controller le indicará a Nuve esta circunstancia para que Nuve realice una llamada al API del proveedor consutando la IP pública de la máquina en la que está corriendo el Erizo Controller.

Erizo Controller posee una lista con el registro de las salas multimedia que está gestionando. Este registro inluye un identificador de la sala, una lista de los Streams publicados en la sala, una lista de los clientes conectados a la sala (a través del identificador de su conexión de websockets) una lista con la correspondencia de suscripciones a Streams en la sala y un objeto WebRtcController.

Un objeto WebRtcController representa una sala multimedia desde el punto de vista de Erizo. Por lo tanto se utiliza para realizar las acciones necesarias para gestionar la MCU utilizando la interfaz del wrapper Node de Erizo. Así utiliza los objetos WebRtcCOnnection, ExternalInput, OneToManyProcessor y OneToManyTranscoder que se explicaron anteriormente y las acciones sobre ellos.

Los mensajes enviados por los clientes que Erizo Controller puede procesar escuchando en el puerto de websockets son los siguientes:

- "token": mensaje recibido cuando un cliente se conecta a una sala enviando un token de autenticación. Erizo Controller ejecuta una llamada a procedimiento remoto contra Nuve para verificar que el token es efectivamente válido. En caso afirmativo comprueba si la sala es nueva o ya existía. Si es nueva la registra en su lista de salas creando el objeto WebRtcController correspondiente. Por último devuelve al cliente una confirmación incluyendo la lista de Streams publicados en la sala (si la sala es nueva esta lista estará vacía). En caso de que el token no sea correcto devuelve un mensaje de error al cliente web y cierra la conexión de websockets.

- "sendDataStream": mensaje recibido cuando un cliente envía datos a través de un Stream de datos. Erizo Controller reenvía los datos a todos los clientes que estén conectados a la sala y suscritos a ese Stream. Lo hace a través del websocket correspondiente a ese cliente.

- "publish": mensaje recibido cuando un cliente desea publicar un Stream en una sala. En caso de que el Stream sea únicamente de datos Erizo Controller actualiza el registro de Streams y devuelve una confirmación al cliente. Si además tiene algún componente multimedia ejecuta el método *addPublisher* del objeto WebRtcController de la sala correspondiente. Esto servirá para que se añada un nuevo OneToManyProcessor en la MCU con fuente el Stream que se está publicando. Para ello será necesario el intercambio de mensajes SDP de negociación entre Erizo y el cliente web. Una vez acabado de forma satisfactoria el proceso devolverá un mensaje de confirmación al cliente. En ambos casos (tanto habiendo componentes multimedia como no) si el proceso concluye de forma satisfactoria Erizo Controller enviará un mensaje a todos los clientes conectados a la sala notificando que hay un nuevo Stream publicado en la misma.

- "subscribe": mensaje recibido cuando un cliente se desea suscribir a un Stream de la sala. Al igual que en el caso de publicar si el Stream contiene solo datos Erizo Controller simplemente actualizará el registro de Streams. En caso de que contenga componentes multimedia ejecutará el método *addSubsriber* del objeto WebRtcController de la sala. Esto implica añadir un nuevo destino en el OneToManyProcessor correspondiente al Stream publicado al que se quiere suscribir el cliente. Si la negociación concluye de forma satisfactoria Erizo Controller devolverá un mensaje de confirmación al cliente.

- "unpublish": mensaje recibido cuando un cliente quiere dejar de publicar un Stream en una sala. Erizo Controller actualiza su registro de Streams y en caso de haber componentes multimedia ejecuta el método *removePublisher* del objeto WebRtcController. Esto implica cerrar las conexiones SRTP de todos los clientes que estaban suscritos a ese Stream y eliminar el objeto OneToManyProcessor de Erizo. Si todo va bien Erizo Controller devuelve un mensaje de confirmación al cliente y envía un mensaje a todos los clientes conectados a la sala indicando que un Stream ha sido eliminado de la sala para que realicen las acciones que deseen.

- "unsubscribe": mensaje recibido cuando un cliente quiere dejar de suscribirse a un Stream en una sala. Erizo Controller actualiza su registro de Streams y en caso de haber componentes multimedia ejecuta el método *removeSubscriber* del objeto WebRtcController. Esto implica cerrar la conexión SRTP de este cliente. Si todo va bien Erizo Controller devuelve un mensaje de confirmación al cliente.

- "disconnect": mensaje recibido cuando un cliente desea desconectarse de la sala o cuando se produce un fallo en la conexión de websockets de un cliente. Erizo Controller elimina los Streams que ese cliente estuviera publicando y a los que estuviera suscrito cerrando sus conexiones SRTP. Además cierra las de el resto de clientes de la sala que estuvieran suscritos a cualquier Stream publicado por este cliente y notifica a todos los clientes de la sala que un Stream ha sido eliminado. Por último actualiza el registro de Streams y en caso de que la sala quedase vacía ya que este fuera el único cliente conectado la elimina del registro.

Erizo Controller realiza una monitorización del estado de carga del Erizo que controla. Esta monitorización tiene en cuenta el número de salas que está gestionando. El desarrollador que configura el sistema puede configurar unos valores de referencia de carga. Estos valores son dos, uno indica un número de salas nivel de warning y el otro un número de salas límite. En función del número de salas que estén activas en el Erizo que controla el estado de carga será *available, warning* o *notAvailable.*

- Un estado *available* indica que el número de salas está por debajo del niver de warning y que por lo tanto el Erizo puede gestionar más salas si es necesario.

- Un estado *warning* indica que el número de salas está entre en nivel de warning y el nivel límite y por lo tanto el Erizo no debería gestionar más salas.

- Un estado *notAvailable* indica que el número de salas ha sobrepasado el nivel límite y por lo tanto el Erizo no va a gestionar ninguna sala más en ningún caso.

Este estado se enviará a Nuve (concretamente a Cloud Handler) cada vez que varíe a través de una llamada a procedimiento remoto. De tal modo que Cloud Handler tomará las acciones que sean oportunas en función del estado de los Erizo Controller arrancados. Cada vez que se realice una acción que implique crear una sala, o eliminarla Erizo Controller recalculará su estado y en caso de que varíe enviará la actualización a Nuve.

Interfaz de llamadas a procedimiento remoto

Al igual que el caso de Nuve Erizo Controller dispone de una interfaz de llamadas a procedimiento remoto RPC para comunicarse con Nuve a través de un bus de mensajes sobre el que se ha implementado esta funcionalidad.

Los métodos públicos de los que dispone Erizo Controller son los siguientes:

- getUsersInRoom: obtiene los usuarios conectados a una sala determinada consultando la lista de usuarios.

- deleteRoom: elimina una sala con todo lo que ello implica. Cerrar las conexiones de los clientes que están publicando o suscritos a Streams de la sala y notificándolo a los clientes.

3.4 La configuración de Licode

Al ejecutar los diferentes módulos que hacen falta para el funcionamiento de un servicio basado en Licode y que se han explicado en esta descripción de su arquitectura son necesarios algunos parámetros de configuración. Estos parámetros son utilizados por los diferentes módulos para establecer algunas configuraciones en sus componentes. Se describen en un fichero de configuración que será accesible por todos ellos.

En la Tabla 10 pueden observarse estos parámetros junto con una pequeña descripción de su funcionalidad.

Componente	Parámetro	Descripción
RabbitMQ	host	Host en el que corre el servicio
	port	Puerto en el que corre el servicio
Nuve	dataBaseURL	URL de acceso a la base de datos MongoDB
	superServiceID	Identificador del súper servicio de Nuve
	superServiceKey	Clave del súper servicio de Nuve
Erizo Controller	stunServerURL	URL del servidor de STUN de los clientes
	warning_n_rooms	Número de salas warning en un Erizo
	limit_n_rooms	Número límite de salas en un Erizo
	interval_time_KA	Tiempo de intervalo de mensajes Keep Alive
Cloud Provider	name	Proveedor de Cloud donde corren los Erizos
	host	Identificador de la zona
	accessKey	Clave de acceso al proveedor
	secretAccesKey	Clave secreta de acceso al proveedor

Tabla 10. Configuración Licode

4 Despliegue en el Cloud

Como se ha explicado en la sección anterior, la arquitectura de Licode está muy orientada a su despliegue en el Cloud. Cuando el número de usuarios en el sistema es elevado es necesario arrancar varias instancias de Erizo o MCUs ya que es este módulo el que soporta una mayor carga computacional. Pero además este nivel de carga puede variar de forma dinámica durante el transcurso de una sesión multimedia.

Los requisitos de este módulo casan muy bien con el modelo de Cloud Computing ya que, de acuerdo con la definición NIST [P. Mell et al. 2009] provee características de *On- demand self-service, Broad network access, Resource pooling, Rapid elasticity* y *Measured service.*

En esta sección van a analizarse las oportunidades que ofrece el despliegue de una MCU en una infraestructura Cloud y que podrán aplicarse directamente a una MCU como Erizo. No obstante este despliegue implica también ciertos retos a los que hay que enfrentarse y que también se explicarán en esta sección.

Cabe destacar que la investigación realizada al respecto se encuentra ya publicada en un artículo de investigación [Alvaro Alonso et al. 2013] presentado en el congreso internacional CLOUD COMPUTING, International Conference on Cloud Computing, GRIDs, and Virtualization.

4.1 Oportunidades

Un componente MCU puede requerir diferentes características de computación en función del número de participantes, las condiciones de la sesión (grabación, redirección, transcodificación, composición) y la posición física de los participantes. Además estas condiciones pueden variar de forma dinámica durante la sesión multimedia. En una infraestructura Cloud la sesión va a poder adaptarse de forma sencilla y dinámica a las variaciones de este tipo de condiciones en función de los requisitos de cada momento.

El modelo definido por NIST y citado previamente ilustra estas ventajas que se explicarán a continuación:

. *On-demand self-service:* Los usuarios pueden disponer de capacidades de computación (CPU, ancho de banda, almacenamiento, etc.) según lo necesiten.

- *Broad network access:* Estas capacidades están disponibles en la red en diferentes posiciones y son servidas a través de mecanismos estándar.
- *Resource pooling:* El Cloud sigue un modelo *multi-tenant* por el cual pueden asignarse recursos a diferentes usuarios.
- *Rapid elasticity:* Las capacidades pueden ser provistas y liberadas de forma automática para adaptarse a la demanda de los usuarios.
- *Measured service:* Los recursos son controlados de forma automática y monitorizados y reportados por mecanismos de medida.

4.1.1 Escalabilidad a la demanda de usuarios

Los sistemas multimedia ofrecen a sus usuarios la posibilidad de unirse a una conferencia antes y durante la sesión. Ellos pueden además abandonar la sesión mientras está en curso. Dependiendo del tipo de sesión esta variación en el número de usuarios puede ser más o menos frecuente.

Un alto número de usuarios normalmente significa más ancho de banda, consumo de memoria y CPU. En otras palabras, una MCU demandará más o menos capacidades a su infraestructura de computación en función del número de usuarios conectados a una sesión.

En un entorno tradicional el proveedor debería preparar previamente las máquinas físicas que dedicaría al sistema y éstas deberían ser suficientes para poder asumir los picos de demanda que se produjeran. No obstante esta solución implica desperdiciar una alta cantidad de recursos cuando la demanda por parte de los usuarios sea pequeña.

En un entorno Clod, por el contrario, el proveedor puede proveer de forma dinámica máquinas virtuales y liberarlas cuando no sean necesarias. Esto se suele realizar normalmente encendiendo o apagando estas máquinas virtuales. dependiendo de los recursos necesarios, en relación a los participantes o usuarios que estén conectados en cada momento.

Esto puede también conseguirse incrementando de forma dinámica el rendimiento de una única máquina. Podemos por ejemplo incrementar el nivel de CPU o memoria de una máquina virtual que ya está corriendo. El ancho de banda disponible a la salida y a la entrada puede también modificarse variando el tamaño de las máquinas virtuales. Por ejemplo Amazon EC2 [Amazon] permite diferentes tipos de máquinas virtuales con diferentes capacidades de rendimiento.

4.1.2 Escalabilidad a los requisitos de la sesión

La operación de una MCU también depende del tipo de sesión que esté gestionando. Cada tipo de funcionalidad o tarea que desarrolle requiere diferentes capacidades de computación: redireccionamiento de flujos multimedia, composición de vídeos, transcodificación, grabación de las sesiones, etc.

Un componente MCU básico solamente redirecciona flujos de un participante al resto y requiere bajos niveles de computación. Sin embargo estos niveles (como el uso de memoria y CPU) aumentan considerablemente si la MCU realiza tareas avanzadas. La necesidad de realizar estas tareas puede variar de forma dinámica durante el transcurso de una sesión dependiendo de factores como el número de usuarios, el tamaño de los vídeos generados, los códecs que manejan los usuarios en sus dispositivos, etc.

Por ejemplo un alto número de usuarios puede forzar a la MCU a realizar una composición de vídeos en uno solo a partir de los de un conjunto de usuarios. Por otro lado en escenarios en los que los usuarios se conectan desde diferentes dispositivos, la MCU deberá adaptar los flujos que envía a cada uno de ellos en función de ciertas características de los dispositivos como el tamaño de la pantalla, la conexión de red, etc. Por último la MCU puede tener que grabar la sesión multimedia o parte de ella para realizar una reproducción posterior.

Entornos virtualizados en sistemas Cloud ayudan a la MCU a adaptarse a las variaciones de los requisitos de estas funcionalidades. Como en el caso anterior podemos encender una nueva máquina para realizar nuevas tareas y apagarla cuando no sea necesaria. O añadir capacidad a las máquinas que ya se encuentran corriendo en la infraestructura cuando sea necesario.

Otra posibilidad que ofrece el Cloud es la de configurar diferentes tipos de máquinas virtuales dependiendo de las tareas que vayan a desempeñar. En la Figura 4 se puede observar un ejemplo en el que la máquina encargada de redireccionar los flujos multimedia necesita un alto nivel de CPU y memoria. Por otro lado la máquina encargada de grabar la sesión consumirá poca memoria y CPU si recibe un único flujo de vídeo con la composición de la sesión.

Resumiendo, el despliegue de una MCU en una infraestructura Cloud hace que sea sencillo y dinámico gestionar la configuración de diferentes tipos de máquinas virtuales adaptando sus características a las necesidades del escenario concreto. Así podremos proveer un servicio adaptativo que utilizará de manera eficiente los recursos disponibles reduciendo constes y mejorando el rendimiento del sistema.

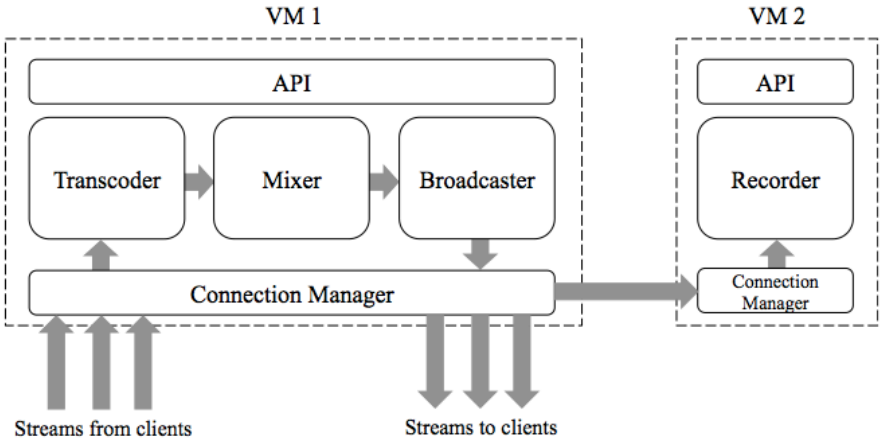

Figura 4. Varias MCUs

4.1.3 Flexibilidad geográfica

Otro factor crítico en las aplicaciones de tiempo real que afecta directamente a la experiencia de usuario es la latencia de las paquetes que viajan entre los usuarios conectados a una sesión multimedia. La latencia normalmente depende de la distancia geográfica entre los clientes. En los escenarios con un dispositivo MCU todos los flujos multimedia atraviesan la máquina en la que ésta está corriendo por lo tanto resulta crucial la posición en la que se encuentra.

Gracias a un sistema basado en el Coud podemos correr MCUs en diferentes localizaciones geográficas conectando cada una de ellas con los usuarios que están utilizando el servicio en cada región. Por ejemplo, en la fecha en que se escribe este documento, Amazon EC2 provee centros de datos en North Virginia, Oregon, North California, Ireland, Singapore, Tokyo, Sydney y Sao Paulo, y Rackspace [Rackspace] en Texas, Illinois, Vancouver, Hong Kong, London y Slough, UK.

Estos proveedores además permiten la interconexión de MCUs en diferentes regiones por lo que pueden ofrecerse sesiones en todo el mundo conectando usuarios a la MCU más cercana e interconectando las MCUs entre ellas. En la Figura 5 puede observarse un ejemplo de esta distribución.

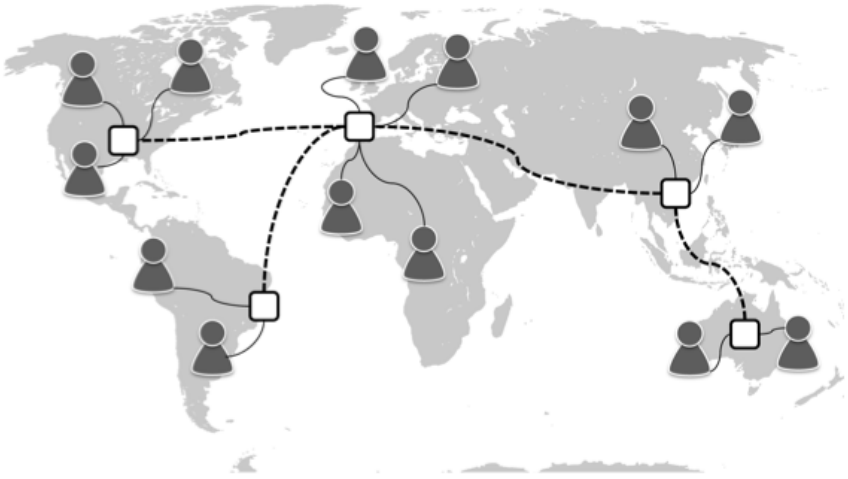

Figura 5. Distribución geográfica de MCUs

4.2 Retos

Para realizar un despliegue que aproveche todas las posibilidades que ofrece el Cloud y que han sido descritas en la sección anterior es necesario realizar un diseño minucioso y teniendo en cuenta todas las características del sistema que manejamos. En esta sección se analizan todos los retos que plantea este problema y se esbozan posibles soluciones que habrá que desarrollar con más detalle en futuros trabajos de investigación.

Además se proponen ciertas soluciones de escalabilidad que se salen de as aproximaciones generales que suelen plantearse como por ejemplo las expuestas en [L. M. Vaquero et al. 2011].

4.2.1 Caracterización del sistema

Caracterizar el rendimiento de la MCU es el primer paso para conseguir un despliegue efectivo en el Cloud. Dependiendo de la tarea (transcocificar, grabar, etc.) que vaya a realizarse en una MCU el hardware y ancho de banda necesario varian significativamente. Midiendo el rendimiento de los dispositivos en entornos conocidos y acotaros podemos extrapolar el comportamiento a otros escenarios para aproximar las características que serán necesarias en cada momento. Así sabremos la cantidad de CPU, memoria o ancho de banda que serán necesarias en las futuras sesiones.

Una vez completada la caracterización del sistema es interesante encontrar relaciones entre los recursos físicos y recursos de más alto nivel como el número de usuarios o el número de salas activas en la sesión.

Encontrando esta correlación podremos simplificar el trabajo a la otra de realizar la monitorización y resultará muy útil para las tareas que se explicarán en secciones siguientes, con el escalado horizontal y vertical. Sabiendo la incidencia de cada nuevo usuario conectado combinado con la monitorización continua de los recursos consumidos por la MCU podremos reaccionar de forma efectiva a los requisitos y los cambios en la demanda del sistema. Por supuesto esto implica conocer el efecto que un nuevo usuario conectado tiene sobre las capacidades y requisitos del sistema completo y cada instancia.

No obstante, hay un reto impuesto por el despliegue de un sistema conocido en el Cloud. Hay mucha literatura [O. Tickoo et al. 2010] [Y. Koh et al. 2007] escrita en cuanto a las posibles interferencias entre diferentes máquinas virtuales corriendo en el mismo host del proveedor así como posibles formas de caracterizar el problema [A. V. Do et al. 2011]. No obstante en el marco de esta investigación se asume que estas interferencias no van a ser relevantes a la hora de la caracterización de un sistema de nuestras características.

A continuación se explica un ejemplo de caracterización del sistema realizado en el marco de esta investigación y utilizando la MCU Erizo corriendo en una máquina del proveedor Amazon EC2. Se han diseñado dos escenarios que son los más comunes en sistemas de videoconferencias: un streaming de vídeo en tiempo real y una videoconferencia multiusuario. En ambos sistemas se ha dealizado una monitorización del consumo de memoria y ancho de banda así como del uso de ancho de banda entrante y saliente.

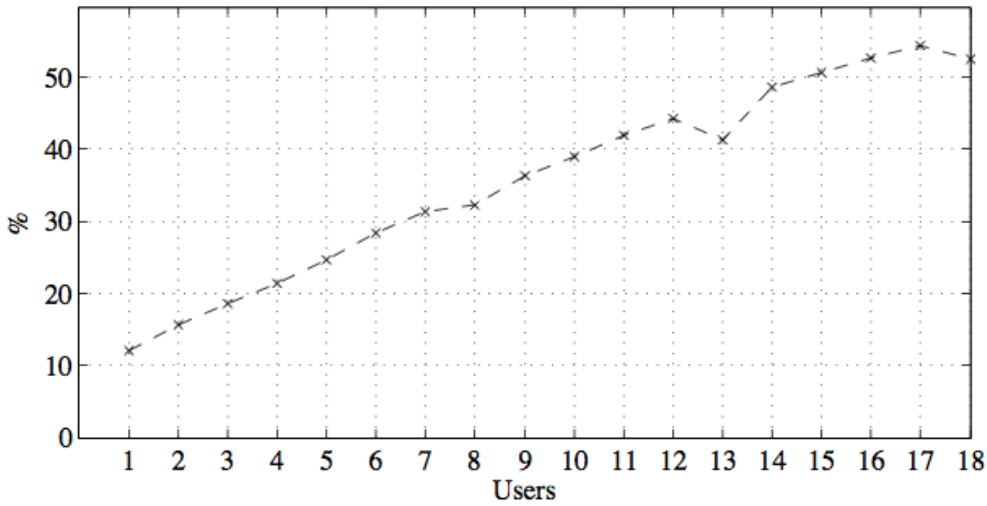

Figura 6. Uso de CPU en Streaming

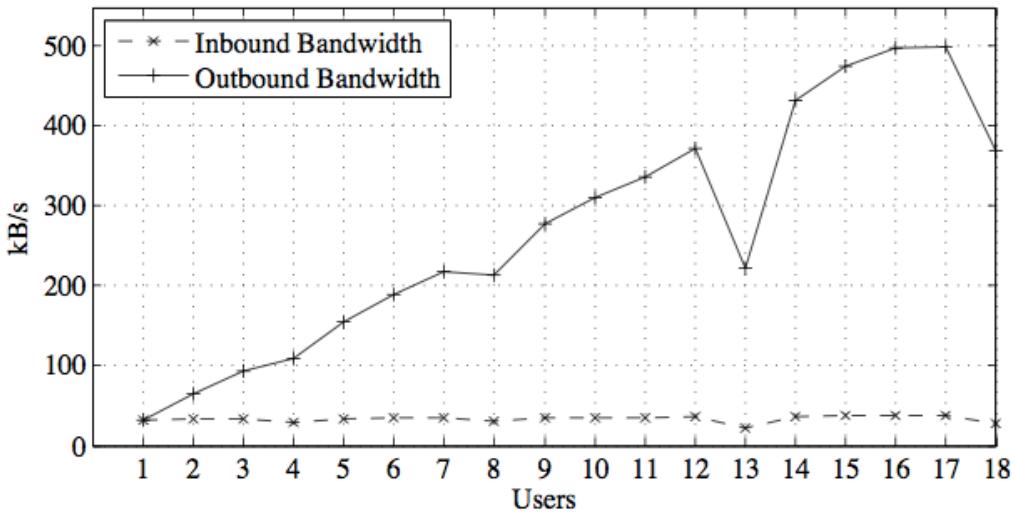

Figura 7. Uso de ancho de banda en Streaming

En el primer escenario, streaming en vivo, uno de los clientes está publicando su flujo de audio y vídeo en la sesión y los clientes que se suscriben a él van añadiéndose de forma gradual. En la Figura 6 podemos observar como el uso de CPU en la MCU aumenta de forma lineal con el incremento de usuarios que se suscriben al streaming. Esto ocurre porque el estándar WebRTC, como ya se ha explicado, utiliza SRTP para la transmisión de paquetes y por lo tanto la MCU tiene que desproteger y volver a proteger los paquetes para realizar la transmisión desde el cliente que publica hacia los que se suscriben.

También podemos observar en la Figura 7 como el ancho de banda entrante es constante durante toda la sesión y el saliente aumenta también de forma lineal debido a que cada nuevo cliente conectado implica un nuevo flujo de salida. Acerca de a memoria utilizada aumenta también de forma lineal pero con una variación mínima durante la sesión (de unos 10 MB). Finalmente puede observarse una pequeña anomalía cuando se conecta el usuario número 8 y el número 13 que probablemente sea debida a un error en el rendimiento del cliente que publica los datos, que también corre en una máquina de Amazon EC2.

En el segundo escenario, la videoconferencia multiusuario, cada usuario que se conecta a la sesión publica su flujo de audio y vídeo y además se suscribe al resto de usuarios que estaban conectados previamente. Se ha establecido un límite de seis usuarios ya que es el número que comúnmente se utiliza en este tipo de sesiones.

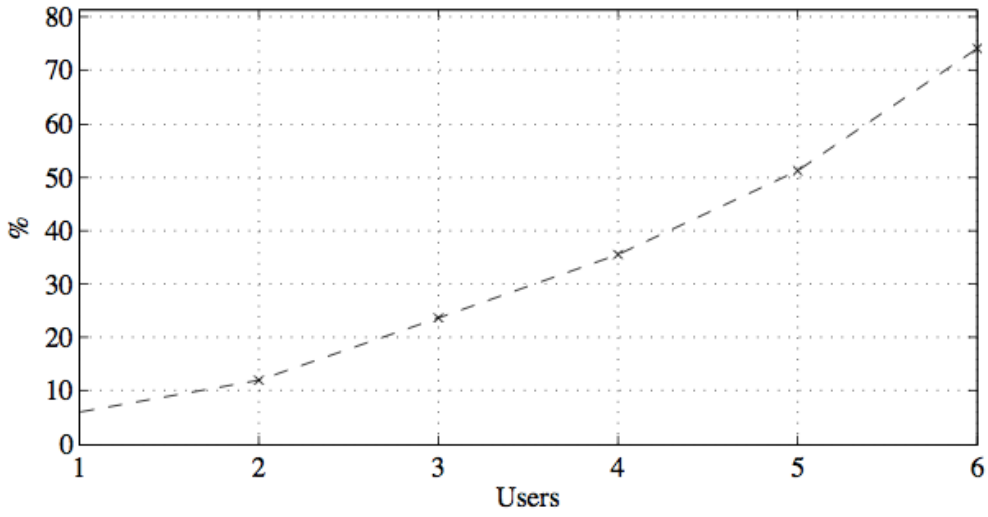

Figura 8. Uso de CPU en Videoconferencia

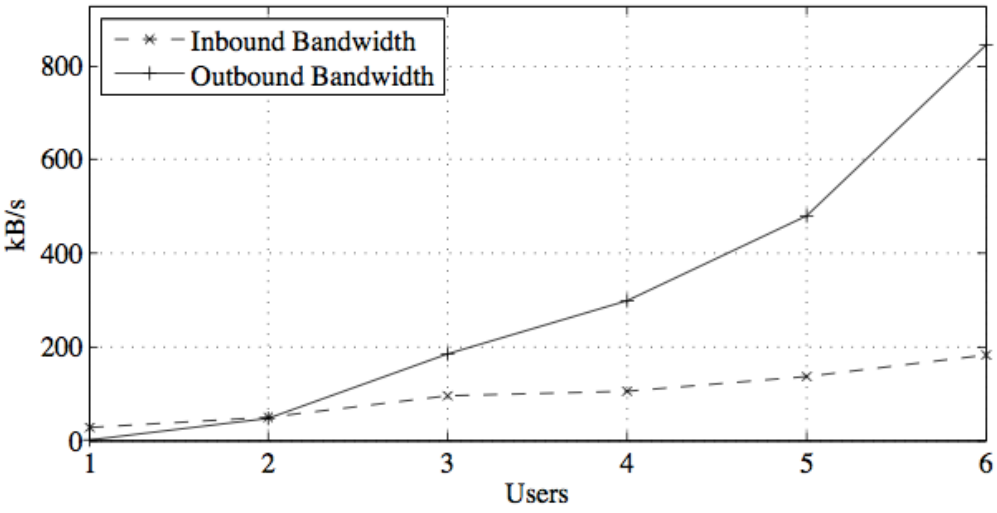

Figura 9. Uso de ancho de banda en Videoconferencia

En la Figura 8 y la Figura 9 podemos observar como en este caso el consume de ancho de banda entrante aumenta linealmente con el incremento de usuarios en la sesión debido al hecho de que cada nuevo usuario publica su flujo en el sistema. Sin embargo el ancho de banda saliente y el consumo de CPU se incrementan de forma exponencial debido a que por cada nuevo usuario la MCU debe redirigir el nuevo flujo al resto de usuarios. Por lo tanto el número de flujos de salida aumentará siguiendo la siguiente ecuación:

$$N = n\ (n-1)$$

Donde n es el número de usuarios en la sala. El uso de memoria en este escenario también aumenta de forma exponencial pero de nuevo con una variación insignificante para este estudio.

4.2.2 Escalabilidad hacia arriba

Cuando los recursos disponibles en una sesión alcanzan su límite podemos aprovechar las características del Cloud aumentando estos recursos para seguir ofreciendo un servicio con la calidad adecuada. Para ello existen dos tipos de escalabilidad, horizontal y vertical.

El escalado horizontal consiste en añadir nuevas máquinas virtuales a las ya arrancadas en el sistema y el escalado vertical en aumentar las características de las máquinas que ya están corriendo.

En una MCU ambos métodos tienen sus ventajas. Si los nuevos recursos se necesitan para hacer una nueva tarea en la sesión, como por ejemplo, grabar una sesión multimedia, la escalabilidad horizontal puede ser más beneficiosa. Sin embargo si los recursos se necesitan porque ha habido un incremento del número de usuarios en la sesión puede ser mejor realizar un escalado vertical para que la sesión siga gestionándose por la misma máquina virtual.

Esto nos dará más facilidades en caso de que sea necesario escalar hacia abajo en el futuro como se explicará en el siguiente apartado. Suele ser en general más conveniente tener todos los usuarios de una sala en la misma MCU. Sin embargo, no todos los proveedores de Cloud públicos ofrecen la posibilidad de realizar un escalado vertical.

Por lo tanto un primer reto importante relacionado con el escalado es el de elegir qué tipo va a utilizarse cuando en el sistema se requiere un mayor nivel de recursos para gestionar una sesión.

Por otro lado ambos tipos de escalado implican una latencia desde que se realiza la acción de aumentar los recursos hasta que éstos están disponibles para su uso. Para lograr una buena experiencia de usuarios es fundamental que no haya ningún tipo de interrupción en las comunicaciones y por ello es necesario anticiparse a los picos de demanda que pueda haber en el sistema para actuar en consecuencia con un tiempo que garantice la transparencia frente al usuario.

Una primera aproximación para solucionar el problema es la de utilizar algoritmos que, basándose en una monitorización del sistema, calculen cuándo va a ser necesario incorporar más recursos a las máquinas virtuales mediante cualquiera de los dos tipos de escalabilidad explicados anteriormente. La MCU deberá monitorizar constantemente el estado del sistema analizando los diferentes factores mostrados anteriormente. Si establecemos los límites del sistema y la correlación con el número de usuarios y salas será sencillo determinar cuando es necesario añadir recursos al sistema con la antelación necesaria. Un ejemplo de un sistema parecido puede observarse en [F. Galán et al. 2009].

Pero podemos ir un poco más allá y utilizar modelos predictivos para anticiparse a los cambios en los requisitos de las MCUs basándonos en el análisis de datos previos. Con este tipo de modelos podemos analizar patrones de comportamiento del sistema para predecir lo que sucederá en un momento determinado. Estos patrones pueden obtenerse de dos formas fundamentales. Bien utilizando el comportamiento previo del propio sistema o el de sistemas similares. La primera opción es la óptima pero no siempre podemos disponer de esos datos por lo que puede ser necesario acudir a la segunda.

En [E. Caron et al. 2010] puede observarse un buen punto de partida en el cual el problema se resuelve con un algoritmo que predice el uso de recursos utilizando un matching de patrones.

4.2.3 Escalabilidad hacia abajo

De la misma forma que durante una sesión puede ser necesario incrementar los recursos disponibles puede ocurrir que en un momento dado un pico de demanda haya desaparecido y estemos ofreciendo más recursos de los necesarios. Como se ha mostrado en la sección anterior hay diferentes métodos de incrementar los recursos de un módulo MCU. En el caso del escalado hacia abajo ocurre lo mismo peor en sentido contrario.

Por lo tanto el primer reto que plantea el escalado hacia abajo es similar al caso anterior. Cuando detectamos que según la demanda del sistema estamos ofreciendo más recursos de los necesarios debemos elegir la mejor forma de disminuirlos. Podemos realizar un escalado vertical y reducir la capacidad de computación de las máquinas que ya están corriendo o realizar un escalado horizontal reduciendo el número de máquinas arrancadas.

Pero en el segundo caso, escalado horizontal hacia abajo, se plantean dificultades adicionales. Debe tenerse en cuenta que en la máquina que vamos a apagar muy probablemente está realizándose la gestión de usuarios, flujos y salas multimedia por lo que deberemos distribuir estos elementos entre el resto de máquinas que seguirán arrancadas. Esta distribución no es un problema trivial.

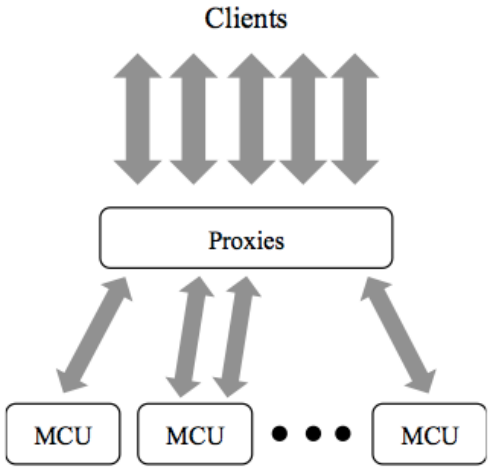

Figura 10. Proxy

Un cliente que está participando en una sesión está enviando y recibiendo varios flujos multimedia desde y hacia la MCU. Si esta MCU va a ser apagada es necesario redirigir estos flujos a otra MCU y esta redirección debe realizarse de forma transparente para el usuario. Además para optimizar el uso de recursos debe realizarse una correcta distribución de los usuarios y las salas de videoconferencia procurando agrupar estos en las mismas MCUs.

Una posible solución a este problema, mostrado en la Figura 10, es la de incluir un proxy entre los clientes y el módulo MCU. De esta forma cuando una máquina va a ser apagada el proxy comenzará a duplicar los flujos entre la antigua MCU y la nueva. Cuando los flujos estén preparados el proxy cambiará en envío y recepción al cliente de la antigua MCU a la nueva y en ese momento podrá apagarse la antigua.

4.2.4 Distribución geográfica

De la mano de la flexibilidad que da la distribución geográfica permitida por los proveedores de Cloud viene el reto de localizar de forma óptima las instancias de MCUs para obtener el mejor servicio posible. a decisión puede ser trivial cuando todos los usuarios están localizados en el mismo continente o zona del proveedor de Cloud.

Pero decidir cómo actuar cuando los usuarios están distribuidos en distintos lugares puede determinar la calidad de la sesión.

Para tomar esta decisión debe tenerse en cuenta el número de usuarios en cada región geográfica pero también a calidad de sus conexiones. Para caracterizar las conexiones entre las regiones es interesante realizar medidas de ancho de banda, jitter, pérdidas de paquetes y RTT. Probando la conexión de cada usuario con las diferentes regiones podemos decidir dónde la calidad será mejor.

Como puede comprobarse en [J. Cervino et al. 2011] las conexiones de red entre instancias de Amazon en diferentes regiones rinden mejor que las conexiones de Internet medias. Debe tenerse esto en cuenta a la hora de realizar el diseño y despliegue del sistema.

5 Resultados

En esta sección se realiza una descripción de los diferentes resultados obtenidos o que se van a obtener en relación a Licode. Esto incluye desarrollo de proyectos del departamento y el grupo de investigación, líneas de investigación y la comunidad creada en torno al proyecto.

El sitio web en el que se encuentra publicada toda la información al respecto del proyecto es

http://lynckia.com/licode

5.1 El proyecto de código libre

Licode es un proyecto de código libre publicado en la página del grupo de investigación GING en la plataforma GitHub. Puede encontrarse en:

https://github.com/ging/licode

Algunos datos en la fecha en que se escribe este documento en esta plataforma son los siguientes:

- 76 stars
- 19 forks
- 496 commits
- 3 pull requests
- 8 branches

Por lo tanto cualquier persona puede descargarse la plataforma, instalarla en su propia máquina en unos minutos y ofrecer un servicio de comunicaciones en tiempo real en la web. La información para realizar esta instalación se encuentra en

http://lynckia.com/licode/install.html

5.2 La comunidad

En torno al proyecto de código libre se ha creado una comunidad de usuarios que cada día se descargan el proyecto y utilizan con diferentes fines realizando por tanto pruebas en diferentes entornos y con distintos casos de uso. Se ha creado una lista de correo para consultar y discutir temas relacionados con Licode. Cualquier desarrollador interesado en el proyecto puede suscribirse a esta lista de correo y participar en ella en:

lynckia@googlegroups.com

Algunos datos acerca de esta lista son los siguientes:

- 87 miembros
- Más de 50 discusiones
- 4/5 mensajes diarios

5.3 Proyectos

El proyecto Licode se está utilizando en algunos proyectos de investigación del departamento. En el momento en que se escribe este documento son los siguientes:

5.3.1 FI-WARE

FI-WARE (Plataforma central del Internet del futuro, del inglés Future Internet Core Platform) es un proyecto europeo que se dedica a la creación de una nueva infraestructura de servicios basada en componentes básicos genéricos y reutilizables denominados Capacitadores Generales (Generic Enablers, GE) para la Internet del Futuro.

5.3.2 Telefónica España MSC5

El proyecto MSC5 es un proyecto realizado por el grupo de investigación GING para Telefónica España en el ámbito de innovación. Consiste en el desarrollo de un

servicio multi dispositivo de televisión social, videovigilancia, teleasistencia y juegos que funcionará utilizando el nuevo estándar de comunicaciones web HTML5 WebRTC.

5.4 Investigación

Durante este año se ha realizado una contribución publicando un artículo [Alvaro Alonso et al. 2013] en el congreso internacional IARIA CLOUD COMPUTING 2013. The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization.

En esta contribución se realiza el análisis descrito en la sección 4 de este documento, en relación a los retos y oportunidades que plantea el despliegue del componente Erizo en una infraestructura Cloud.

Además fruto del trabajo realizado hasta el momento surgen más líneas de investigación relacionadas con este despliegue que se explicarán a continuación en la sección dedicada a trabajos futuros.

Durante este periodo también se ha realizado una contribución relacionada con el campo de la videoconferencia en la web junto a Pedro Rodríguez, Joaquín Salvachúa y Javier Cerviño. En concreto acerca de un algoritmo de calidad de servicio en la plataforma VaaS explicada en el Estado del Arte de este documento. La contribución va a publicarse en el congreso internacional WCCIT'13 World Congress on Computer and Information Technologies, en concreto en ICMMP'2013 - The 2013 International Conference on Multi Media Processing.

6 Conclusiones

Licode es un proyecto software que provee comunicaciones en tiempo real en los navegadores de tal forma que puede instalarse en cualquier infraestructura orientado a ser escalable en el Cloud. Por lo tanto cumple los requisitos previstos en este Trabajo Fin de Máster. Como resultado del trabajo y en colaboración con los compañeros del grupo se han cumplido los siguientes objetivos:

- Se ha diseñado una arquitectura del sistema orientada al Cloud que permite proveer servicios avanzados de comunicaciones en tiempo real en los navegadores web tales como videoconferencia entre muchos usuarios, streaming en tiempo real, juegos online, etc.

- Se ha desarrollado cada uno de los módulos utilizando las tecnologías más adecuadas para cada uno de los casos.

- Se ha realizado un trabajo en equipo y en grupos de desarrollo de proyectos adecuándose a los requisitos concretos de cada uno de ellos y a los plazos establecidos.

- Se ha publicado el proyecto como código libre consiguiendo crear una comunidad en torno a él que proporcione feedback y mejore el rendimiento y funcionalidades del sistema.

- Se ha realizado un estudio de investigación en torno al despliegue del sistema en infraestructuras Cloud que además se enfoca a la continuación con nuevas vías de investigación.

6.1 Trabajo futuro

En cuanto al lado técnico del sistema, el trabajo futuro consiste en ampliar y evolucionar las funcionalidades y capacidades del sistema añadiendo tareas de grabación y mezcla de vídeos en el módulo Erizo. Esto implica también realizar mejoras en el resto de módulos del sistema para ser compatibles y ofrecer distintos servicios a los usuarios.

En relación al despliegue del sistema en el Cloud el primer paso es continuar con el trabajo de investigación proponiendo algoritmos de escalado y monitorización así como modelos predictivos. Además estas nuevas capacidades se irán incluyendo en el sistema una vez estén probadas y preparadas para su despliegue.

En este trabajo se incluye también el diseño y despliegue de proxies entre los clientes y las MCUs para realizar de forma efectiva las tareas de escalado hacia abajo. Gracias a todo esto se conseguirá un despliegue efectivo que ahorre costes y haga un consumo eficiente de los recursos.

Por otro lado el trabajo futuro incluye el mantenimiento de la comunidad y la incorporación de nuevas funcionalidades a la arquitectura general como la posibilidad de realizar comunicaciones peer – to – peer.

Además se pretende que el sistema Licode sea un toolkit webRTC de tal forma que puedan desarrollarse módulos externos compatibles para realizar tareas como procesamiento de datos, realidad aumentada, etc.

Bibliografía

[Alvaro Alonso et al. 2013] Alvaro Alonso, Pedro Rodr iguez, Joaquin Salvachua, Javier Cerviño. "Deploying a Multipoint Control Unit in the Cloud- Opportunities and Challenges". 2013, CLOUD COMPUTING, International Conf. on Cloud Computing, GRIDs, and Virtualization.

[Amazon] Amazon AWS. http://aws.amazon.com [retrieved: March, 2013].

[A. Bergkvist et al. 2012] 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].

[A. V. Do et al. 2011] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in Cloud Computing (CLOUD), 2011 IEEE International Conference on, July 2011, pp. 660 –667.

[C. Jennings et al. 2013] Cullen Jennings, Ted Hardie and Magnus Westerlund, "Real – Time Communications fot the Web2" IEEE Communications Magazine. Abril 2013.

[E. Caron et al. 2010] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference, December 2010, pp. 456 – 463.

[F. Galán et al. 2009] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Va- quero, "Service specification in cloud environments based on extensions to open standards," in Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE, ser. COMSWARE '09, New York, NY, USA, 2009, pp. 19:1–19:12.

[J. Cervino et al. 2011] J. Cervino, P. Rodriguez, I. Trajkovska, A. Mozo, and J. Salvachua, "Testing a cloud provider network for hybrid p2p and cloud streaming architectures," in Cloud Computing (CLOUD), 2011 IEEE International Conference, July 2011, pp. 356 –363.

[L. M. Vaquero et al. 2011] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," SIGCOMM Comput. Commun. Rev. vol 41, num 1, January 2011, pp. 45 –52.

[M. Baugher et al. 2004] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The secure real-time transport protocol (srtp)," Internet Engineering Task Force, March 2004, updated by RFC 5506.

[Mell et al. 2011] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology*. NIST Special Publication, vol. 145, no. 6, page 7, 2011.

[OAuth] OAuth: http://oauth.net/

[O. Tickoo et al. 2010] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: challenges and approaches," SIGMETRICS Per- form. Eval. Rev. January, 2010, vol. 37, no. 3, pp. 55 – 60.

[P. Mell et al. 2009] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf [retrieved: March, 2013], 2009.

[P. Rodríguez et al. 2012] P. Rodriguez, J. Cervino, I. Trajkovska, and J. Salvachua, "Advanced videoconferencing services based on webrtc," IADIS International Conferences Web Based Communities and Social Media 2012 and Collaborative Technologies 2012, pp. 180–184.

[Rackspace] Rackspace. http://www.rackspace.com [retrieved: March, 2013].

[S. Loreto et al. 2012] S. Loreto and S. Romano, "Real-time communications in the web: Issues, achievements, and ongoing standardization efforts, september- october, 2012," Internet Computing, IEEE, vol. 16, no. 5, pp. 68 –73.

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

[Y. Koh et al. 2007] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium, April 2007.